Tail-GNN: Tail-Node Graph Neural Networks

Zemin Liu
Singapore Management University
Singapore
zmliu@smu.edu.sg

Trung-Kien Nguyen
Singapore Management University
Singapore
tknguyen@smu.edu.sg

Yuan Fang
Singapore Management University
Singapore
yfang@smu.edu.sg

ABSTRACT

The prevalence of graph structures in real-world scenarios enables important tasks such as node classification and link prediction. Graphs in many domains follow a long-tailed distribution in their node degrees, *i.e.*, a significant fraction of nodes are *tail* nodes with a small degree. Although recent graph neural networks (GNNs) can learn powerful node representations, they treat all nodes uniformly and are not tailored to the large group of tail nodes. In particular, there is limited structural information (*i.e.*, links) on tail nodes, resulting in inferior performance. Toward robust tail node embedding, in this paper we propose a novel graph neural network called Tail-GNN. It hinges on the novel concept of *transferable neighborhood translation*, to model the *variable ties* between a target node and its neighbors. On one hand, Tail-GNN learns a neighborhood translation from the structurally rich *head* nodes (*i.e.*, high-degree nodes), which can be further transferred to the structurally limited tail nodes to enhance their representations. On the other hand, the ties with the neighbors are variable across different parts of the graph, and a global neighborhood translation is inflexible. Thus, we devise a node-wise adaptation to localize the global translation w.r.t. each node. Extensive experiments on five benchmark datasets demonstrate that our proposed Tail-GNN significantly outperforms the state-of-the-art baselines.

CCS CONCEPTS

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

KEYWORDS

Graph neural networks, tail node embedding, transferable neighborhood translation

ACM Reference Format:
Zemin Liu, Trung-Kien Nguyen, and Yuan Fang. 2021. Tail-GNN: Tail-Node Graph Neural Networks. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '21), August 14–18, 2021, Virtual Event, Singapore.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3447548.3467276

KDD '21, August 14–18, 2021, Virtual Event, Singapore

ACM ISBN 978-1-4503-8332-5/21/08…$15.00
https://doi.org/10.1145/3447548.3467276

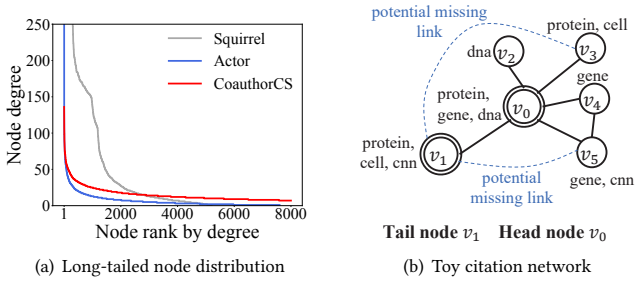
1 INTRODUCTION

Real-world objects often link together to form complex network structures, also known as graphs. For example, research papers reference each other to form a citation graph, and users interact with (such as click or buy) products to form an e-commerce network. The prevalence of graph structures draws active research on various graph-based tasks, such as node classification for categorizing papers into different topics in a citation graph, and link prediction for product recommendation in e-commerce networks.

Traditional machine learning on graphs focuses on feature engineering, a largely manual and costly process. The rise of graph representation learning enables automatic feature extraction by projecting graph nodes into a low dimensional vector space. On one hand, earlier graph embedding approaches [24, 27] employ local structures such as random walks and node proximities to constrain node embeddings. On the other hand, recent GNNs [8, 12, 31, 36] adopt a global message passing mechanism, hinging on a powerful neighborhood aggregation operator in which each node receives and aggregates messages (*i.e.*, node contents such as attributes) from its neighboring nodes recursively.

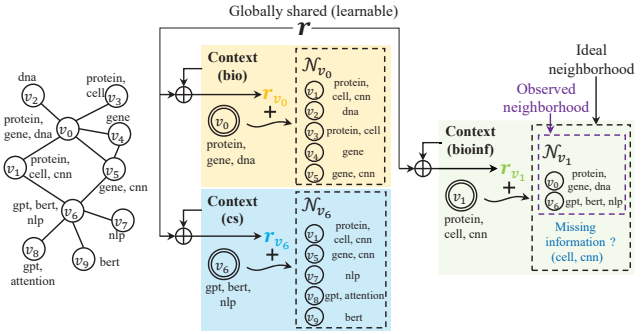
Our problem. Many real-world graphs are long-tailed, *i.e.*, the node degrees follow a power-law like distribution where a significant fraction of nodes have low degrees. As shown in Fig. 1(a), several benchmark graphs in a variety of domains exhibit a characteristic long tail. However, GNNs depend their performance heavily on the abundance of structural information. While the high-degree or so-called *head* nodes link to enough neighboring nodes for meaningful neighborhood aggregation, the low-degree or so-called *tail* nodes link to a small neighborhood that can be biased or underrepresented, leading to unsatisfactory performance [17]. Unfortunately, most state-of-the-art GNNs do not pay special attention to tail nodes, which can marginalize the group of tail nodes and present a major bottleneck in their performance. Thus, in this paper we investigate the problem of modeling tail nodes in graph neural networks, to learn robust tail node embeddings with limited structural connectivity.

Prior work. Recent works [29, 34] employ degree-specific transformations on nodes of different degrees. While they distinguish nodes based on their degrees, they aim to improve the overall performance and are not specifically designed to enhance the embeddings of the tail nodes, an especially challenging group due to their structural limitation. A more related study known as meta-tail2vec [17] proposes a two-stage framework for robust tail node embedding. In the first stage, a base embedding model (*e.g.*, DeepWalk [24]) is employed to produce initial node embeddings for all nodes; in the second stage, the embeddings of tail nodes are further refined by learning from the higher-quality embeddings of the head nodes. The main disadvantage of this two-stage approach is that the base



(a) Long-tailed node distribution (b) Toy citation network

Figure 1: Illustration of tail nodes.



(a) Toy network (b) Neighborhood translation for head nodes (c) Neighborhood translation for tail nodes

Figure 2: Illustration of neighborhood translation.

embedding model is decoupled from the refinement stage, implying that any inherent bias or noise in the base embedding model cannot be fundamentally corrected at the root. Thus, a more desirable approach is to directly design a new GNN that asserts robustness for the tail nodes in an end-to-end learning process.

Challenges and present work. In comparison to head nodes, tail nodes have a small neighborhood, which may potentially suffer from missing information. As shown in Fig. 1(b), the head node (paper) v_0 has sufficient neighborhood information matching its own attributes (*i.e.*, keywords "protein", "gene", "dna"). In contrast, the tail node v_1 has a smaller and biased neighborhood covering "protein" but missing "cell" and "cnn" that v_1 itself is focused on, due to potential missing links indicated in the figure. This would distort the neighborhood aggregation in GNNs. Thus, toward robust tail node embedding, we need to address the core issue of missing neighborhood information, which presents two challenges.

First, *how to uncover the missing neighborhood information for tail nodes?* Essentially, the exploitation of missing information should be underpinned by a unified mechanism across the graph, and yet allow for flexibility since nodes reside in diverse local contexts [16] on the graph. This leads to the second challenge: *how to localize the missing information for each tail node while maintaining the generality across nodes?*

To address these challenges, we propose a new graph neural network named Tail-GNN for robust tail node embedding, in order to close the structural gap between head and tail nodes. Tail-GNN boils down to the novel concept of *neighborhood translation*. As the basis of neighborhood aggregation in GNNs, each node is closely related to its neighbors. Consider the toy citation network in Fig. 2(a), where we take v_0 and v_6 as example head nodes (papers) and v_1 as an example tail node. Each head node tends to study a similar topic as its neighboring nodes, as reflected by their features: v_0 and its neighbors have similar biology keywords, whereas v_6 and its neighbors have similar computer science keywords. In other words, there exists a *relational tie* between each head node and its neighborhood. We model this relational tie as a translation operation, called *neighborhood translation*. As shown in Fig. 2(b), $\mathbf{r}_{v_i}$ represents the translation vector between the embeddings of the head node v_i and its neighbors set $\mathcal{N}_{v_i}$.

Subsequently, to answer the first challenge, we predict the missing neighborhood information for tail nodes by exploiting a *transferable* neighborhood translation. That is, we learn a globally shared neighborhood translation $\mathbf{r}$ from the head nodes v_0 and v_6 in Fig. 2(b), and then transfer it to the tail node v_1. The gap between the translated tail node (*i.e.*, its ideal neighborhood) and its observed neighborhood represents the missing neighborhood information as shown in Fig. 2(c). The predicted missing information can complement neighborhood aggregation of a tail node to correct the potential bias in its observed neighborhood. Moreover, to answer the second challenge, we tailor the shared neighborhood translation to each target node w.r.t. its local context. As shown in Fig. 2(b), the shared translation $\mathbf{r}$ is localized into $\mathbf{r}_{v_0}$ and $\mathbf{r}_{v_6}$ for the head node v_0 and v_6 to suit their contexts of biology and computer science, respectively. Similarly, the shared translation is localized to suit the tail node v_1 in Fig. 2(c). While the shared translation aims to capture the general pattern across the graph, the node-wise localization accounts for variations across diverse localities on the graph.

