Graph Mixed Supervised Learning via Lipschitz Bounded Mixture-of-Experts Similarity Network

Anonymous Author(s)

ABSTRACT

The success of GNNs highly relies on the accurate labeling of data. Existing methods of ensuring accurate labels, such as weakly-supervised learning, mainly focus on the existing nodes in the graphs. However, in realistic graphs, new nodes always continuously emerge, with different categories and even label noises. In light of this, we introduce a new problem, *Graph Mixed Supervised Learning*, that describes the need to model new nodes with novel classes and potential label noises. To solve this problem, we propose **L**ipshitz Bounded M**I**xture-of-Experts **S**imilari**T**y N**E**twork Tra**N**sfer (LISTEN), a novel model to encode new nodes and handle label noises. Specifically, we first design a node similarity network to capture the knowledge from the original classes, aiming to obtain insights for the emerging novel classes. Then, to enhance the similarity network's generalization to new nodes that could be out-of-distribution, we employ the Mixture-of-Experts to increase the width of the graph neural network encoder and similarity network. To further avoid losing generalization ability during training, we introduce the Lipschitz bound to stabilize model output, when distribution shift happens. Empirical experiments validate LISTEN's effectiveness: we observe a substantial enhancement of up to 11.34% in node classification accuracy compared to the backbone model when subjected to the challenges of label noise on novel classes across five benchmark datasets.

KEYWORDS

Graph Representation Learning, Mixture-of-Experts, Lipschitz Constant, Label Noise

ACM Reference Format:

Anonymous Author(s). 2023. Graph Mixed Supervised Learning via Lipschitz Bounded Mixture-of-Experts Similarity Network. In . ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Graph Neural Networks (GNNs) employ message-passing mechanisms to iteratively update node representations, allowing them to aggregate information from neighboring nodes. This capability makes GNNs particularly potent for modeling and learning from graph-structured data. Consequently, GNNs have consistently demonstrated state-of-the-art performance across various graph-related tasks, such as node classification [8, 13, 25] and graph classification [31, 32]. Furthermore, GNNs have proven to be notably effective in real-world web applications, including social network analysis [19, 36], fraud detection [37], natural language processing [5, 33], and recommendation systems [4, 11, 34].

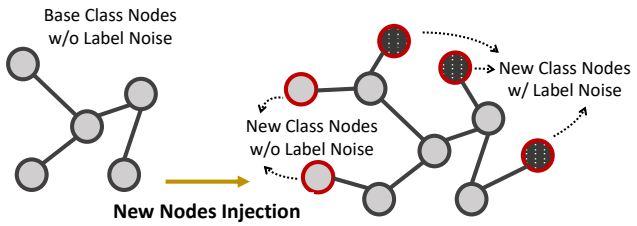

Figure 1: Graph Mixed Supervised Learning: in the original graph, base class nodes with accurate labels; then new novel class nodes with label noise are attached to the original graph.

However, the successes of GNNs rely heavily on the precise annotation of training data. If nodes are labeled with noise, the model can be fooled, leading to inaccurate representations with nodes belonging to different categories mixed together. Considering there are always new nodes with noisy or sparse labels added to the graphs, encoding nodes on real-world web graphs such as social, citation, and sales networks can be challenging. Therefore, many graph weakly supervised learning methods are proposed to handle node label noise [3, 20, 21]. For example, NRGNN [3] and RTGNN [22] are proposed to predict accurate pseudo-labels and distinguish noisy labels from clean labels to provide supervision for the new nodes with label noise. Nevertheless, these methods are limited to handling new nodes that have pre-existing classes in the training set. When there is a need to model the nodes with novel classes and potential label noises, these methods can not utilize the knowledge contained in the original graph and fail to perform well.

Modeling and encoding new nodes is an important and common situation in the real world. While predicting new nodes that have existing classes can be straightforward, it is challenging to learn and predict the new nodes with classes that are distinct from existing node classes in the original graphs. In light of this, we introduce a new problem, i.e. **Graph Mixed Supervised Learning**, which corresponds to learning from existing nodes with correct labels to deduce the true category of new nodes, while noises can present for these new nodes. To solve this problem, we propose a novel framework called LISTEN (**L**ipshitz Bounded M**I**xture-of-Experts **S**imilari**T**y N**E**twork Tra**N**sfer). Specifically, we first obtain the position features and concatenate them to node features. Then the concatenated features of the original graph are used to train a similarity network, which encodes the base class node without label noise and generates similarity scores of any two node embedding pairs. After

, ,

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

that, the similarity network inferences on new class nodes with potential label noises. For each node, the similarity scores between it and the other nodes with the same label are averaged as its loss weight during subsequent model training for node classification. To generalize the similarity transfer from training node pairs to inference node pairs, we utilize the Mixture-of-Experts technique to increase the width of the similarity network and introduce the Lipshitz constant as loss regularization with theoretical analysis. The major contributions of this paper are summarized as follows:

- To the best of our knowledge, this work first proposes a realistic graph learning scenario called "Graph Mixed supervised Learning". According to our empirical studies in experimental sections, most current works are ineffective in this setting because they do not fully utilize existing clean labels of the base class nodes.
- The proposed LISTEN model utilizes a similarity network to extract implicit knowledge for weighing the training loss of new class nodes with label noises. To further generalize this process, the Mixture-of-Experts module is adopted, and the corresponding solution of the Lipshitz bound is proposed with a theoretical guarantee.
- Extensive experiments on several graph datasets demonstrate the effectiveness of our method. Results show that LISTEN outperforms other popular baselines and handles the graph mixed supervised learning scenario well against label noises.

2 RELATED WORK

Graph Neural Networks (GNNs). GNNs have gained widespread attention due to their ability to effectively learn non-Euclidean data and achieve outstanding performance in various graph mining tasks [1, 8, 30]. Early works related to GNN proposed such as graph convolutional networks (GCN) are proposed to apply convolutional concepts to graph data [6, 13, 28]. Graph attention networks were introduced to improve GCNs by incorporating attention mechanisms to weight the importance of neighboring nodes during message passing [25, 27]. In addition, graph recurrent neural networks have been proposed to address the limitations of GNNs in handling long-distance message passing on large graphs by incorporating gating mechanisms inspired by recurrent neural networks [23]. To overcome the problem of oversmoothing, deeper GNNs were constructed using skip connections to learn more comprehensive representations [15–17]. While prior works have primarily focused on improving standard accuracy, our method addresses the growing concern of label noise of nodes with novel classes.

Graph Learning with Label Noise. Previous research [20, 35]has shown that deep graph models are vulnerable to label noises, which urges the need to design robust graph learning methods against label noise. Among current methods that handle node label noise, D-GNN [20] employs backward loss correction [21] to improve performance. NRGNN [3] learns robust node representations with noisy and sparse labels by connecting unlabeled nodes with labeled nodes and further predicting accurate pseudo-labels to provide supervision. Different from NRGNN which explicitly governs noisy labels, RTGNN [22] distinguishes noisy labels from clean labels and provides label correction to reduce the impact of label noises. However, when there is the need to model new nodes with novel classes and potential label noises, these methods do not utilize knowledge contained by strongly labeled base class nodes in the original graph to handle label noise with novel classes.

Mixed-supervised Learning. Mixed-supervised learning refers to learning novel categories with cheap weak labels, with knowledge from a set of base categories that are already accurately labeled. Computer vision researchers have explored similar settings (also named cross-supervised learning or weak-shot learning) in different tasks, such as object detection [9, 18, 38], fine-grained classification [2], semantic segmentation [38], and instance segmentation [10, 14]. To the best of our knowledge, we are the first to attempt to define mixed-supervised learning on graphs.