Contributions. To the best of our knowledge, this is the first end-to-end GNN model for robust tail node embeddings. To summarize, we make the following contributions. (1) We introduce a novel concept of transferable neighborhood translation to model the relational tie between a node and its neighboring nodes. (2) Hinged on this concept, we propose a novel model Tail-GNN to narrow the gap between head and tail nodes. (3) We conduct extensive experiments on five benchmark datasets. Our proposed Tail-GNN obtains state-of-the-art performance in comparison to a comprehensive suite of baselines.

2 RELATED WORK

Graph representation learning. Graph embedding [24, 27] has been effective in learning structure-preserving latent representations of a graph. More recently, graph neural networks (GNNs) [8, 12, 16, 22, 31, 36] have emerged as state-of-the-art approaches for graph representation learning, due to their ability of capturing both node contents and graph structures through the powerful operation of neighborhood aggregation in a message passing framework. For a more elaborate discussion on GNNs, we refer readers to a comprehensive survey [35].

Robust tail node embedding. Most existing robustness techniques for GNNs [3, 20, 32] only aim to enhance the overall robustness of all node representations, and do not specifically target

the most vulnerable tail nodes. Some recent GNN-based models consider degree-specific transformations on nodes of different degrees [29, 34], or use structural features including degrees to differentiate nodes of different roles [1], but they likewise focus on the overall performance. A closely related work is meta-tail2vec [17], which proposes a two-stage framework for robust tail node embedding. In the first stage, they apply a base embedding approach (*e.g.*, graph embedding or GNN) to produce the initial node embeddings; in the second stage, the embeddings of tail nodes are further refined by aggregating the well-learned embeddings of their neighboring head nodes. In natural language processing, to address tail word embedding, a series of similar two-stage refinement approaches such as Additive [13], a la carte [11] and Nonce2vec [9] have also been proposed. However, all these approaches separate the embedding and refinement processes, in which the initial embedding cannot benefit from the subsequent refinement to address the root of the problem, calling for an end-to-end approach.

Other long-tailed problems. Many studies have been devoted to long-tailed scenarios in other domains or problems. For example, in cold-start recommendation [4, 15, 18, 38], there is often a long tail of new items or users with limited historical interaction. However, they are not designed for GNNs, and may require additional side information such as brand/price correlation [4], session or behavior sequence [15, 38] and meta-paths [18]. Another common long-tailed problem exists in imbalanced classes where there is a long tail of instances in many small classes [10, 14, 26, 28, 39], or imbalanced features where many instances have sparse feature vectors [21]. While many of these studies also resort to the principle of transferring information from head to tail distributions, due to the different problem setting they cannot be directly applied to tail node embedding.

3 PRELIMINARIES

In this section, we first formulate the problem of robust tail node embedding. Next, we present a brief background on graph neural networks, which lays the foundation of our approach.

3.1 Problem formulation

Graph. A *graph* is denoted by $\mathcal{G} = \{\mathcal{V}, \mathcal{E}, \mathbf{X}\}$, where $\mathcal{V}$ is the set of nodes, $\mathcal{E}$ is the set of edges between the nodes, and $\mathbf{X} \in \mathbb{R}^{|\mathcal{V}| \times d_X}$ is the input feature matrix for all the nodes such that $\mathbf{x}_v$ represents the d_X-dimensional feature vector of node $v \in \mathcal{V}$.

For every node $v \in \mathcal{V}$, let $\mathcal{N}_v$ denote the set of neighboring nodes of v. $\mathcal{N}_v$ is also called the *neighborhood* of v, and its cardinality $|\mathcal{N}_v|$ is the *degree* of v. Moreover, let $\mathcal{V}_{\text{head}}$ and $\mathcal{V}_{\text{tail}}$ denote the set of *head* and *tail* nodes, respectively. For some threshold K, *tail* nodes are defined as nodes with a degree not exceeding K, *i.e.*, $\mathcal{V}_{\text{tail}} = \{v : |\mathcal{N}_v| \leqslant K\}$, whereas *head* nodes are the complement of tail nodes, *i.e.*, $\mathcal{V}_{\text{head}} = \{v : |\mathcal{N}_v| > K\}$. By their definitions, it is apparent that $\mathcal{V} = \mathcal{V}_{\text{head}} \cup \mathcal{V}_{\text{tail}}$ and $\mathcal{V}_{\text{head}} \cap \mathcal{V}_{\text{tail}} = \emptyset$. In this paper, we treat K as a predetermined hyperparameter. An alternative could be adopting a self-adaptive mechanism so that the threshold K fits each graph spontaneously, which is left for future work.

Problem. In this paper, we address the problem of graph representation learning, with a focus on enhancing the representations of structurally limited tail nodes. Formally, given a graph $\mathcal{G}$, the general goal of graph representation learning is to find a mapping $\psi : \mathcal{V} \to \mathbb{R}^d$ that can project each node $v \in \mathcal{V}$ to a d-dimensional vector space. Further to the general goal, we specifically emphasize on narrowing the structural gap between tail and head nodes, so that the quality of tail node embeddings, $\{\psi(v) : v \in \mathcal{V}_{\text{tail}}\}$, can be made more robust to match the quality of head node embeddings, $\{\psi(v) : v \in \mathcal{V}_{\text{head}}\}$.

3.2 Graph neural networks

Modern graph neural networks (GNNs) [35] resort to the key operation of neighborhood aggregation in a message passing framework. Specifically, each node receives and aggregates messages (*i.e.*, features or embeddings) from its neighboring nodes recursively in multiple layers. Formally, in the l-th layer of a GNN,

$$\mathbf{h}_v^l = \mathcal{M}(\mathbf{h}_v^{l-1}, \{\mathbf{h}_i^{l-1} : i \in \mathcal{N}_v\}; \theta^l), \tag{1}$$

where $\mathbf{h}_v^l \in \mathbb{R}^{d_l}$ is the d_l-dimensional embedding vector of node v in the l-th layer, and $\mathcal{M}(\cdot)$, parameterized by θ^l in the l-th layer, is the message passing function for neighborhood aggregation. In the input layer, $\mathbf{h}_v^0$ is given by the input node features, *i.e.*, $\mathbf{h}_v^0 \equiv \mathbf{x}_v$.

While our approach Tail-GNN is built upon GNNs, it is flexible to the choice of message passing function $\mathcal{M}$, ranging from mean pooling [12, 33] to complex neural networks [8, 31, 36].

4 CONCEPT: TRANSFERABLE NEIGHBORHOOD TRANSLATION

To enhance the representation learning of tail nodes, we propose a novel concept called *neighborhood translation*, based on which we further design a scheme of head-to-tail knowledge transfer.

4.1 Neighborhood translation

Generally, the tight structural connectivity between a node and its neighbors gives rise to a relational tie between them. In particular, a node tends to be similar to its neighboring nodes, an implicit assumption in GNNs and graph-based approaches in general. For instance, in Fig. 2(a), v_0 and its neighborhood are similarly described by biology keywords, whereas v_6 and its neighborhood are similarly described by computer science keywords.

Inspired by relational translation models in knowledge graph [2], we propose neighborhood translation to model the relational tie between a node v and its neighborhood $\mathcal{N}_v$ using a translation operation. Formally, let $\mathbf{h}_v$ denote the embedding vector of a head node v, and let $\mathbf{h}_{\mathcal{N}_v}$ denote the embedding vector of v's neighborhood $\mathcal{N}_v$, which can be obtained via a pooling over the embedding vectors of all the neighbors of v. The neighborhood translation on node v is formulated as

$$\mathbf{h}_v + \mathbf{r}_v \approx \mathbf{h}_{\mathcal{N}_v}. \tag{2}$$

Here $\mathbf{r}_v$ is the translation vector that can be "predicted" by a learnable model in order for Eq. (2) to approximately hold, which will be materialized in Sect. 5. Note that the traditional concept of translation on graphs [2, 30] regards each edge as a translation between two nodes to model relations, whereas in our work the translation happens at the neighborhood level to model the missing information in a neighborhood.

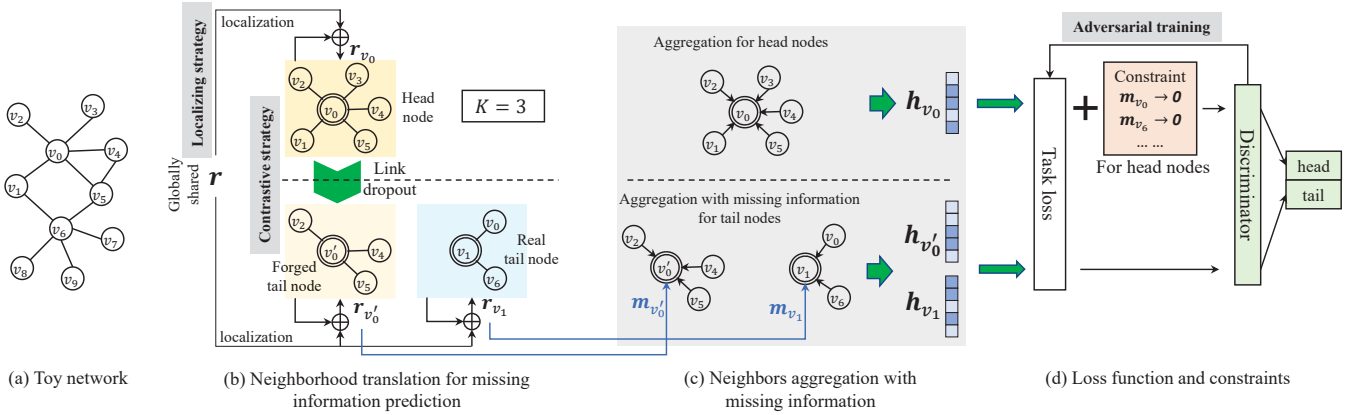


Figure 3: Overall framework of Tail-GNN.

4.2 Predicting missing neighborhood information by head-to-tail transfer

We attempt to uncover the missing neighborhood information of tail nodes by transferring the knowledge of neighborhood translation from head to tail nodes. In the following, we examine the divergence and commonality between head and tail nodes to enable transferable neighborhood translation.

Neighborhood of head nodes. As head nodes are well connected in the graph, we assume that their *observed neighborhoods* are complete and representative enough, so that the neighborhood translation naturally exist between a head node and its neighborhood. Thus, we can directly learn a model to predict the translation vectors in Eq. (2) for the head nodes. To be more clear, herein the observed neighborhood simply refers to the set of neighboring nodes explicitly given by the input graph.

Neighborhood of tail nodes. On the contrary, tail nodes are structurally limited due to a variety of reasons, such as an inactive user on a social network, a new product on a e-commerce platform, or a paper in a less visible journal. This results in a small observed neighborhood that can be potentially biased. In other words, the observed neighborhood for a tail node may not be representative enough for meaningful aggregation in GNNs. Thus, it becomes imperative to uncover the *missing neighborhood information* for tail nodes. Specifically, the missing information of a tail node $v \in \mathcal{V}_{\text{tail}}$, denoted by $\mathbf{m}_v$, is given by the difference between the embedding vectors of its *ideal neighborhood* $\mathcal{N}_v^*$ and observed neighborhood $\mathcal{N}_v$. That is,

$$\mathbf{m}_v = \mathbf{h}_{\mathcal{N}_v^*} - \mathbf{h}_{\mathcal{N}_v}. \tag{3}$$

Here the ideal neighborhood $\mathcal{N}_v^*$ refers to a latent, theoretical neighborhood containing not only the observed neighbors but also nodes that could have been linked to v. The relationship among the ideal and observed neighborhoods, as well as the missing neighborhood, is depicted in Fig. 2(c).

Predicting missing information. To compute Eq. (3), we need to first "predict" the representation of the unknown ideal neighborhood $\mathbf{h}_{\mathcal{N}_v^*}$. Specifically, we can leverage a unifying translation model on both head and tail nodes w.r.t. their ideal neighborhoods. On head nodes, we regard its observed neighborhood as ideal, and learn how to predict the translation vector $\mathbf{r}_v$ in Eq. (2); on tail nodes, we apply the prediction model for the translation vector to construct their ideal neighborhood, thereby transferring knowledge from head to tail nodes. That is, given a tail node $v \in \mathcal{V}_{\text{tail}}$, we predict the representation of its ideal neighborhood as $\mathbf{h}_{\mathcal{N}_v^*} = \mathbf{h}_v + \mathbf{r}_v$, where the translation vector $\mathbf{r}_v$ is predicted by the translation model learned from head nodes. We can subsequently predict the missing neighborhood information of the tail node v as follows:

$$\mathbf{m}_v = \mathbf{h}_v + \mathbf{r}_v - \mathbf{h}_{\mathcal{N}_v}. \tag{4}$$

5 CONCRETE APPROACH: TAIL-GNN

In this section, we develop a concrete approach called Tail-GNN, hinged on the novel concept of transferable neighborhood translation just discussed. We first present the overall framework of Tail-GNN in Fig. 3. As shown in Fig. 3(b), we distill the head nodes to learn the transferable neighborhood translation, in order to predict the missing information on tail nodes (Sect. 5.1). Next, in Fig. 3(c), neighborhood aggregation is conducted on both the head and tail nodes to realize a message-passing GNN (Sect. 5.2). Finally, in Fig. 3(d), the node representations are utilized to optimize the task loss, incorporating several auxiliary constraints to improve the optimization (Sect. 5.3).

5.1 Realizing neighborhood translation

Our neighborhood translation boils down to two key strategies: the contrastive strategy and localizing strategy.

Contrastive strategy. On one hand, as head nodes are structurally rich, we assume their observed neighborhoods are representative enough to be regarded as the ideal neighborhoods, as discussed in Sect. 4. Thus, we exploit and learn the neighborhood translation from head nodes, and then transfer the shared translation to tail nodes in order to predict the missing neighborhood information to complement the observed neighborhood during aggregation.

Specifically, we employ the translation concept in Eq. (2) in each layer of Tail-GNN. Thus, the embedding vector of a *head* node v in

the l-th layer should satisfy

$$\mathbf{h}_v^l + \mathbf{r}_v^l = \mathbf{h}_{\mathcal{N}_v^*}^l, \tag{5}$$

where $\mathbf{r}_v^l$ is the translation vector on node v in the l-th layer, which can be constructed by a shared model. Moreover, $\mathbf{h}_{\mathcal{N}_v^*}^l$ is the representation of the ideal neighborhood in the l-th layer. In particular, given that v is a head node, $\mathbf{h}_{\mathcal{N}_v^*}^l$ can be approximated by $\mathbf{h}_{\mathcal{N}_v}^l$, the representation of the observed neighborhood $\mathcal{N}_v$ in the same layer. Thus, in practice, Eq. (5) can be reformulated as Eq. (6), which will be implemented as an auxiliary loss term to be further discussed in Sect. 5.3.

$$\forall v \in \mathcal{V}_{\text{head}}: \quad \mathbf{h}_v^l + \mathbf{r}_v^l - \mathbf{h}_{\mathcal{N}_v}^l \to \mathbf{0}, \tag{6}$$

where $\mathbf{h}_{\mathcal{N}_v}^l$ can be represented by a mean-pooling over all nodes in the observed neighborhood, *i.e.*, $\mathbf{h}_{\mathcal{N}_v}^l = \textsc{Mean}(\{\mathbf{h}_i^l : i \in \mathcal{N}_v\})$.

On the other hand, we further utilize tail nodes as a contrast to the head nodes, to more precisely inform the model on how to employ the neighborhood translation on tail nodes to predict missing information. However, there exists no one-one correspondence between head and tail nodes for a direct and more meaningful contrast. To this end, to enhance the contrastive power, we supplement the tail nodes with *forged tail nodes*, which can be generated by randomly dropping some links from the head nodes to mimic the actual tail nodes. They contrast with their corresponding head nodes to enhance the prediction and usage of missing neighborhood information. Specifically, for a head node $v \in \mathcal{V}_{\text{head}}$ with neighborhood $\mathcal{N}_v$, we generate a forged tail node v' with neighborhood $\mathcal{N}_{v'}$ such that $\mathcal{N}_{v'} \subset \mathcal{N}_v$ and $|\mathcal{N}_{v'}| \leqslant K$ (the degree threshold), which can be achieved by randomly dropping some of the links of v. For example, in Fig. 3(b), there is a forged tail nodes v_0' generated by link dropout from the head nodes v_0.

Toward robust tail node embedding, the key idea is to uncover the missing neighborhood information. We apply the transferable translation concept in Eqs. (3) and (4) in each layer of the GNN. Specifically, the missing neighborhood information of a *tail* node v (forged or real) in the l-th layer can be predicted as

$$\mathbf{m}_v^l = \mathbf{h}_{\mathcal{N}_v^*}^l - \mathbf{h}_{\mathcal{N}_v}^l = \mathbf{h}_v^l + \mathbf{r}_v^l - \mathbf{h}_{\mathcal{N}_v}^l. \tag{7}$$

The predicted missing information on tail nodes will be further employed in the neighborhood aggregation, in order to complement their observed neighborhoods which may be biased.

Localizing strategy. A straightforward approach to construct the translation vector $\mathbf{r}_v$ on each node v is to employ a globally shared vector $\mathbf{r}$ (per layer), which can be made learnable. While such a shared vector can capture the general pattern across the graph, there still exist significant variations across a large graph containing diverse contexts [16]. For instance, a citation graph contains papers in different fields or topics, and a social network encompasses users from different backgrounds. Thus, it is important to reflect the local context of each node.