3 PRELIMINARY

Problem Definition. Consider an undirected attributed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{X}, \mathcal{Y})$, where $\mathcal{V}$ is the set of N nodes, $\mathcal{E}$ is the set of edges, $\mathcal{X} \in \mathbb{R}^{N \times D}$ denotes the set of node features with D dimensions, and $\mathcal{Y}$ denotes node labels with $\mathcal{K}$ classes. The label for these nodes with **base classes** is **clean**. After new nodes with **novel classes** attached to original graph, the graph becomes to $\mathcal{G}' = (\mathcal{V}', \mathcal{E}', \mathcal{X}', \mathcal{Y}')$. Note $\mathcal{Y}'$ has $\mathcal{K}'$ classes with $|\mathcal{K}' - \mathcal{K}|$ classes annotated with **noise**, and the true label is $\mathcal{Y}^t$. The objective is to learn a model f_θ to maximum prediction accuracy in terms of the true label. Formally, the Graph Mixed Supervised Learning objective can be formulated as:

$$\min_{\theta} \mathcal{L}(f_\theta(\mathcal{G}'), \mathcal{Y}^t_{k \in \mathcal{K}'}). \tag{1}$$

Sparse Mixture of Experts. The Mixture-of-Experts (MoE) is a machine learning design that ensembles multiple expert models to make predictions or perform other tasks. The basic idea behind MoE is to divide the input space into numerous partitions, and assign different experts to different partitions. Each expert is a specialized model that is trained to perform well on a specific subset of the input space. The final output is obtained by assembling the predictions of all the experts, typically using a gating mechanism that determines the weight of each expert based on the input. Formally, let h be the input space to the MoE, and $\mathbf{E} = \{E_i(\cdot)\}_{i=1}^{N}$ denotes that an MoE layer consists of n experts. The output of the MoE is given by:

$$y = \sum_{i=1}^{N} p_i(h) E_i(h), \tag{2}$$

where E_i is the i-th expert model, $p_i(h)$ is the weight assigned to the i-th expert for the input space h, and N is the number of experts. The weights are typically obtained from a gating network, which is trained to assign higher weights to experts that are more likely to make accurate predictions for a given input space.

A popular approach to design the gating mechanism is to use a softmax function to calculate the weights as the gating mechanism, and to use neural networks as the expert models by $p_i(h) = \frac{\exp(t(h)_i)}{\sum_{j=1}^{N} \exp(t(h)_j)}$, where $t(h)$ is a linear transformation to compute the logits of the experts given the input space h, $t(h)_i$ is the i-th value of the obtained logits, which is the weight for the i-th expert in the current layer.

Lipschitz Continuous. A function $f : \mathbb{R}^n \to \mathbb{R}^m$ is established to be Lipschitz continuous on an input set $\mathcal{X} \subseteq \mathbb{R}^n$ if there exists a constant $K \geq 0$ such that for all $x, y \in \mathcal{X}$, f satisfies the following

inequality:

$$||f(x) - f(y)|| \leq K||x - y||, \forall x, y \in \mathcal{X}. \tag{3}$$

The smallest possible K in Equation (3) is the Lipschitz constant of f, denoted as $\text{Lip}(f)$:

$$\text{Lip}(f) = \sup_{x,y \in \mathcal{X}, x \neq y} \frac{||f(x) - f(y)||}{||x - y||}, \tag{4}$$

and we say that f is a K-Lipschitz function. Previous works such as [26] propose to use Lipschitz bound as spectral norm regularization to make the model more stable.

4 METHODS

This section presents a novel model called LISTEN to increase the GNN performance when novel category data has label noise. Figure 2 illustrates the overall design of the model. LISTEN is consisted with two training processes. First, a Mixture-of-Experts Similarity Network (**SimMoE**) is introduced and trained on the original graph to predict whether two nodes are similar or not in terms of their labels. SimMoE consists of two key modules: Mixture-of-Experts GNN encoder (**GNNMoE**) and Mixture-of-Experts MLP similarity predictor (**MLPMoE**). The pipeline of SimMoE can be divided into three steps: concatenating structure embedding, encoding node features by GNNMoE, and predicting similarity scores by MLPMoE. In addition, we incorporate the Mixture-of-Experts to GNNMoE and MLPMoE to improve the model generalization. The Lipshitz bound for MLPMoE is also proposed to regularize model training to avoid overfitting. Second, we utilize the trained SimMoE to infer the graph injected by new nodes and calculate accumulated similarity scores between nodes that have the same label. Since outlier nodes with noise labels are dissimilar to the nodes with right labels, they have lower accumulated similarity scores. Finally, the similarity scores are taken as the weight of node losses to enforce model discard knowledge from nodes with wrong labels during the training process.

4.1 Training the Mixture-of-Experts Similarity Network on the Original Graph

SimMoE that contains GNNMoE and MLPMoE is used to predict whether two nodes are similar or not. The illustration of SimMoE is shown in part (a) of Figure 2. Specifically, input node features of the original graph $\mathcal{G}$ are $\boldsymbol{X} \in \mathbb{R}^{N \times D}$, where N is the number of nodes and D is the feature dimension. $\boldsymbol{X}$ are concatenated to the structure features to obtain final node features $\boldsymbol{F} \in \mathbb{R}^{N \times D'}$, where D' is the final node feature dimension. Second, the GNNMoE encodes features to embeddings $\boldsymbol{Z} \in \mathbb{R}^{N \times H}$, where H is the hidden dimension. After that, we pairwise the node embeddings to obtain node pair similarity features $\boldsymbol{S} \in \mathbb{R}^{N \times 2H}$. Finally, the node pair similarity features are used to train the MLPMoE with binary similarity labels, i.e., "similar" (two nodes have the same label) and "dissimilar" (two nodes have different labels).

Concatenating Structure Embedding. To comprehensively learn the structural information of the subgraph, we propose to extend node features by obtaining structural embedding from unsupervised structural learning methods such as Node2Vec [7]. Specifically, the Node2Vec model learns the structural information from the adjacency matrix $\boldsymbol{A}$, and the learned embedding $\boldsymbol{P}$ of nodes is concatenated to original node features $\boldsymbol{X}$ to obtain final node features $\boldsymbol{F}$. This process can be formulated as:

$$\boldsymbol{P} = \text{Node2Vec}(\boldsymbol{A}), \boldsymbol{F} = \text{CONCAT}(\boldsymbol{P}, \boldsymbol{X}). \tag{5}$$

Encoding Node Features by GNNMoE. When we employ the SimMoE to infer the new graph with injected new nodes, there exists a distribution shift since the model is trained on the original graph with base classes. To improve the generalization of SimMoE, we incorporate the Mixture-of-Experts techniques in GNNMoE to learn diverse representations. Specifically, we divide FC-layer $\text{Linear}(\cdot)$ into multiple expert networks to process the target representation h. The process is formulated as follows:

$$\text{LinMoE}(h) = \sum_{i \in \mathcal{T}} p_i(h) \cdot \text{Linear}_i(h), \tag{6}$$

where $\mathcal{T}$ represents the set of activated top-k expert indices. $\text{LinMoE}(\cdot)$ combines the output of multiple expert networks. In particular, the top-k activated expert indices are determined by the gate-value $p_i(h_v)$, which can be obtained using a softmax function:

$$p_i(h) = \frac{\exp(t(h)_i + \varepsilon_i)}{\sum_{k=1}^{N} \exp(t(h)_k + \varepsilon_k)}, \tag{7}$$

where $t(\cdot)$ is a linear transformation, and N is the number of all experts. The activated expert indices in the LinMoE module are determined by a gate-value $p_i(h)$. To illustrate, $p_i(h)$ takes the logits of all experts, obtained through a linear transformation of the input feature vector h, and computes the activation probability of each expert. The logits are weighted by the i-th value $t(h)_i$ of the linear transformation, and a random noise term ε_i is added to ensure randomness in the expert activating procedure. ε_i is typically chosen to be a sample from a Gaussian distribution. Each layer of the GNNMoE encoder first transforms the features of the target node and its neighboring nodes using the $\text{LinMoE}^{(l)}(\cdot)$ and then aggregates the transformed features of the neighboring nodes using $\text{AGG}(\cdot)$, which can be formulated as follows:

$$\begin{aligned} h_v^{(l)} =& \text{COMB}^{(l)} \Big(\text{LinMoE}^{(l)}(h_v^{(l-1)}), \\ & \text{AGG} \left(\left\{ \text{LinMoE}^{(l)}(h_u^{(l-1)}), \forall u \in N_v \right\} \right) \Big), \end{aligned} \tag{8}$$

where $\text{AGG}(\cdot)$ and $\text{COMB}(\cdot)$ represent the neighbor aggregation and combination functions, respectively. N_v denotes the set of node v's all neighboring nodes u. $h_v^{(l-1)}$ is l-th layer node representations.

Predicting Similarity Scores by MLPMoE. After obtaining node embeddings $\boldsymbol{Z}$, we pairwise node embeddings to form node similarity features $\boldsymbol{S}$. MLPMoE takes $\boldsymbol{S}$ as the input to output similarity scores for each node pair. Same with GNNMoE, we need to improve the generalization of the similarity predictor when we employ the SimMoE to infer the new graph. We incorporate LinMoE to formulate MLPMoE, which updates the node pair similarity representation h_s in layer l as follows:

$$s^l = \text{LinMoE}(s^{l-1}). \tag{9}$$

Although the MLPMoE improves the generalization, as the training process iterates, the model gradually overfits the training data while losing generalization. Therefore, we introduce the Lipschitz bound to gain better generalization by stabilizing the output of

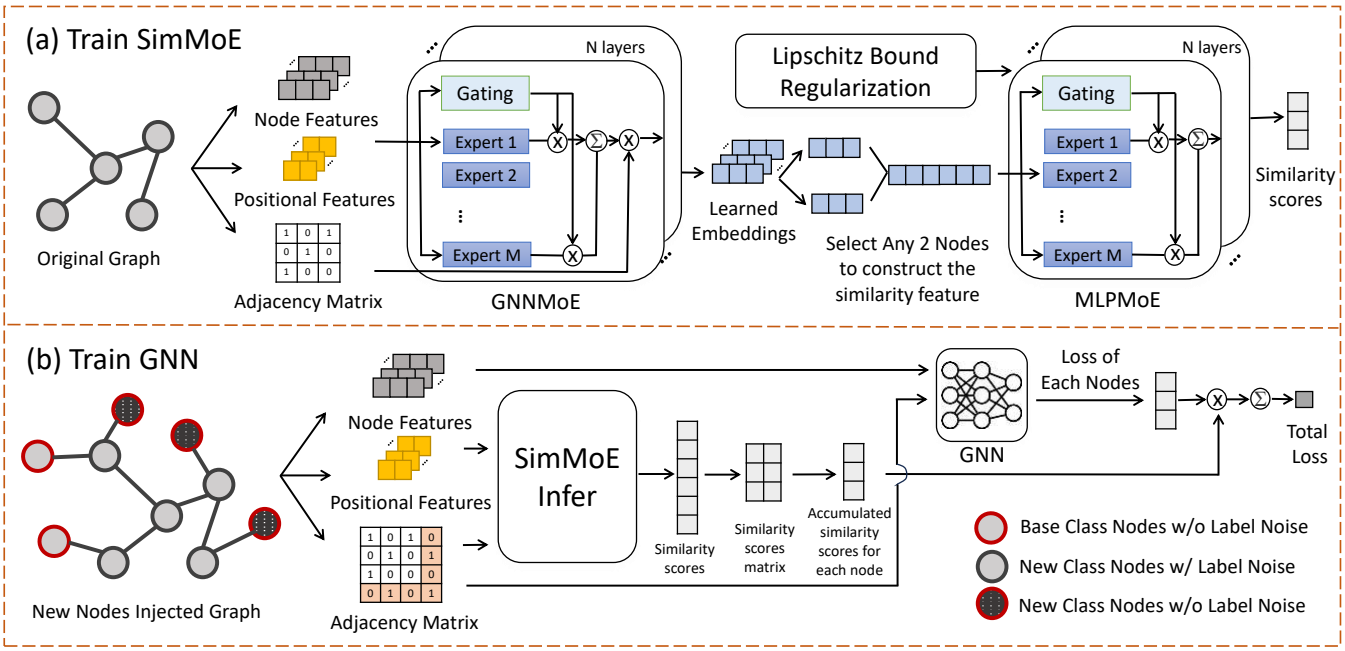

Figure 2: Our framework. (a) First, SimMoE which contains GNNMoE and MLPMoE is trained on the original graph. SimMoE is trained to predict whether two nodes are similar or not in terms of their labels. The pipeline of SimMoE can be divided into three steps: concatenating structure embedding, encoding node features by GNNMoE, and predicting similarity scores by MLPMoE. (b) Second, GNN is trained for classification with the support of weighted loss. We utilize the trained SimMoE to infer the graph injected by new nodes and calculate accumulated similarity scores for each node, where lower scores indicate wrong labels. Finally, the similarity scores are taken as the weight of node losses to enforce model discard knowledge from nodes with wrong labels during the training process.

the MLPMoE when the distribution shift happens during model inference. Here we first analyze MLPMoE and give its theoretical Lipschitz bound, then utilize it to regularize the train loss.

Consider MLPMoE to be represented as $f : \boldsymbol{S} \in \mathbb{R}^{N \times F^{\text{in}}} \to \boldsymbol{Y} \in \mathbb{R}^{N \times F^{\text{out}}}$, where $\boldsymbol{S}$ is the input feature matrix, $\boldsymbol{Y}$ is the output feature matrix, and N is the number of node feature pairs. To analyze the stability of the model output, we examine the Lipschitz bound of the Jacobian matrix of MLPMoE by introducing the following lemma and proposition. The detailed proofs are listed in Appendix A.

Lemma 4.1. *Let g be a Lipschitz continuous function. Denote g_i to be the i-th layer of g, $i = 1, \cdots, m$. Then the Lipschitz constant of function g satisfies*

$$\text{Lip}(g) \leq \left\| [\text{Lip}(g_i)]_{i=1}^{m} \right\|, \tag{10}$$

where $[\text{Lip}(g_i)]_{i=1}^{m}$ denotes the m-dimensional vector whose i-th component is $\text{Lip}(g_i)$.

Based on Lemma 4.1, we present the following proposition:

Proposition 4.2. *Let $\boldsymbol{Y}$ be the output of an L-layer MLPMoE (represented in $f(\cdot)$) with $\boldsymbol{X}$ as the input. Assuming the activation function (represented in $\rho(\cdot)$) is ReLU with a Lipschitz constant of $\text{Lip}(\rho) = 1$, then the global Lipschitz constant of the Mixture-of-Experts network, denoted as $\text{Lip}(f)$, satisfies the following inequality:*

$$\text{Lip}(f) \leqslant \max_j \prod_{l=1}^{L} \left\| F^l \right\| \left\| \left[\sum_{k=1}^{K_l} p_l^k \mathcal{J}^k(h^l) \right]_j \right\|_\infty, \tag{11}$$

where F^l represents the output dimension of the l-th Mixture-of-Experts layer which contains K_l experts; j is the index of the node; k is the index of the expert in the l-th layer; p_l^k is the gate value for k-th expert in the l-th layer and the vector $\left[\mathcal{J}^k(h^l)\right]$ is as follows:

$$\left[\mathcal{J}^k(h^l)\right] = \left[\left\| \boldsymbol{J}_1^k(h^l) \right\|, \left\| \boldsymbol{J}_2^k(h^l) \right\|, \cdots, \left\| \boldsymbol{J}_{F^l}^k(h^l) \right\| \right]. \tag{12}$$

Notably, $\boldsymbol{J}_i^k(h^l)$ denotes the input and output of the i-th row of the Jacobian matrix of the k-th expert in the l-th layer.