More specifically, we aim to incorporate the local context of each node into the translation vector, and simultaneously preserve the generality across the graph. As shown in Fig. 3(b), starting from a globally shared and learnable translation vector $\mathbf{r}$, we further personalize it into a locality-aware translation vector $\mathbf{r}_v$ for each node v w.r.t. its local context.

Formally, in the l-th layer of Tail-GNN, given the globally shared $\mathbf{r}^l \in \mathbb{R}^{d_l}$ in that layer, we devise a personalization function ϕ to transform $\mathbf{r}^l$ into a localized translation vector $\mathbf{r}_v^l \in \mathbb{R}^{d_l}$ for node v. That is,

$$\mathbf{r}_v^l = \phi(\mathbf{h}_v^l, \mathbf{h}_{\mathcal{N}_v}^l, \mathbf{r}^l; \theta_\phi^l), \tag{8}$$

where $\mathbf{h}_v^l$ and $\mathbf{h}_{\mathcal{N}_v}^l$ collectively describe the local context of v, and θ_ϕ^l contains the parameters of ϕ in the l-th layer.

To materialize the personalization function, we consider scaling and shifting transformations [23]. For node v in the l-th layer, its localized vector is given by

$$\mathbf{r}_v^l = \phi(\mathbf{h}_v^l, \mathbf{h}_{\mathcal{N}_v}^l, \mathbf{r}^l; \theta_\phi^l) = (\gamma_v^l + \mathbf{1}) \odot \mathbf{r}^l + \beta_v^l, \tag{9}$$

where $\gamma_v^l \in \mathbb{R}^{d_l}$ is a scaling vector and $\beta_v^l \in \mathbb{R}^{d_l}$ is a shifting vector that are functions of the local context $\{\mathbf{h}_v^l, \mathbf{h}_{\mathcal{N}_v}^l\}$, given by Eqs. (10) and (11). In particular, they have the same dimension as the shared translation vector $\mathbf{r}^l$, which means scaling ($\odot$) and shifting (+) can be done in an element-wise manner. Moreover, $\mathbf{1}$ is a vector of ones to ensure that the scaling factors are centered around one.

$$\gamma_v^l = \textsc{LeakyReLU}(\mathbf{W}_\gamma^{l,1}\mathbf{h}_v^l + \mathbf{W}_\gamma^{l,2}\mathbf{h}_{\mathcal{N}_v}^l) \tag{10}$$

$$\beta_v^l = \textsc{LeakyReLU}(\mathbf{W}_\beta^{l,1}\mathbf{h}_v^l + \mathbf{W}_\beta^{l,2}\mathbf{h}_{\mathcal{N}_v}^l) \tag{11}$$

Here $\textsc{LeakyReLU}(\cdot)$ is used as the activation function. Moreover, each $\mathbf{W}_*^{l,*} \in \mathbb{R}^{d_l \times d_l}$ is a learnable weight matrix. Thus, the personalization function ϕ is essentially parameterized by these weight matrices, *i.e.*, $\theta_\phi^l = \{\mathbf{W}_\gamma^{l,1}, \mathbf{W}_\gamma^{l,2}, \mathbf{W}_\beta^{l,1}, \mathbf{W}_\beta^{l,2}\}$.

5.2 Realizing neighborhood aggregation

We discuss the key operation of neighborhood aggregation in Tail-GNN, on head and tail nodes, respectively.

As we do not assume any missing neighborhood information on the head nodes, their neighborhood aggregation follows the standard GNNs in Eq. (1). That is, we directly aggregate over the observed neighborhood, which is taken as a good approximation of the ideal neighborhood.

In contrast, the observed neighborhoods of tail nodes tend to be biased and not representative enough for meaningful aggregation. As discussed in Sect. 5.1, we can predict the missing neighborhood information on tail nodes to complement their observed neighborhood, in order to approximate the ideal neighborhood. In particular, for a tail node v, in the l-th layer, we first predict its missing neighborhood information $\mathbf{m}_v^l$ by Eq. (7). Next, we perform neighborhood aggregation for the $(l+1)$-th layer by considering both the observed neighborhood $\mathcal{N}_v$ and the missing information $\mathbf{m}_v^l$. That is,

$$\mathbf{h}_v^{l+1} = \mathcal{M}(\mathbf{h}_v^l, \{\mathbf{m}_v^l\} \cup \{\mathbf{h}_i^l : i \in \mathcal{N}_v\}; \theta^{l+1}). \tag{12}$$

5.3 Overall loss

While the node representations can be fed into the task loss for end-to-end learning, to enhance model training we further incorporate several auxiliary constraints into the overall loss.

Task loss. The node representations from the output layer can be employed in an end-to-end manner to minimize the loss for a specific task, such as node classification and link prediction. Using node classification as an example, given a set of training nodes $\mathcal{V}_{\text{tr}}$

(which also contains the corresponding forged tail nodes of the head nodes in the training set), the task loss $\mathcal{L}_t$ can be formulated by cross entropy:

$$\mathcal{L}_t = \sum_{v \in \mathcal{V}_{\text{tr}}} \textsc{CrossEnt}(\mathbf{h}_v^\ell, \mathbf{y}_v) + \lambda_t \|\Theta\|_2^2, \tag{13}$$

where the output layer $\mathbf{h}_v^\ell$ (given ℓ layers in total) has the same dimension as the number of classes and uses a softmax activation, $\mathbf{y}_v$ is a one-hot vector that encodes the class of node v, $\textsc{CrossEnt}(\cdot)$ is the cross entropy function, λ_t is a hyperparameter to adjust the parameter regularization, and Θ encompasses all learnable parameters of Tail-GNN. Specifically, $\Theta = \{\Theta^l : 1 \leqslant l \leqslant \ell\}$ where $\Theta^l = \{\theta^l, \mathbf{r}^{l-1}, \theta_\phi^{l-1}\}$ denotes the parameters set in the l-th layer.

Loss for missing information constraint. As we assume that the head nodes do not have missing information, we need to ensure the missing information, if predicted on a head node, is close to zero, as specified in Eq. (6). Thus, we formulate the following loss to constrain the missing information:

$$\mathcal{L}_m = \sum_{v \in \mathcal{V}_{\text{tr}}} I_v \sum_{l=1}^{\ell} \|\mathbf{m}_v^{l-1}\|_2^2, \tag{14}$$

where $I_v = 1$ if v is a head node and $I_v = 0$ if otherwise.

Loss for adversarial constraint. To make the representations more robust, we further employ a discriminator D [7], to tell whether a node is head or tail based on their output representation. The discriminator, by trying to differentiate head and tail node representations, would force Tail-GNN to generate more effective missing information for the tail nodes and thus improve their representations in order to fool the discriminator. Here we regard the output layer of Tail-GNN as the generator, which contests with the discriminator in the learning process. In particular, we use the following loss for the discriminator:

$$\mathcal{L}_d = \sum_{v \in \mathcal{V}_{\text{tr}}} \textsc{CrossEnt}(I_v, D(\mathbf{h}_v^\ell; \theta_d)) + \lambda_d \|\theta_d\|_2^2,$$

where $D(\cdot; \theta_d)$ is the discriminator's scoring function parameterized by θ_d, which calculates the probability of a node being a head node, as follows.

$$D(\mathbf{h}_v^\ell; \theta_d) = \sigma(\mathbf{w}_d^\top \textsc{LeakyReLU}(\mathbf{W}_d \mathbf{h}_v^\ell + \mathbf{b}_d)), \tag{15}$$

where σ is the sigmoid function, $\mathbf{W}_d \in \mathbb{R}^{d_\ell \times d_\ell}$ is a parameter matrix, $\mathbf{b}_d$ and $\mathbf{w}_d \in \mathbb{R}^{d_\ell}$ are parameter vectors. Thus, $\theta_d = \{\mathbf{W}_d, \mathbf{b}_d, \mathbf{w}_d\}$ contains the learnable parameters of the discriminator D. λ_d is a hyperparameter to adjust the regularizer on θ_d.

Overall loss. Finally, we integrate the task loss and the auxiliary constraints to form the overall loss, which is optimized by

$$\min_\Theta \max_{\theta_d} \ \mathcal{L}_t + \mu \mathcal{L}_m - \eta \mathcal{L}_d, \tag{16}$$

where μ and η are hyperparameters to control the importance of the missing information constraint and the adversarial constraint, respectively. Note that the overall loss involves a minimax game between the discriminator and the output layer of Tail-GNN (*i.e.*, the generator). We present the pseudocode of the training process in Supplement A.

6 EXPERIMENTS

In this section, we perform an extensive empirical evaluation of Tail-GNN on several public benchmark datasets.

Table 1: Summary of datasets.