Proposition 4.2 provides a method to estimate the Lipschitz constants $\text{Lip}(f)$ of similarity network $f(\cdot)$ by estimating Lipschitz constants of subnetworks. However, it is challenging to consider the potential cumulative effect and calculate Lipshitz constants of complex multiple mixture-of-experts layers in the similarity network. Therefore, we consider MLPMoE as a unitary model and directly derive the Lipschitz bound from the input and output of MLPMoE. To achieve this, we introduce the Jacobian matrix and define the Jacobian matrix of i-th nodes pair

similarity feature as $[J_i]_{F^{\text{out}} \times F^{\text{in}}} = \left[J_{i1}^\top, J_{i2}^\top, \cdots, J_{iF^{\text{out}}}^\top\right]^\top$, where $J_{ij} = \left[\frac{\partial Y_{ij}}{\partial X_{i1}}, \frac{\partial Y_{ij}}{\partial X_{i2}}, \cdots, \frac{\partial Y_{ij}}{\partial X_{iF^{\text{in}}}}\right]^\top$. Then we define $\mathcal{J}$ as follows:

$$\mathcal{J} = \left[\mathcal{J}_1^\top, \mathcal{J}_2^\top, \cdots, \mathcal{J}_N^\top\right]^\top, \tag{13}$$

where $\mathcal{J}_i = [\|J_{i1}\|, \|J_{i2}\|, \cdots, \|J_{iF^{\text{out}}}\|]^\top$. According to the definition of $\mathcal{J}$, we take the l_2-*norm* for each row of $\mathcal{J}$ and then take the infinite norm for the entire $\mathcal{J}$ to obtain the Lipschitz bound of the whole MLPMoE, which is formulated as follows:

$$\text{Lip}(f) = \|\mathcal{J}\|_{\infty,2}\,. \tag{14}$$

During the training, the Lipschitz bound for the model output is computed using the gradients and norms of the input node pair features. We utilize this Lipschitz bound as a regularization term in the loss function, ensuring that the model's output stays within the defined constraints when we generalize the MLPMoE for inference. Therefore, the final loss for training SimMoE can be formulated as:

$$\mathcal{L}^{sim} = \mathcal{L}^s + \text{Lip}(f), \tag{15}$$

where $\mathcal{L}^s$ is the training loss supervised by labels for similarity features (The label of the similarity feature is similar/dissimilar if two nodes have the same/different labels).

4.2 Training GNN on the Injected Graph with Weighted Node Classification Loss

After SimMoE is trained, we employ the SimMoE to infer the graph $\mathcal{G}' = (\mathcal{V}', \mathcal{E}', \mathcal{X}')$ which is injected with new nodes to obtain similarity scores. Note that during inference, the process of forming similarity features is different from training. Specifically, for nodes $\mathcal{V}'$ with c class, we randomly sample n nodes from each class to form the nodes set $\mathcal{V}'_s$, where $|\mathcal{V}'_s| = nc$. Then, we pairwise each node from $\mathcal{V}'$ to each node $\mathcal{V}'_s$ to construct similarity features. Finally, we take the output of similarity scores and construct a $N \times n$ similarity scores matrix $\mathcal{S}$. The element $\mathcal{S}_{i,j}$ represents the similarity score between the node $v' \in \mathcal{V}'$ and node $v'_s \in \mathcal{V}'_s$ that has the same label with v'_i. Finally, we calculate the accumulated similarity score $w_{v'} = \frac{1}{c}\sum_{j=1}^{c} \mathcal{S}_{v',v'_s}$ for the node v' and let $w_{v'}$ be the weight for downstream GNN classification training loss for node v'. Formally, this process can be formulated as follows:

$$\mathcal{L} = \frac{1}{|\mathcal{V}'|} \sum_{v' \in \mathcal{V}'} -w_i y'_{v'} \log[f(x_{v'})], \tag{16}$$

where $y'_{v'}$ is the label for node v'. Since we calculate the accumulated similarity scores between the nodes with the same label, outlier and non-dominant nodes that do not belong to their labeled class have lower similarity scores. Lower weights indicate the wrong annotations, and accordingly these nodes contribute less to learning the GNN classifier. This helps GNN avoid being fooled by noise labels during training.

5 EXPERIMENTS

This section presents comprehensive experiments to evaluate the effectiveness of our method when training on novel category data with noise labels. The experiments aim to address the following key research questions: **RQ-1**: Can our method outperform other state-of-the-art (SOTA) methods in terms of graph weak shot learning?

Table 1: Statistics of datasets used in the experiments.

Name	#nodes	#edges	#features	#classes
Cora	2,708	10,556	1,433	7
CiteSeer	3,327	9,104	3,703	6
Computers	13,752	491,722	767	10
Photo	7,650	238,162	745	8
Actor	7,600	30,019	932	5
Wisconsin	251	515	1,703	5

RQ-2: What is the contribution of each component w.r.t robustness against noise labels? **RQ-3**: How does our method generalize similarity when data is out-of-distribution? **RQ-4**: How similarity scores enhance representations learning on the graph?

5.1 Setup

Datasets. We use six widely used PyG benchmark datasets with different graph types, i.e., Cora, CiteSeer, Computers, Photo, Actor, and Wisconsin, to evaluate our method. For experiments, we randomly split datasets into 80% training sets, 10% validation sets, and 10% test sets. The statistics of datasets used in the experiments are listed in Table 1.

Baselines. We compare our method with four types of baselines. First, we compare the general GNN model GCN [13]. Second, we compare popular graph weakly supervised learning methods, including CP-GNN [35], NRGNN [3], RTGNN [22]. Third, we consider the graph few-shot learning method GPPT [24]. Finally, we adapt the mixed supervised learning method SimTrans [2] for computer vision to graphs. We also adapt Jaccard-GCN [29] to our setting by calculating Jaccard similarity scores to replace end-to-end similarity calculation in LISTEN. We keep the parameters and configurations of baselines the same as the origin paper.

Evaluation. To adapt the datasets to our graph weak shot learning setting, we construct two graphs from the original graph by categories. We split the original category set C to half ($\frac{\lceil |C| \rceil}{2}$) base category set C_b and half ($\frac{\lfloor |C| \rfloor}{2}$) novel category set C_n, where $C_b \cup C_n = C$ and $C_b \cap C_n = \emptyset$. After that, we construct the graph with label noise rate $r \in \{0, 0.1, 0.3\}$ for training. The rate r is formally defined as the probability of a new training node v having another label among novel categories. Finally, we train our method and baselines on the constructed graph with label noises. We evaluate node classification accuracy according to the test set that has true labeled data.

Implementation Details. We report the mean and standard deviation of ten independent runs with the same data splits and random seeds. We use two layers for the GCN encoder and the MLP in similarity network. Four experts is utilized for each layer. In addition, we set the learning rate to 0.01 and the noisy gate rate to 0.01. The epoch number for training similarity network is 100 and for node classification is 500. We use Adam [12] to optimize the model. Both the SimMoE and the GNN classifier are implemented in PyTorch and trained on one NVIDIA 3090 GPU. The hidden sizes for GNNMoE and MLPMoE are 32 and 16 respectively. For classification GCN, the hidden sizes are 16, 16, 64, 64, 32, 32 for Cora,

Table 2: Performance comparison of different methods. N.R. denotes label noise rate on novel classes and w.o. represents no label noise. The best result is bolded and the runner-up is underlined.