	# Nodes	# Edges	# Features	# Classes	# Tail ($K = 5$)
Email	1,005	25,571	128	42	235
Squirrel	5,201	217,073	2,089	5	942
Actor	7,600	33,391	931	5	4,823
CoauthorCS	18,333	327,576	6,805	15	8,037
Amazon	937,349	12,455,925	100	44	248,125

6.1 Experimental Setup

Datasets. We employ five public benchmark datasets for evaluation, as summarized in Table 1. (1) *Email* [37] is an e-mail network between members of an European research institution, where each node is a member, and the edges denote the e-mail communications between members. (2) *Squirrel* [22] is a Wikipedia network on the topic of squirrel, in which each node is a page, and the edges denote the citations between pages. (3) *Actor* [22] is an actor co-occurrence network, in which each node is an actor, and each edge links two actors co-occurring in the same Wikipedia page. (4) *CoauthorCS* [25] is a co-authorship graph, in which each node is an author, and each edge links two co-authors. (5) *Amazon* [5] is a co-purchasing network, in which each node is an item, and each edge links two items that have been purchased in the same transaction. Further details and additional processing of the datasets are in Supplement B.

Base GNN models. Our Tail-GNN is agnostic of the base GNN model, and can flexibly work with any neighborhood aggregation-based architecture (see Sect. 3.2). By default, we employ GCN [12] as the base GNN model in our experiments. To further evaluate the flexibility of Tail-GNN, we also use two other popular GNNs as the base models, *i.e.*, GAT [31] and GraphSAGE [8]. Their descriptions and parameter settings are included in Supplement C.

Baselines. To comprehensively evaluate Tail-GNN, we compare it to four categories of baselines, as follows. The details of each baseline are further described in Supplement D.

(1) *Conventional graph representation learning models*: DeepWalk [24] and GCN [12]. They treat all nodes uniformly and do not distinguish head and tail nodes at all.

(2) *Tail node refinement models*: Additive [13], a la carte [11], and meta-tail2vec [17]. They first learn node representations by a base embedding model (any graph embedding or GNN). Next, they perform an embedding refinement step for the tail nodes, by incorporating additional information from neighboring head nodes.

(3) *Robust graph representation learning models*: SDNE [32], ARGA [20] and DDGCN [3]. They aim to enhance the overall robustness of representation learning especially on sparse graphs, but do not specifically target at improving tail nodes.

(4) *Degree-aware models*: DEMO-Net [34] and role2vec [1]. DEMO-Net treats nodes of different degrees differently based on degree-specific transformations, whereas role2vec is able to differentiate the role of nodes based on structural features including degree. Neither explicitly aims at learning robust tail node embeddings.

Settings and parameters. Following the same setup in meta-tail2vec [17], we set the default degree threshold to $K = 5$, *i.e.*,

Table 2: Evaluation on tail node classification using GCN as the base model.

Henceforth, tabular results are in percent; the best result is **bolded** and the runner-up is underlined; a dash (-) denotes no result reported for failing to work on a large dataset.

Methods	Email		Squirrel		Actor		CoauthorCS		Amazon	
	Accuracy	Micro-F	Accuracy	Micro-F	Accuracy	Micro-F	Accuracy	Micro-F	Accuracy	Micro-F
DeepWalk	54.4 ± 0.3	51.3 ± 0.3	28.8 ± 1.6	28.0 ± 2.3	21.8 ± 0.6	18.2 ± 0.9	84.1 ± 0.7	81.5 ± 0.7	83.7 ± 0.1	74.3 ± 0.6
GCN	57.9 ± 1.2	57.7 ± 1.3	24.8 ± 1.3	23.2 ± 1.8	29.7 ± 0.2	15.0 ± 0.9	88.4 ± 0.1	86.1 ± 0.1	82.3 ± 0.2	70.6 ± 0.1
Additive	55.4 ± 0.4	52.5 ± 0.2	27.0 ± 1.7	22.9 ± 1.6	28.1 ± 0.3	15.1 ± 1.3	89.5 ± 0.1	87.8 ± 0.1	84.2 ± 0.2	73.2 ± 0.6
a la carte	21.1 ± 0.4	17.9 ± 0.5	22.5 ± 1.1	22.5 ± 0.7	28.0 ± 0.5	14.8 ± 1.4	88.7 ± 0.2	86.7 ± 0.3	81.1 ± 0.1	69.7 ± 0.7
meta-tail2vec	57.1 ± 0.1	55.3 ± 0.2	25.1 ± 0.5	21.5 ± 0.3	29.7 ± 0.4	20.1 ± 0.7	89.3 ± 0.1	87.4 ± 0.1	81.9 ± 0.1	71.4 ± 0.4
SDNE	32.9 ± 0.6	29.8 ± 0.5	23.8 ± 3.2	16.6 ± 6.2	24.4 ± 0.8	12.6 ± 5.6	70.6 ± 0.9	64.5 ± 1.1	-	-
ARGA	45.1 ± 0.9	41.2 ± 1.0	22.4 ± 1.0	22.8 ± 1.9	25.9 ± 0.3	8.2 ± 0.6	74.6 ± 1.8	67.9 ± 2.5	-	-
DDGCN	39.8 ± 0.6	38.9 ± 0.7	26.3 ± 2.1	26.4 ± 3.3	24.0 ± 0.4	11.7 ± 0.7	73.6 ± 0.9	68.8 ± 1.0	-	-
DEMO-Net	56.9 ± 0.6	56.5 ± 0.7	28.3 ± 0.5	22.5 ± 2.2	28.4 ± 0.8	22.0 ± 1.3	90.8 ± 0.5	88.9 ± 0.6	83.1 ± 0.1	72.0 ± 0.4
role2vec	44.9 ± 1.6	43.8 ± 2.4	26.3 ± 0.8	27.5 ± 1.7	23.1 ± 0.1	18.3 ± 0.6	62.7 ± 0.3	56.3 ± 0.3	77.1 ± 0.2	61.5 ± 0.5
Tail-GCN	**59.2** ± 0.8	**58.5** ± 1.3	**30.2** ± 1.1	**31.1** ± 1.1	**34.9** ± 0.5	**25.2** ± 0.6	**93.6** ± 0.1	**92.7** ± 0.1	**87.0** ± 0.1	**78.2** ± 0.2

Table 3: Evaluation on tail node classification using other GNNs as the base model.

Methods	Email		Squirrel		Actor		CoauthorCS		Amazon	
	Accuracy	Micro-F	Accuracy	Micro-F	Accuracy	Micro-F	Accuracy	Micro-F	Accuracy	Micro-F
GAT	57.9 ± 0.4	57.3 ± 0.2	24.1 ± 2.4	23.1 ± 2.6	29.8 ± 0.6	13.2 ± 2.7	88.6 ± 0.2	86.2 ± 0.2	-	-
Tail-GAT	**59.4** ± 0.9	**58.2** ± 1.2	**28.8** ± 2.1	**30.4** ± 2.6	**34.5** ± 1.3	**24.7** ± 2.0	**92.5** ± 0.1	**90.8** ± 0.1	-	-
GraphSAGE	52.0 ± 1.6	51.3 ± 1.7	27.1 ± 2.7	26.4 ± 4.9	33.1 ± 1.1	23.2 ± 2.4	89.8 ± 2.4	87.7 ± 1.1	79.1 ± 0.4	62.8 ± 0.6
Tail-GraphSAGE	**55.7** ± 0.6	**54.9** ± 0.7	**28.5** ± 1.6	**28.2** ± 2.4	**34.1** ± 1.7	**26.8** ± 1.8	**93.8** ± 0.7	**92.4** ± 1.4	**85.1** ± 0.2	**75.5** ± 0.3

nodes with degree no greater than 5 are regarded as tail nodes. Nevertheless, in our model analysis (Sect. 6.4), we vary the threshold to confirm if Tail-GNN is still superior under different values of K. All experiments are repeated five times, and we report the average results with standard deviations. For detailed hyperparameters used in the baselines and Tail-GNN, please refer to Supplement E.

6.2 Node Classification for Tail Nodes

We conduct tail node classification on the five benchmark datasets. In particular, we use all the head nodes as training, and split the tail nodes into validation and test with a ratio of 1:2. Note that, in model analysis, we will conduct a further experiment to evaluate the head nodes, to ensure that the representations of head nodes remain competitive. Moreover, we apply a multi-class cross entropy loss in Eq. (13) for node classification, and employ accuracy and micro-F score as the evaluation metrics. Note that, to compute micro-F score in the multi-class setting, we ignore the largest class.

For a fair comparison, we employ GCN as the base model for all GNN-based approaches including our Tail-GNN, the refinement models and other GNN-based models. Note that, as refinement models work in a two-stage manner, we utilize the representations learned from GCN as their input for refinement. Moreover, to evaluate the flexibility of Tail-GNN on diverse GNN architectures, we further adopt GAT and GraphSAGE as the base model, and compare them with Tail-GAT and Tail-GraphSAGE, respectively.

Using GCN as base model. We report the results of tail node classification in Table 2. Overall, Tail-GCN significantly outperforms not only its base model GCN, but also all other baselines owning to the mechanism of transferable neighborhood translation. In the following, we discuss the underlying reasons.

Firstly, DEMO-Net and role2vec can distinguish nodes of different degrees, which potentially benefit the group of tail nodes by treating them differently from the head nodes. In particular, DEMO-Net often outperforms the base model GCN, demonstrating that it helps to discriminate nodes of varying structural features. However, since they are not purposely designed for enhancing the tail nodes, which is a particularly challenging group, their performance still falls short. Secondly, SDNE, ARGA and DDGCN improve the robustness of graph learning in general, but like DEMO-Net and role2vec they do not specifically target the tail nodes. In contrast, Tail-GNN is able to leverage the head nodes to help the tail nodes. Furthermore, they cannot cope with the large-scale Amazon dataset. Specifically, ARGA becomes unresponsive without any output in over 24 hours, while SDNE and DDGCN run out of memory. Thirdly, the various tail node refinement models work in two stages, in which their initial embedding stage cannot benefit from the refinement stage in an end-to-end manner. Thus, they are unable to fundamentally address the tail-node problem, and produce inferior results in comparison to Tail-GNN. Nevertheless, Additive and meta-tail2vec often outperform the base model GCN, due to the additional refinement step on the tail nodes.

Using other GNNs as base models. We further utilize GAT and GraphSAGE as the base GNN model and yield Tail-GAT and Tail-GraphSAGE, respectively. Their results are reported in Table 3. We observe that in each case, Tail-GNN outperforms its corresponding base GNN model, showing the flexibility of Tail-GNN to effectively

Table 4: Evaluation on tail link prediction.

Methods	Squirrel		Actor		CoauthorCS	
	MAP	NDCG	MAP	NDCG	MAP	NDCG
DeepWalk	29.5±0.9	45.8±0.8	30.0±1.6	46.1±1.3	32.5±1.5	48.2±1.2
GCN	38.5±0.9	53.2±0.4	33.2±1.2	48.9±0.9	77.7±1.2	83.9±0.8
Additive	36.4±0.7	51.5±0.5	32.1±0.9	48.1±0.8	77.8±0.2	83.9±0.2
meta-tail2vec	39.7±0.6	54.2±0.7	33.1±1.0	48.8±0.8	76.0±0.5	82.7±0.2
Tail-GCN	**41.8**±2.4	**55.9**±1.3	**35.8**±2.2	**51.1**±2.0	**81.0**±0.8	**86.1**±0.4

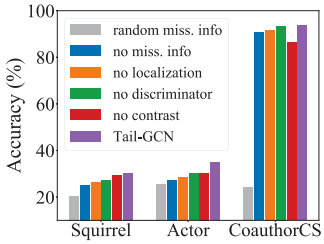


Figure 4: Ablation study.

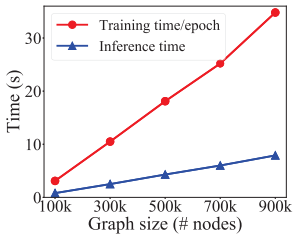


Figure 5: Scalability study.

work with various base models. Note that GAT runs out of memory on the large-scale Amazon dataset.

6.3 Link Prediction for Tail Nodes

We employ a "hide-and-rediscover" approach for link prediction. From the original graph, we randomly remove 5% links, and the new graph will be used for learning node representations for all methods, while the removed links form our ground truth. The ground truth is divided in such a way that all links in the training set contain at least one head node whereas all links in validation/test sets contain at least one tail node. Subsequently, we adopt a ranking-based methodology for link prediction [17]. Given a link (v, a) in the ground truth, we form a triple (v, a, b) in which node b is randomly sampled from the graph and serves as a negative node. We adopt a ranking loss in which we prefer the score between v and a to be higher than the score between v and b. We employ the widely used ranking metrics NDCG and MAP for evaluation. A detailed formulation of the task is included in Supplement F.

We choose several representative baselines for tail link prediction, using GCN as the base model. Their results on three datasets are reported in Table 4. Comparing with tail node classification, we make similar observations. In particular, Tail-GCN consistently outperforms these baselines.

6.4 Model Analysis

We further analyze various aspects of our model Tail-GNN, based on the task of node classification using GCN as the base model.

Ablation study. To evaluate the contribution from each component of Tail-GNN, we conduct an ablation study by comparing with several ablated variants: (1) *random miss. info*: we randomly sample a vector from a standard normal distribution, then feed it to a multilayer perceptron (MLP) to generate the missing information; (2) *no miss. info*: we remove the usage of missing information for neighborhood aggregation on tail nodes; (3) *no localization*: we remove the localization of the globally shared neighborhood translation; (4) *no discriminator*: we remove the adversarial constraint from Tail-GCN; (5) *no contrast*: we remove the forged tail nodes and only use the head nodes for learning the neighborhood translation. Since there are only head nodes, we also remove the discriminator.

We show the results of the ablation study in Fig. 4, and make the following observations. Firstly, random or no missing information impairs the performance of Tail-GNN significantly, showing the importance of missing information for improving the tail node embeddings. Secondly, without localization on the globally shared neighborhood translation, Tail-GNN cannot account for the diverse local contexts of the nodes, which also hurts the performance. Thirdly, the discriminator also contributes to the performance. That means, by discriminating the head and tail nodes, the generation of missing information can be further enhanced. Lastly, without the contrastive strategy, the performance also becomes worse, since there is no one-one correspondence between head and tail nodes for a direct and more meaningful differentiation.

Table 5: Impact of degree threshold K.

Methods	$K = 3$				$K = 7$			
	Squirrel		Actor		Squirrel		Actor	
	Acc.	Mi.F	Acc.	Mi.F	Acc.	Mi.F	Acc.	Mi.F
DeepWalk	28.7	28.9	22.6	19.3	27.9	27.1	22.0	18.2
GCN	24.5	22.8	29.8	15.7	24.4	21.5	30.3	16.2
Additive	28.3	25.8	29.0	16.7	24.8	22.9	28.4	17.4
meta-tail2vec	26.9	21.2	30.4	20.5	24.7	20.5	29.1	20.0
Tail-GCN	**31.5**	**31.7**	**34.9**	**25.5**	**29.9**	**30.9**	**32.9**	**23.6**

Table 6: Evaluation on head node classification.

Methods	Squirrel		Actor		CoauthorCS	
	Acc.	Micro-F	Acc.	Micro-F	Acc.	Micro-F
GCN	28.8±1.0	24.4±0.2	**29.7**±0.8	13.0±1.3	92.0±0.1	90.6±0.1
Tail-GCN	**30.1**±1.3	**24.7**±1.9	28.7±2.3	**24.3**±3.3	**94.2**±0.2	**93.1**±0.4

Scalability study. On the largest dataset Amazon, we sample five subgraphs with between 100k and 900k nodes. In Fig. 5, we report the training (per epoch) and inference time for Tail-GCN on the five subgraphs. We observe that both the training and inference time increase linearly w.r.t. the graph size in terms of the number of nodes. The linear growth demonstrates that the proposed Tail-GNN can scale to very large graphs in real-world scenarios.

Degree threshold K. Although we follow meta-tail2vec [17] to set the degree threshold as $K = 5$ to define tail nodes, in Table 5 we also investigate the impact of the threshold on model performance. Specifically, we also adopt $K = 3$ and $K = 7$ on two datasets, and evaluate the performance of Tail-GNN and several representative baselines. We observe that Tail-GCN still consistently outperforms these baselines under different K values, showing that our proposed approach is generally applicable by transferring the neighborhood translation from head to tail nodes, regardless of small changes in the degree threshold. Note that the data splits (*i.e.*, train/validation/test sets) are different under different K values, and thus their results are *incomparable*.

Head node classification. While our main goal is to improve tail node embedding, we should not sacrifice the performance of head nodes too much relative to the base GNN model. Thus, we further compare Tail-GCN to the base model GCN, and show the results of head node classification in Table 6. Specifically, for each dataset we split the head nodes into two parts, 80% for training and the rest for testing. Note that, due to the difference in training and testing split, the performance reported here is therefore *incomparable* to the results shown in Table 2.

We observe that, on head nodes, our Tail-GCN can achieve competitive and even slightly better performance than GCN. We hypothesize that, although our goal is focused on tail node embedding, the head nodes can still indirectly benefit from the improved tail node embeddings during neighborhood aggregation, as well as the additional constraints in the overall loss.

7 CONCLUSION

In this paper, we investigate the problem of tail node embedding in graph neural networks. We first introduce a new concept of transferable neighborhood translation to capture the relational tie between a node and its neighboring nodes. Subsequently, we propose a novel model Tail-GNN to narrow the gap between head and tail nodes for robust tail node embedding. Specifically, based on a transferable form of neighborhood translation that can be further localized to suit the local context of each node, we predict missing neighborhood information for the tail nodes to complement their neighborhood aggregation. Extensive experiments on five public benchmark datasets show that Tail-GNN can obtain state-of-the-art performance against a comprehensive suit of baselines. For future work, we plan to exploit high-order neighborhood information for tail node embedding.