Dataset	N.R.	GCN [13]	CP-GNN [35]	NRGNN [3]	RTGNN [22]	GPPT [24]	JacGCN [29]	SimTrans [2]	LISTEN
Cora	w.o.	86.83 ± 0.38	86.72 ± 0.31	87.64 ± 0.40	85.61 ± 0.31	86.72 ± 0.33	85.20 ± 0.38	86.79 ± 0.22	**88.34 ± 0.32**
	0.1	82.47 ± 0.55	83.47 ± 0.72	84.13 ± 0.48	84.94 ± 0.62	83.32 ± 0.36	83.32 ± 0.45	84.24 ± 0.43	**86.94 ± 0.24**
	0.3	77.86 ± 0.35	78.04 ± 0.69	79.23 ± 0.48	82.95 ± 0.57	81.70 ± 0.51	79.00 ± 0.38	81.85 ± 0.40	**88.23 ± 0.44**
CiteSeer	w.o.	73.80 ± 0.32	70.92 ± 0.40	71.70 ± 0.15	72.66 ± 0.65	73.56 ± 0.42	72.75 ± 0.36	72.54 ± 0.28	**75.82 ± 0.33**
	0.1	67.58 ± 0.76	70.68 ± 0.71	71.40 ± 0.26	72.00 ± 0.33	69.59 ± 0.56	67.73 ± 0.61	69.92 ± 0.32	**75.28 ± 0.70**
	0.3	62.98 ± 0.57	64.66 ± 0.39	63.64 ± 0.64	67.91 ± 0.57	66.38 ± 0.61	65.83 ± 0.77	67.43 ± 1.10	**74.32 ± 0.32**
Photo	w.o.	**94.89 ± 0.08**	93.70 ± 0.88	93.91 ± 0.19	92.84 ± 0.19	92.54 ± 0.34	91.33 ± 0.46	93.07 ± 0.19	94.51 ± 0.18
	0.1	91.29 ± 1.36	92.82 ± 0.20	92.67 ± 1.06	90.95 ± 0.46	92.75 ± 0.26	92.25 ± 3.22	92.58 ± 0.77	**93.25 ± 0.24**
	0.3	87.99 ± 2.40	87.87 ± 4.77	92.60 ± 1.27	91.96 ± 0.74	91.03 ± 0.47	90.39 ± 3.53	90.48 ± 0.23	**93.79 ± 0.28**
Computer	w.o.	89.83 ± 3.33	88.95 ± 0.40	87.84 ± 1.08	88.20 ± 1.30	90.33 ± 0.62	88.21 ± 1.04	87.61 ± 0.71	**90.28 ± 1.08**
	0.1	85.91 ± 3.92	88.62 ± 2.14	86.51 ± 0.66	86.04 ± 3.26	88.00 ± 2.94	87.27 ± 2.36	85.91 ± 3.92	**89.45 ± 0.21**
	0.3	80.44 ± 7.31	79.13 ± 5.54	83.64 ± 1.04	83.46 ± 2.81	85.83 ± 6.37	84.92 ± 5.85	83.37 ± 5.00	**89.13 ± 1.22**
Actor	w.o.	26.09 ± 0.73	26.67 ± 0.51	26.87 ± 0.41	26.95 ± 0.63	26.95 ± 0.63	26.97 ± 0.34	27.33 ± 0.44	**29.17 ± 0.51**
	0.1	25.45 ± 0.46	26.37 ± 0.52	26.12 ± 0.31	25.25 ± 0.37	27.13 ± 0.30	27.12 ± 0.68	26.88 ± 0.14	**29.68 ± 0.34**
	0.3	25.22 ± 0.65	25.86 ± 0.26	25.70 ± 0.35	25.96 ± 0.46	26.75 ± 0.47	25.80 ± 0.77	26.71 ± 0.59	**29.68 ± 0.24**
Wisconsin	w.o.	47.20 ± 2.40	43.60 ± 1.96	40.40 ± 1.96	44.80 ± 2.04	44.40 ± 2.65	43.20 ± 2.04	46.40 ± 2.65	**48.40 ± 2.65**
	0.1	34.00 ± 1.26	37.60 ± 1.50	36.00 ± 2.19	38.80 ± 0.98	35.20 ± 1.60	36.40 ± 1.96	37.60 ± 1.50	**44.00 ± 2.83**
	0.3	27.60 ± 1.96	33.60 ± 1.96	27.20 ± 1.60	35.60 ± 3.67	35.60 ± 3.44	26.40 ± 4.08	31.20 ± 2.40	**38.80 ± 2.40**

CiteSeer, Photo, Computer, Actor, Wisconsin datasets. The code can be accessed through this anonymous link[1].

5.2 Performance Comparison

To answer the first research question **RQ-1**, we evaluate the graph mixed supervised performance of LISTEN and compare it with other SOTA baselines. The results are reported in Table 2. According to the table, we can observe that LISTEN comprehensively outperforms other baselines under 3 different label noise rates (w.o. noise, noise rate 0.1, and noise rate 0.2) across six graph datasets.

Specifically, when nodes with novel class have label noise, LISTEN demonstrates impressive robust accuracy under two different label noise rates across all the datasets, while other baseline methods experience a severe decrease in accuracy as the label noise rate increases. For instance, on the Cora and CiteSeer datasets, LISTEN outperforms the most competitive baseline RTGNN by 2.00% and 3.28% when the label noise rate is 0.1, respectively. LISTEN outperforms RTGNN by 5.28% on Cora and 6.41% on CiteSeer when the label noise rate increases to 0.3. On the Amazon dataset, compared to the most competitive baselines, LISTEN outperforms the runner-up baselines by 3.53% when the label noise rate is 0.1 and by 0.43% when the label noise rate increases to 0.3 on the Photo dataset. On the Computer dataset, our method outperforms the runner-up baselines by 0.83% when the label noise rate is 0.1 and by 3.30% when the label noise rate increases to 0.3. On the Actor dataset, LISTEN outperforms the most competitive baseline GPPT by 2.55% when the label noise rate is 0.1 and by 2.93% when the label noise rate increases 0.3. On the Wisconsin dataset, our method outperforms the runner-up baseline RTGNN by 5.20% when the label noise rate is 0.1 and by 3.30% when the label noise rate increases 0.3.

[1]https://tinyurl.com/GraphMixedSupervisedLearning

In addition, our LISTEN also demonstrates strong representation ability on clean graphs without label noise and outperforms other baselines. This observation indicates that our model discriminates many outlier nodes and assigns them lower loss weight during training, which helps the model concentrate on learning representations from nodes with better quality. In summary, the above results showcase the effectiveness of LISTEN against label noise against different noise rates on the graph datasets, facilitated by our Lipschitz bounded Mixture-of-Experts similarity network.

5.3 Ablation Study

LISTEN integrates the structure embedding, the network with Mixture-of-Experts, and the corresponding Lipshitz bound to improve similarity transfer. Thus, to answer the second research question **RQ-2**, we remove each of the components of our method and conduct experiments to observe the performance. The results are shown in Table 3. In particular, we ablate the model as (a) without structure embedding (w.o. Struc.), (b) without Mixture-of-Experts in MLPMoE (w.o. MLPMoE), (c) without Mixture-of-Experts in GNNMoE (w.o. GNNMoE), (d) without the Lipshitz bound (w.o. Lipshitz), and (e) using the vanilla GCN instead of LISTEN. Our experiments show that removing any component of the LISTEN decreases its performance when novel classes contain noises. This highlights the critical role each component plays in similarity transfer and its ability to provide robustness against label noises. For (a) **w.o. Struc.**, removing the structure embedding of the similarity network decreases LISTEN's ability to learn structure similarity knowledge. For example, on the CiteSeer dataset, the model experiences a 0.69% loss in accuracy when the label noise rate is 0.1, and a 4.18% loss in accuracy when the label noise rate increases to 0.3. For (b) **w.o. MLPGNN**, removing the MoE in MLPMoE prevents the model

Table 3: Ablation studies for our method on three graph datasets of varying label noise rate. N.R. denotes label noise rate on novel classes and w.o. represents no label noise. The best result is bolded and the runner-up is underlined.

Dataset	Noise	GCN	w.o. Struc	w.o. MLPMoE	w.o. GNNMoE	w.o. Lipshitz	LISTEN
Cora	w.o.	86.83 ± 0.38	87.79 ± 0.14	87.38 ± 0.25	87.12 ± 0.35	88.34 ± 0.27	**88.34 ± 0.32**
	w. 0.1	82.47 ± 0.55	86.72 ± 0.35	86.53 ± 0.26	86.63 ± 0.13	84.54 ± 0.49	**86.94 ± 0.24**
	w. 0.3	77.86 ± 0.35	87.75 ± 0.34	86.75 ± 0.46	86.82 ± 0.67	79.96 ± 0.79	**88.23 ± 0.44**
CiteSeer	w.o.	73.80 ± 0.32	74.44 ± 0.59	76.33 ± 0.40	**76.93 ± 0.76**	75.64 ± 0.55	75.82 ± 0.33
	w. 0.1	67.58 ± 0.76	74.59 ± 0.53	73.62 ± 0.28	74.22 ± 0.56	74.53 ± 0.42	**75.28 ± 0.70**
	w. 0.3	62.98 ± 0.57	70.14 ± 0.43	73.17 ± 0.36	73.07 ± 0.27	64.72 ± 0.92	**74.32 ± 0.32**
Photo	w.o.	94.89 ± 0.08	94.59 ± 0.13	94.59 ± 0.77	**94.89 ± 0.44**	93.19 ± 0.60	94.51 ± 0.18
	w. 0.1	91.29 ± 1.36	92.96 ± 0.76	92.45 ± 0.50	92.68 ± 0.32	91.47 ± 0.36	**93.25 ± 0.24**
	w. 0.3	87.99 ± 2.40	93.75 ± 2.40	91.65 ± 0.80	91.88 ± 1.07	89.43 ± 0.80	**93.79 ± 0.28**
Computer	w.o.	89.83 ± 3.33	90.83 ± 0.87	**90.83 ± 1.24**	89.45 ± 2.09	89.83 ± 3.33	90.28 ± 1.08
	w. 0.1	85.91 ± 3.92	89.12 ± 1.75	87.77 ± 2.55	88.66 ± 0.68	85.91 ± 3.92	**89.45 ± 0.21**
	w. 0.3	80.44 ± 7.31	88.87 ± 4.01	86.90 ± 4.01	87.65 ± 2.12	80.44 ± 7.31	**89.13 ± 1.22**
Actor	w.o.	26.09 ± 0.73	29.09 ± 0.21	28.76 ± 0.43	28.87 ± 0.35	26.01 ± 0.66	**29.17 ± 0.51**
	w. 0.1	25.45 ± 0.46	29.57 ± 0.25	28.78 ± 0.21	29.07 ± 0.34	27.47 ± 0.49	**29.68 ± 0.34**
	w. 0.3	25.22 ± 0.65	28.00 ± 0.40	28.45 ± 0.38	28.67 ± 0.24	26.96 ± 0.21	**29.68 ± 0.24**
Wisconsin	w.o.	47.20 ± 2.40	46.80 ± 2.40	47.00 ± 2.40	48.40 ± 0.88	46.80 ± 1.05	**48.40 ± 2.65**
	w. 0.1	34.00 ± 1.26	44.00 ± 2.26	35.20 ± 2.26	43.80 ± 1.55	37.20 ± 1.40	**44.00 ± 2.83**
	w. 0.3	27.60 ± 1.96	38.40 ± 2.40	35.00 ± 3.48	38.20 ± 2.06	38.60 ± 2.45	**38.80 ± 2.40**

from learning diverse similarity representations, indicating lower model capacity limits the model's generalization to unseen data distribution. On the Cora dataset, the removal of this component results in a 23.2% loss in accuracy when the label noise rate is 0.1, and a 29.5% loss in accuracy when the noise rate increases to 0.3. For (c) **w.o. GNNMoE**, removing MoE in GNNMoE prevents the model from learning diverse node representations. On the Photo dataset, the removal of this component results in a 0.57% loss in accuracy when the label noise rate is 0.1, and a 1.81% loss in accuracy when the noise rate increases to 0.3. For (d) **w.o. Lipshitz**, removing the Lipshitz bound for the Mixture-of-Experts network decreases the model's performance due to the loss of generalization for similarity. On the Actor dataset, it results in a 2.21% increase in accuracy when the noise rate is 0.1, and a 2.72% loss in accuracy when the noise rate increases to 0.3. For (e) **Vanilla GCN**, removing all components degenerate LISTEN into a vanilla GCN model. This resultes in a 10.37% decrease in accuracy on the Cora dataset, an 11.34% loss in accuracy on the CiteSeer dataset, and an 8.69% loss on the Computer dataset when the label noise rate is 0.3. This ablation study demonstrates the contribution of each component in the proposed LISTEN model and shows the effectiveness of LISTEN in improving the GNNs' performance against noise labels.

5.4 What Wins Better performance against Label Noise?

To address the third research question **RQ-3**, we investigate the main generalizing contribution of our model provided by the Mixture-of-Experts as well as corresponding Lipshitz bounds and analyze how they improve similarity transfer.

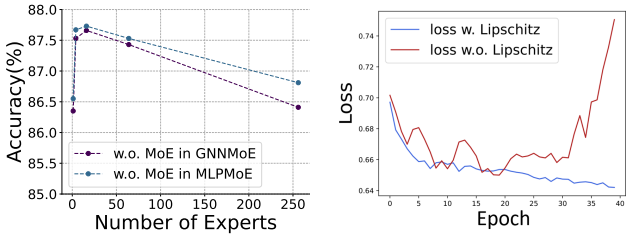

Figure 3: The classification accuracy with different numbers of experts (left) and the inference loss on novel classes when training on base classes on the Cora dataset (right).

Mixture-of-Experts in the similarity Network. To generalize the model to node pair features with distribution shift, we introduce the Misture-of-Experts to the similarity network. SimMoE model contains multiple experts, thus it obtains diverse representations and improves the generalization. As shown in Figure 3, models with more experts bring better performance of LISTEN. Although too many experts decrease the model performance because each expert is dispatched with too little data, i.e., the data-hungry problem, it can be solved by sampling more nodes to form more similar features in SimMoE during training.

Lipshitz bound. To study the effectiveness of Lipshitz bound during transfer similarity inference when a data distribution shift happens, we plot the loss curve in Figure 3. The results show that as the training epoch increases, thanks to the generalization ability learned by the Lipshitz bounded network, the loss of the similarity network equipped with LISTEN continues to decrease. On the other hand, the similarity network without the Lipshitz bound gradually

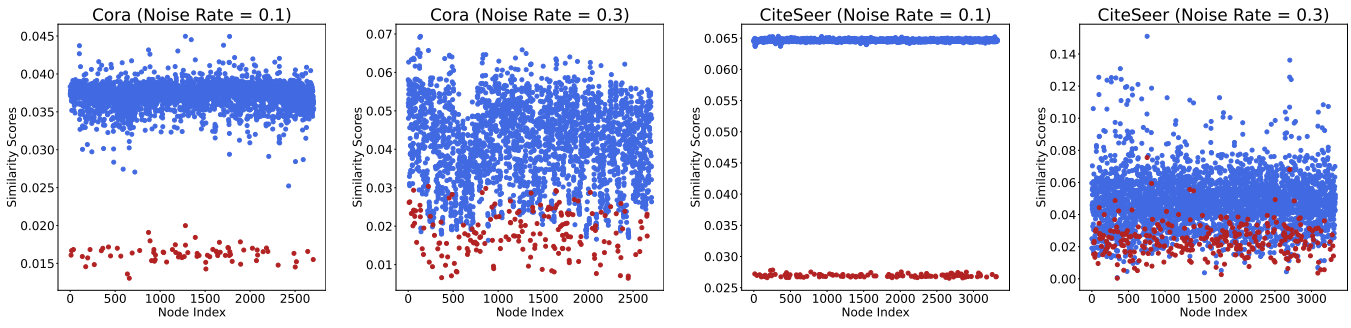

Figure 4: The similarity scores of nodes with indexes. Blue nodes denote nodes without label noise and red nodes denote nodes with label noise. Nodes with label noise will be recognized and separated from nodes without label noise in terms of similarity scores by LISTEN. The higher label noise rate makes it harder to recognize nodes with label noise.