ACKNOWLEDGMENTS

This research is supported by the Agency for Science, Technology and Research (A*STAR) under its AME Programmatic Funds (Grant No. A20H6b0151).

REFERENCES

[1] Nesreen Ahmed, Ryan Anthony Rossi, John Lee, Theodore Willke, Rong Zhou, Xiangnan Kong, and Hoda Eldardiry. 2020. Role-based graph embeddings. *IEEE TKDE* (2020).
[2] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. 2013. Translating embeddings for modeling multi-relational data. In *NeurIPS*. 2787–2795.
[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* 36, 16 (2020), 4458–4465.
[4] Zhihong Chen, Rong Xiao, Chenliang Li, Gangfeng Ye, Haochuan Sun, and Hongbo Deng. 2020. ESAM: Discriminative Domain Adaptation with Non-Displayed Items to Improve Long-Tail Performance. In *SIGIR*. 579–588.
[5] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. 2019. Cluster-GCN: An efficient algorithm for training deep and large graph convolutional networks. In *KDD*. 257–266.
[6] Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. In *ICML*. 1126–1135.
[7] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *NeurIPS*. 2672–2680.
[8] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *NeurIPS*. 1024–1034.
[9] Aurélie Herbelot and Marco Baroni. 2017. High-risk learning: acquiring new word vectors from tiny data. In *EMNLP*. 304–309.
[10] Bingyi Kang, Saining Xie, Marcus Rohrbach, Zhicheng Yan, Albert Gordo, Jiashi Feng, and Yannis Kalantidis. 2020. Decoupling representation and classifier for long-tailed recognition. In *ICLR*.
[11] Mikhail Khodak, Nikunj Saunshi, Yingyu Liang, Tengyu Ma, Brandon M Stewart, and Sanjeev Arora. 2018. A La Carte Embedding: Cheap but Effective Induction of Semantic Feature Vectors. In *ACL*. 12–22.
[12] Thomas N Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *ICLR*.
[13] Angeliki Lazaridou, Marco Marelli, and Marco Baroni. 2017. Multimodal word meaning induction from minimal exposure to natural text. *Cognitive science* 41 (2017), 677–705.
[14] Jialun Liu, Yifan Sun, Chuchu Han, Zhaopeng Dou, and Wenhui Li. 2020. Deep Representation Learning on Long-tailed Data: A Learnable Embedding Augmentation Perspective. In *CVPR*. 2970–2979.
[15] Siyi Liu and Yujia Zheng. 2020. Long-tail session-based recommendation. In *RecSys*. 509–514.
[16] Zemin Liu, Yuan Fang, Chenghao Liu, and Steven C.H. Hoi. 2021. Node-wise localization of graph neural networks. In *IJCAI*.
[17] Zemin Liu, Wentao Zhang, Yuan Fang, Xinming Zhang, and Steven CH Hoi. 2020. Towards locality-aware meta-learning of tail node embeddings on networks. In *CIKM*. 975–984.
[18] Yuanfu Lu, Yuan Fang, and Chuan Shi. 2020. Meta-learning on Heterogeneous Information Networks for Cold-start Recommendation. In *KDD*. 1563–1573.
[19] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *NeurIPS*. 3111–3119.
[20] 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.
[21] Dongmin Park, Hwanjun Song, Minseok Kim, and Jae-Gil Lee. 2020. TRAP: Two-level Regularized Autoencoder-based Embedding for Power-law Distributed Data. In *TheWebConf*. 1615–1624.
[22] Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. 2020. Geom-GCN: Geometric graph convolutional networks. In *ICLR*.
[23] Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. 2018. FiLM: Visual reasoning with a general conditioning layer. In *AAAI*.
[24] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. DeepWalk: Online learning of social representations. In *KDD*. 701–710.
[25] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. 2018. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868* (2018).
[26] Min Shi, Yufei Tang, Xingquan Zhu, David Wilson, and Jianxun Liu. 2020. Multi-Class Imbalanced Graph Convolutional Network Learning. In *IJCAI*. 2879–2885.
[27] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. LINE: Large-scale information network embedding. In *WWW*. 1067–1077.
[28] Kaihua Tang, Jianqiang Huang, and Hanwang Zhang. 2020. Long-Tailed Classification by Keeping the Good and Removing the Bad Momentum Causal Effect. *NeurIPS* (2020).
[29] Xianfeng Tang, Huaxiu Yao, Yiwei Sun, Yiqi Wang, Jiliang Tang, Charu Aggarwal, Prasenjit Mitra, and Suhang Wang. 2020. Investigating and Mitigating Degree-Related Biases in Graph Convolutional Networks. In *CIKM*. 1435–1444.
[30] Cunchao Tu, Zhengyan Zhang, Zhiyuan Liu, and Maosong Sun. 2017. TransNet: Translation-Based Network Representation Learning for Social Relation Extraction.. In *IJCAI*. 2864–2870.
[31] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2018. Graph attention networks. In *ICLR*.
[32] Daixin Wang, Peng Cui, and Wenwu Zhu. 2016. Structural deep network embedding. In *KDD*. 1225–1234.
[33] Felix Wu, Amauri H Souza Jr, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Q Weinberger. 2019. Simplifying Graph Convolutional Networks. In *ICML*. 6861–6871.
[34] Jun Wu, Jingrui He, and Jiejun Xu. 2019. DEMO-Net: Degree-specific graph neural networks for node and graph classification. In *KDD*. 406–415.
[35] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. 2020. A comprehensive survey on graph neural networks. *IEEE TNNLS* 32, 1 (2020), 4–24.
[36] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How powerful are graph neural networks?. In *ICLR*.
[37] Hao Yin, Austin R Benson, Jure Leskovec, and David F Gleich. 2017. Local higher-order graph clustering. In *KDD*. 555–564.
[38] Jianwen Yin, Chenghao Liu, Weiqing Wang, Jianling Sun, and Steven CH Hoi. 2020. Learning Transferrable Parameters for Long-tailed Sequential User Behavior Modeling. In *KDD*. 359–367.
[39] Dawei Zhou, Jingrui He, Hongxia Yang, and Wei Fan. 2018. SPARC: Self-paced network representation for few-shot rare category characterization. In *KDD*. 2807–2816.

REPRODUCIBILITY SUPPLEMENT

A Pseudocode

We outline the pseudocode of model training in Alg. 1.

Algorithm 1 Model Training for Tail-GNN

Input: graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X})$, task-related training data $\mathcal{V}_{\text{tr}}$.
Output: Model parameters Θ.

1: initialize parameters Θ, θ_d;
2: **while** not converged **do**
3: sample a batch of nodes from $\mathcal{V}_{\text{tr}}$;
4: **for** each node v in the batch **do** ▷ construct forged tail nodes
5: **if** v is a head node **then**
6: $v' \leftarrow \text{LinkDropout}(v)$;
7: add v' to the batch;
8: **for** each node v in the batch **do**
9: **for** each layer $l \in \{0, ..., \ell - 1\}$ **do**
10: $\mathbf{r}_v^l = \phi(\mathbf{h}_v^l, \mathbf{h}_{\mathcal{N}_v}^l, \mathbf{r}^l; \theta_\phi^l)$; ▷ localization, Eq. (9)
11: $\mathbf{m}_v^l \leftarrow \mathbf{h}_v^l + \mathbf{r}_v^l - \mathbf{h}_{\mathcal{N}_v}^l$; ▷ missing information, Eq. (7)
12: **if** v is a head node **then** ▷ aggregation for head, Eq. (1)
13: $\mathbf{h}_v^{l+1} \leftarrow \mathcal{M}(\mathbf{h}_v^l, \{\mathbf{h}_i^l : i \in \mathcal{N}_v\}; \theta^{l+1})$;
14: **else** ▷ aggregation for tail, Eq. (12)
15: $\mathbf{h}_v^{l+1} \leftarrow \mathcal{M}(\mathbf{h}_v^l, \{\mathbf{m}_v^l\} \cup \{\mathbf{h}_i^l : i \in \mathcal{N}_v\}; \theta^{l+1})$;
16: update Θ by minimizing Eq. (16) with θ_d fixed;
17: update θ_d by maximizing Eq. (16) with Θ fixed;
18: **return** Θ.

In line 1, we initialize all the parameters. In line 3, we sample a batch of nodes from the training data. In lines 4–7, we construct forged tail nodes from the head nodes by link dropout, for training with our contrastive strategy. In lines 8–15, we perform neighborhood aggregation for all the nodes. In particular, for each node, we first personalize the neighborhood translation w.r.t. local context in line 10; then missing information is calculated in line 11; afterwards, neighborhood aggregation is conducted differently for head and tail nodes in lines 12–15. In lines 16–17, we train the model with the adversarial constraint, alternating between the generator and the discriminator. Finally, we return the trained model Θ.

Based on the pseudocode, compared to GNNs, the overhead of Tail-GNN includes link dropout on head nodes, the localization of neighborhood translation, and the calculation and usage of missing information. Taking GCN as an example base model. In GCN, the neighborhood aggregation for one node in the l-th layer has complexity $O(d_l d_{l-1} \bar{d})$, where d_l is the dimension of the l-th layer and $\bar{d}$ is the average node degree. In contrast, in Tail-GNN, for one node in the l-th layer: (1) link dropout incurs $O(K)$ time since we only need to sample up to K links on a head node to forge a tail node, where K is the degree threshold of the tail node; (2) localizing the neighborhood translation requires traversing and mapping all the neighbors, thus with complexity $O({d_{l-1}}^2 \bar{d})$ based on the calculation of $\mathbf{h}_{\mathcal{N}_v}^{l-1}$, and Eqs. (9), (10) and (11); (3) the calculation of missing information takes $O(d_{l-1})$ time based on Eq. (7); (4) neighborhood aggregation with missing information takes $O(d_l d_{l-1}(\bar{d}+1))$ complexity based on Eq. (12) where the missing information can be regarded as an "extra" neighbor. Thus, the complexity of Tail-GNN for one node in a single layer is $O\big(K + {d_{l-1}}^2 \bar{d} + d_{l-1} + d_l d_{l-1}(\bar{d}+1)\big)$. As K, d_l, d_{l-1} are small constants, it belongs to the same complexity class as GCN, both running in linear time in $\bar{d}$ for one node in a single layer. Overall, on graphs with the same average degree, Tail-GNN incurs linear time w.r.t. the number of nodes.

B Details of Datasets

We give further details and describe any additional processing on the five datasets to supplement Sect. 6.1.

(1) *Email* [37] is an e-mail network between members of an European research institution, where each node is a member, and the edges denote the e-mail communications between them. Each member belongs to one of the 42 departments in the institution, taken as the 42 node classes. Following previous work [17], we employ the 128-dimensional vectors generated by DeepWalk to serve as the node features.

(2) *Squirrel* [22] is a Wikipedia network on the topic of squirrel, in which each node is a page, and the edges denote the citations between pages. Node features correspond to informative nouns in the articles. The articles are classified into five categories by the number of average monthly traffic of viewership.

(3) *Actor* [22] is an actor co-occurrence network, in which each node is an actor, and each edge links two actors co-occurring in the same Wikipedia page. Node features are bag-of-word vectors, and the actors are classified into five categories.

(4) *CoauthorCS* [25] is a co-authorship graph, in which each node is an author, and each edge links two co-authors. Node features are paper keywords from the author's papers, and the class indicates the most active field of each author out of 15 fields.

(5) *Amazon* [5] (originally called *Amazon2M*) is a co-purchasing network, in which each node is an item, and each edge links two items that have been purchased in the same transaction. Node features are generated by extracting bag-of-word features from the item descriptions followed by a Principal Component Analysis. The top-level item categories are used as the class labels. In our experiments, we use a connected subgraph with 937,349 nodes sampled from the original graph, with a total of 44 classes.

C Details of Base GNN Models and Settings

We give additional details and model setup of the various base GNN models used in our experiments.

Descriptions. We describe the three base GNN models below.

- GCN [12]: GCN is characterized by a graph convolution operator to aggregate information from neighboring nodes recursively. The convolution operator is roughly equivalent to a mean pooling over the neighbors for each node.
- GAT [31]: GAT assigns different weights to neighbors by a self-attention mechanism during aggregation, thus accounting for the importance of each neighbor.
- GraphSAGE [8]: GraphSAGE is similar to GCN, but places more emphasis on the target node embedding during aggregation.

Model setup. Based on the recommendations from their original papers, we further tune the parameters for each base GNN model to attain optimal performance. In particular, for all base GNN models, we adopt a two-layer architecture and utilize an Exponential Linear Unit as the activation function for the hidden layer. In the output layer, we set the dimension as the number of classes and employ a

softmax activation for node classification, and set the dimension as 16 for link prediction. Additionally, for GCN, we set the hidden layer dimension as 32; for GAT, we set the hidden layer dimension as 8, use 3 attention heads in each layer, and apply a dropout rate of 0.5; for GraphSAGE, we adopt a mean-pooling over the neighbors and concatenate the pooled embedding with the target node embedding during aggregation, and set the hidden layer dimension as 16.

D Details of Baselines

In this section, we describe each baseline in more details.

(1) *Conventional graph representation learning models.*
- DeepWalk [24], which first samples a large number of paths by random walk, then feeds them to a skip-gram model [19] to capture node co-occurrences.
- GCN [12], the same as the base model in Supplement C.

(2) *Tail node refinement models.*
- Additive [13], which aggregates the embeddings of neighboring nodes as the embedding of a tail node.
- a la carte [11], an extension of Additive, which further employs a transformation through an auxiliary regression task after the aggregation.
- meta-tail2vec [17], which formulates the tail node embedding problem as a few-shot regression task based on the embeddings of 2-hop neigbhbors, under a meta-learning paradigm [6].

(3) *Robust graph representation learning models.*
- SDNE [32]: a deep graph embedding model to capture both the global and local structural information of the graph, which is especially designed to improve the overall robustness on sparse networks.
- ARGA [20]: an adversarially regularized graph autoencoder to enhance the overall robustness of graph representation learning.
- DDGCN [3]: a graph neural network which utilizes a dual dropout strategy at node- and edge-levels to enhance the overall robustness of GNNs on sparse networks. It is optimized w.r.t. a self-supervised objective.

(4) *Degree-aware models.*
- role2vec [1]: a graph embedding approach based on attributed random walks, which utilizes features (*e.g.*, motif and degree) to distinguish the role of different nodes.
- DEMO-Net [34], which employs degree-specific transformations on nodes, so that nodes with different degrees are treated differently.

These baselines, except GCN and DEMO-Net, are not trained end-to-end with downstream tasks, for which we train a logistic regression to perform node classification and an MLP to perform link prediction.

E Hyperparameters Settings

Baselines. To achieve optimal performance, we tune the parameters for each baseline based on their default settings in their papers. For DeepWalk, we set the number of walks per node as 80 with walk length 40, and set window size as 10. For Additive and a la carte, we employ mean-pooling as the aggregation function, which produces better performance than min- and max- pooling. For meta-tail2vec, we set the learning rate of adaptation as 0.01, the number of gradient updates as 5, the size of hidden layer in the regression model as 1024, and the number of hops as 2. For both a la carte and meta-tail2vec, we use the same regression model with Euclidean norm. For SDNE, we set the weight of first-order proximity as 100, and the weight of reconstruction as 10. For ARGA, we set the dimension of its hidden layer as 32. For DDGCN, we set the dropout probability as 0.5, and set the dual-dropout coefficient as 1.0. For role2vec, we utilize node degrees to distinguish roles, and set the number of walks per node as 10 with walk length 80, and window size as 5. For DEMO-Net, we set the hidden dimension of the first layer as 32, and the hash dimension as 256. Its output layer follows the same setting as the base GNN model.

Our method. As we adapt Tail-GNN to different base GNN models (*i.e.*, GCN, GAT and GraphSAGE), we obtain three corresponding models, namely, Tail-GCN, Tail-GAT, and Tail-GraphSAGE. In particular, for each of them, we employ the same setup as their corresponding base GNN model. Additionally, we set the other parameters of Tail-GNN. For node classification, we set the weights of the auxiliary loss in Eq. (16) as $\mu = 0.001$ and $\eta = 0.1$; for link prediction, we adopt $\mu = 0.001$ and $\eta = 1.0$, For both tasks, we set the coefficient of regularizations including λ_t and λ_d as 0.0001.

F Formulation of Tail Link Prediction

We elaborate on the task of tail link prediction, as supplemental information for the description in Sect. 6.3.

In general, we employ a "hide-and-rediscover" principle. Specifically, from the original graph, we randomly remove 5% links from all links incident to nodes with degree between 2 and 10, so that the removed links are more concentrated on the tail nodes. The new graph will be used for learning node representations for all methods, while the removed links form our ground truth. In particular, we employ the ground-truth links with two head nodes as training instances, and those with two tail nodes as validation/test instances. Moreover, we equally divide the ground-truth links with one head node and one tail node into two subsets, one for training and the one for validation/test. As such, all links in training contain at least one head node, and all links in validation/test contain at least one tail node. The links in validation/test is further split into validation and test sets by a ratio of 1:2. Note that, there is no overlap between ground-truth links in training, validation and test sets.

For training, given a ground-truth link (v, a) with v as a head node, we form a triple (v, a, b) in which b is randomly sampled from the graph and serves as a negative node. We adopt a ranking loss in which we prefer the score between v and a to be higher than the score between v and b. To calculate the score between two nodes, we adopt the dot product of their representation vectors. For testing, given a ground-truth link (v, a) such that v is a tail node, we further randomly sample another nine nodes as negative nodes, to form a candidate list including a and the nine negative nodes that are randomly shuffled. For evaluation, we rank the ten nodes based on their scores with node v, and ideally a should be ranked higher in the list of ten. We employ the widely used ranking metrics NDCG and MAP for evaluation.