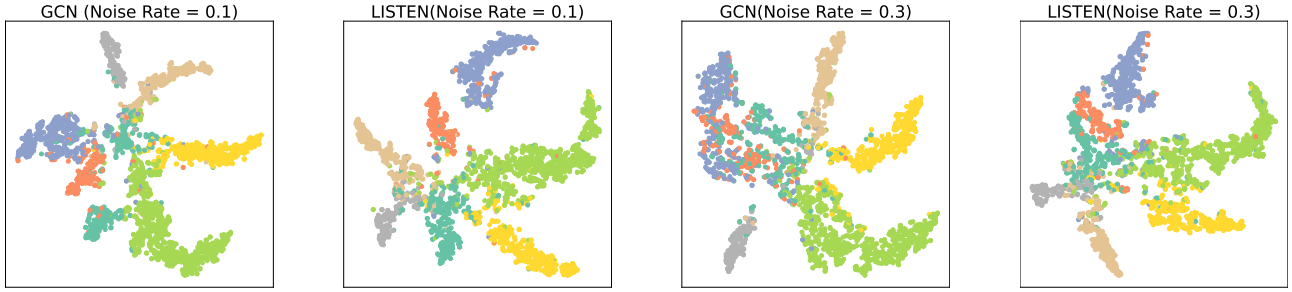

Figure 5: The node representations obtained by vanilla GCN and LISTEN. Vanilla GCN is fooled by label noise and wrongly clusters node representations even if these nodes have different true labels (Blue, orange, and dark green nodes in the figure). LISTEN still performs well on the clustering of these nodes with label noise.

overfits the training data and loses generalization. This demonstrates how the Lipshitz bound generalizes similarity transfer.

5.5 How Similarity Scores Enhance Representation Learning against Noise

To address the fourth research question **RQ-4**, we explore how obtained similarity scores enhance the graph representation learning when the noise is assigned to the labels. As Figure 4 shows, the similarity network outputs the scores for each node. Generally, the similarity scores for nodes without label noise (blue) are higher than the scores for nodes with label noise (red). When the label noise rate is low, i.e. 0.1, nodes with label noise will be easily recognized and separated from nodes without label noise. When the label noise rate increases to 0.3, the scores for nodes with and without label noise are relatively closer, which indicates that more label noise hinders the right calculation of similarity scores. However, we observe that the average similarity scores for nodes with label noise are still lower than that for nodes without label noise.

After obtaining the similarity scores, we take them as the weight of each node's loss to minimize the negative effect of nodes with label noise. As Figure 5 shows, vanilla GCN is fooled by label noise. It wrongly clusters node representations even if these nodes have different true labels. This phenomenon becomes even worse when the label noise rate increases from 0.1 to 0.3. In contrast, LISTEN performs well on the clustering of these nodes with label noises. Moreover, LISTEN also maps some node representations to the cluster (class) that has a different label and no label noises, while having better performance under w.o. noise setting in Table 2. We explain this phenomenon as LISTEN can assign lower loss weight to some outlier or wrongly labeled training nodes and learn better representations.

6 CONCLUSION

In this paper, we first identify a new practical issue within *Graph Mixed Supervised Learning* which describes the need to model new nodes with novel classes and potential label noises. To solve this problem, we propose LISTEN, a novel model to encode new nodes and handle label noises. In particular, LISTEN trains the similarity network to capture the knowledge from the original classes, aiming to obtain insights for the emerging novel classes. We also employ the Mixture-of-Experts techniques and Lipschitz bound to increase the generalization of the similarity network. By utilizing the Lipschitz bounded Mixture-of-Experts similarity network to calculate loss weight for nodes with potential label noise, the GNN classifier avoids the negative effect of new nodes with noise labels. Our experimental results show the effectiveness of LISTEN in encoding new nodes with novel categories and the superiority in handling label noises. Overall, LISTEN is an outstanding solution to nodes' label noises for Graph Mixed Supervised Learning scenarios in real-world web applications.

REFERENCES

[1] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

[2] Junjie Chen, Li Niu, Liu Liu, and Liqing Zhang. Weak-shot fine-grained classification via similarity transfer. In *NeurIPS*, 2021.

[3] Enyan Dai, Charu Aggarwal, and Suhang Wang. Nrgnn: Learning a label noise resistant graph neural network on sparsely and noisily labeled graphs. In *KDD*, 2021.

[4] Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. Graph neural networks for social recommendation. In *WWW*, 2019.

[5] Zichu Fei, Qi Zhang, and Yaqian Zhou. Iterative gnn-based decoder for question generation. In *EMNLP*, 2021.

[6] Hongyang Gao, Zhengyang Wang, and Shuiwang Ji. Large-scale learnable graph convolutional networks. In *KDD*, 2018.

[7] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *KDD*, 2016.

[8] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NeurIPS*, 2017.

[9] Judy Hoffman, Sergio Guadarrama, Eric S Tzeng, Ronghang Hu, Jeff Donahue, Ross Girshick, Trevor Darrell, and Kate Saenko. Lsda: Large scale detection through adaptation. In *NeurIPS*, 2014.

[10] Ronghang Hu, Piotr Dollár, Kaiming He, Trevor Darrell, and Ross Girshick. Learning to segment every thing. In *CVPR*, 2018.

[11] Tinglin Huang, Yuxiao Dong, Ming Ding, Zhen Yang, Wenzheng Feng, Xinyu Wang, and Jie Tang. Mixgcf: An improved training method for graph neural network-based recommender systems. In *KDD*, 2021.

[12] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

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

[14] Weicheng Kuo, Anelia Angelova, Jitendra Malik, and Tsung-Yi Lin. Shapemask: Learning to segment novel objects by refining shape priors. In *ICCV*, 2019.

[15] Guohao Li, Matthias Müller, Ali Thabet, and Bernard Ghanem. Deepgcns: Can gcns go as deep as cnns? In *ICCV*, 2019.

[16] Guohao Li, Matthias Müller, Guocheng Qian, Itzel Carolina Delgadillo Perez, Abdulellah Abualshour, Ali Kassem Thabet, and Bernard Ghanem. Deepgcns: Making gcns go as deep as cnns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[17] Guohao Li, Matthias Müller, Bernard Ghanem, and Vladlen Koltun. Training graph neural networks with 1000 layers. In *ICML*, 2021.

[18] Yan Liu, Zhijie Zhang, Li Niu, Junjie Chen, and Liqing Zhang. Mixed supervised object detection by transferring mask prior and semantic similarity. In *NeurIPS*, 2021.

[19] Seth A Myers, Aneesh Sharma, Pankaj Gupta, and Jimmy Lin. Information network or social network? the structure of the twitter follow graph. In *WWW*, 2014.

[20] Hoang NT, Choong Jun Jin, and Tsuyoshi Murata. Learning graph neural networks with noisy labels. *arXiv preprint arXiv:1905.01591*, 2019.

[21] Giorgio Patrini, Alessandro Rozza, Aditya Krishna Menon, Richard Nock, and Lizhen Qu. Making deep neural networks robust to label noise: A loss correction approach. In *CVPR*, 2017.

[22] Siyi Qian, Haochao Ying, Renjun Hu, Jingbo Zhou, Jintai Chen, Danny Z Chen, and Jian Wu. Robust training of graph neural networks via noise governance. In *WSDM*, 2023.

[23] Luana Ruiz, Fernando Gama, and Alejandro Ribeiro. Gated graph recurrent neural networks. *IEEE Transactions on Signal Processing*, 2020.

[24] Mingchen Sun, Kaixiong Zhou, Xin He, Ying Wang, and Xin Wang. Gppt: Graph pre-training and prompt tuning to generalize graph neural networks. In *KDD*, 2022.

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

[26] Aladin Virmaux and Kevin Scaman. Lipschitz regularity of deep neural networks: analysis and efficient estimation. In *NeurIPS*, 2018.

[27] Xiang Wang, Xiangnan He, Yixin Cao, Meng Liu, and Tat-Seng Chua. Kgat: Knowledge graph attention network for recommendation. In *KDD*, 2019.

[28] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In *ICML*, 2019.

[29] Huijun Wu, Chen Wang, Yuriy Tyshetskiy, Andrew Docherty, Kai Lu, and Liming Zhu. Adversarial examples on graph data: Deep insights into attack and defense. In *IJCAI*, 2019.

[30] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[31] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In *NeurIPS*, 2018.

[32] Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. Graph contrastive learning with augmentations. In *NeurIPS*, 2020.

[33] Donghan Yu, Chenguang Zhu, Yiming Yang, and Michael Zeng. Jaket: Joint pre-training of knowledge graph and language understanding. In *AAAI*, 2022.

[34] Junliang Yu, Hongzhi Yin, Xin Xia, Tong Chen, Lizhen Cui, and Quoc Viet Hung Nguyen. Are graph augmentations necessary? simple graph contrastive learning for recommendation. In *SIGIR*, 2022.

[35] Mengmei Zhang, Linmei Hu, Chuan Shi, and Xiao Wang. Adversarial label-flipping attack and defense for graph neural networks. In *2020 IEEE International Conference on Data Mining (ICDM)*, 2020.

[36] Yanfu Zhang, Shangqian Gao, Jian Pei, and Heng Huang. Improving social network embedding via new second-order continuous graph neural networks. In *KDD*, 2022.

[37] Da Zheng, Minjie Wang, Quan Gan, Zheng Zhang, and George Karypis. Learning graph neural networks with deep graph library. In *WWW*, 2020.

[38] Yuanyi Zhong, Jianfeng Wang, Jian Peng, and Lei Zhang. Boosting weakly supervised object detection with progressive knowledge transfer. In *ECCV*, 2020.

A PROOFS

A.1 Proof of Lemma 4.1 in Section 4

Proof. We observe that

$$\begin{aligned}\frac{\|g(\boldsymbol{x})-g(\boldsymbol{y})\|}{\|\boldsymbol{x}-\boldsymbol{y}\|} &= \frac{\left\|[g_i(\boldsymbol{x})-g_i(\boldsymbol{y})]_{i=1}^n\right\|}{\|\boldsymbol{x}-\boldsymbol{y}\|} \\ &= \left\|\left[\frac{|g_i(\boldsymbol{x})-g_i(\boldsymbol{y})|}{\|\boldsymbol{x}-\boldsymbol{y}\|}\right]_{i=1}^n\right\|.\end{aligned} \tag{17}$$

Furthermore, for each $i \in [n]$, $\frac{|g_i(\boldsymbol{x})-g_i(\boldsymbol{y})|}{\|\boldsymbol{x}-\boldsymbol{y}\|} \leq \mathrm{Lip}(g_i)$. Therefore, we can write

$$\begin{aligned}\mathrm{Lip}(g) &= \sup_{\boldsymbol{x}\neq\boldsymbol{y}} \frac{\|g(\boldsymbol{x})-g(\boldsymbol{y})\|}{\|\boldsymbol{x}-\boldsymbol{y}\|} \\ &= \sup_{\boldsymbol{x}\neq\boldsymbol{y}} \left\|\left[\frac{|g_i(\boldsymbol{x})-g_i(\boldsymbol{y})|}{\|\boldsymbol{x}-\boldsymbol{y}\|}\right]_{i=1}^n\right\| \\ &\leq \sup_{\boldsymbol{x}\neq\boldsymbol{y}} \left\|[\mathrm{Lip}(g_i)]_{i=1}^n\right\| = \left\|[\mathrm{Lip}(g_i)]_{i=1}^n\right\|,\end{aligned} \tag{18}$$

□

A.2 Proof of Proposition 4.2 in Section 4

Proof. Let $\boldsymbol{Y}$ denotes the output of an L-layer Mixture-of-Experts network with input $\boldsymbol{X}$. Assuming the commonly used ReLU activation function as the non-linear layer $\rho(\cdot)$, we have $\mathrm{Lip}(\rho) = 1$. First, we consider the Lipschitz constant between the hidden states of two node feature pairs output by any layer $h(\cdot)$ in $f(\cdot)$. Let $h(x_1)$ and $h(x_2)$ represent the hidden states of input feature x_1 and x_2, respectively. Since two inputs have the same activated experts, we assume each expert has a very close gate value, i.e. p_l^k, from a statistical perspective. By applying the triangle inequality, Lemma 4.1, we obtain:

$$\begin{aligned}\frac{\|h(x_1)-h(x_2)\|}{\|x_1-x_2\|} &= \frac{\left\|\sum_{k=1}^K p^k\,[h(x_1)-h(x_2)]\right\|}{\|x_1-x_2\|} \\ &\leqslant \left\|\frac{\sum_{i=1}^{F'}\sum_{k=1}^m p^k\left[h^k(x_1)_i - h^k(x_2)_i\right]}{\|x_1-x_2\|}\right\| \\ &\leqslant \left\|F' \times \max_i \frac{\sum_{k=1}^m p^k[h^k(x_1)_i - h^k(x_2)_i]}{\|x_1-x_2\|}\right\|,\end{aligned} \tag{19}$$

let $f(x)_j$ denotes the j-th column of the matrix $f(x)$. We denote $h_l^k(\cdot)$ as the k-th expert of the l-th layer in $f(\cdot)$, then by applying the conclusion from Lemma 4.1 and leveraging the Lipschitz property of the ReLU activation function, we have:

$$\frac{\left\|f(x_1)_{j,1}-f(x_2)_{j,2}\right\|}{\left\|x_{j,1}-x_{j,2}\right\|} \leqslant \prod_{l=1}^L \left\|F^{l\prime}\right\| \left\|\left[\sum_{k=1}^{K^l} p_l^k\,\mathcal{J}^k(h^l)\right]_j\right\|_\infty, \tag{20}$$

where $x_{j,1}$ and $x_{j,2}$ denote j-th nodes pair similarity features in x_1 and x_2, respectively, and $\left[\mathcal{J}(h^l)\right]_j$ represents the j-th node pair similarity feature's the Jacobian matrix of the l-th layer. Therefore, the Lipschitz constant for the MLPMoE can be expressed as:

$$\mathrm{Lip}(f) = \max_j \prod_{l=1}^L \left\|F^{l\prime}\right\| \left\|\left[\sum_{k=1}^{K^l} p_l^k\,\mathcal{J}^k(h^l)\right]_j\right\|_\infty. \tag{21}$$

In summary, we have shown that for any two input samples x_1 and x_2, the Lipschitz constant of the MLPMoE, denoted as $\mathrm{Lip}(f)$, satisfies:

$$\|Y_1 - Y_2\| \leqslant \mathrm{Lip}(f)\,\|X_1 - X_2\|, \tag{22}$$

where $\boldsymbol{Y}$ denotes the output of the MLPMoE for inputs $\boldsymbol{X}$. This inequality implies that the Lipschitz constant $\mathrm{Lip}(f)$ controls the magnitude of changes in the output based on input distribution shift. Therefore, we have established the following result:

$$\|Y_1 - Y_2\| \leqslant \prod_{l=1}^L \left\|F^{l\prime}\right\| \left\|\left[\sum_{k=1}^{K^l} p_l^k\,\mathcal{J}^k(h^l)\right]_j\right\|_\infty \|X_1 - X_2\|. \tag{23}$$

This inequality demonstrates that the Lipschitz constant of the MLPMoE, $\mathrm{Lip}(f)$, controls the magnitude of the difference in the output Y based on the difference in the input $\boldsymbol{X}$. It allows us to analyze the stability of the model's output with respect to input distribution shift. □