TECHNICAL REPORT FOR OGBN-ARXIV EXPERIMENTS

TECHNICAL REPORT

Shichao Ma*
Topology Lab, Data Intelligence Department
OPPO Research Institute
Shenzhen, China
mashichao@oppo.com

November 2, 2022

ABSTRACT

In this technical report, we build a two-tower model framework **LGGNN** to full use the data in semi-supervised classification task of **ogbn-arxiv**. For the local tower part, it captures the local context with original graph, and the global tower part, a knn graph is built with node features to capture the global context. GraphSAGE model is used as backbone in our to generate the local & global representation for each node. Some commonly used techniques are also involved in our experiments, such as BoT, C&S, Self-KD. The implementation is public available [2]

Keywords Node Classification · Graph Neural Networks · Semi-Supervised Learning

1 Introduction

The **ogbn-arxiv** dataset is a directed graph, representing the citation network between all Computer Science (CS) arXiv papers indexed by MAG [Wang et al., 2020]. Each node is an arXiv paper and each directed edge indicates that one paper cites another one. Beside the relational data from the citations, a 128-dimensional feature vector extracted from the title and abstract of each paper is also supplied by certain language model.

The task is to predict the 40 subject areas of arXiv CS papers (multi-class, semi-supervised classification), e.g., cs.AI, cs.LG, and cs.OS, which are manually determined (i.e., labeled) by the paper's authors and arXiv moderators. The evaluation metric is the classification accuracy, the higher the better.

Considering the practical scenario of helping the authors and moderators annotate the subject areas of the newly-published arXiv papers, it propose to train on papers published until 2017, validate on those published in 2018, and test on those published since 2019.

Recently, many significant progress in addressing node classification tasks on large scale homogeneous/heterogeneous graphs have been made, and the main directions are: (1) **Model Structure Design** (SAGN [Sun and Wu, 2021], NARS [Yu et al., 2021], GAMLP [Zhang et al., 2021a], SeHGNN [Yang et al., 2022], etc.), (2) **Data Augmentation** (GIANT-XRT [Chien et al., 2022], etc.) and (3) **Learning Strategies** (C&S [Huang et al., 2021], SLE [Sun and Wu, 2021], RLU [Zhang et al., 2021a], SCR [Zhang et al., 2021b] , Self-KD, BoT [Wang et al., 2021], etc.)

Here we follow the main research directions and combine the important instructions from the data analysis, a lightweight model framework, **LGGNN** (**L**ocal & **G**lobal **GNN**) has been proposed, and some training strategies also used to improve the final performance.

*Use footnote for providing further information about author (webpage, alternative address)—*not* for acknowledging funding agencies.

[2] https://github.com/oppo-topolab/ogb-project

Table 1: Basic Graph Info of **ogbn-arxiv** Compare to other OGB [Hu et al., 2020] Citation Networks

Name	Scale	Average #Nodes	Average #Edges	Average Node Deg.	Average Clust. Coeff.	MaxSCC Ratio	Graph Diameter
arxiv	small	$169,343$	$1,166,243$	13.7	0.226	1.000	23
mag	medium	$1,939,743$	$25,582,108$	21.7	0.098	1.000	6
paper100M	large	$111,059,956$	$1,615,685,872$	29.1	0.085	1.000	25

2 Methodology

Graph Neural Network follows the **MPNN** (**M**essage **P**assing **N**eural **N**etwork) paradigm:

$$\begin{aligned} M^{(l)}[v] &= \mathbf{Msg}(\{H^{(l-1)}[u] : y \in \mathcal{N}(u)\}) \\ H^{(l)}[v] &= \mathbf{Agg}(H^{(l-1)}[v], M^{(l)}[v]) \end{aligned} \tag{1}$$

the original node features $H^{(0)} = X$ are processed after several GNN layers to get the final embedding $H^{(K)}$ which contains the full information in both node features and K-hop graph structure.

Benefit from this function of GNN, we could use the additional K-hop neighbours' information to help in node classification. Consider the extream case, no linkage between any two nodes, the GNN model degenerate to MLP (only use node features for predition). In practical scenario, the linkage between two nodes represent some relationship or similarities, e.g. **ogbn-arxiv** dataset, one paper cites another, they are likely to be in the same category.

Our idea is not to design a brand new GNN model structure, but use the basic GNN model(e.g. GraphSAGE, GAT, etc.) as backbone, extract characteristics of the **ogbn-arxiv** dataset (see Appendix A) to enhance the backbone in data augmentation and learning strategies.

Model structure – Two tower, use both local and global information from neighbors [Li et al., 2019]

2.1 KNN Graph

In the original citation network, paper nodes can use K-hop neighbors' information to enhance the representation, which captures the **local context**. But it shows from the **ogbn-arxiv** dataset:

- some paper nodes do not link to others – with zero in degrees in graph G
- some paper nodes with low homophily – very little nodes in their K-hop have the same label

It is quite common in practise, when researchers write paper, they not only cite papers in the same category, but refer to some paper in other categories as well. In the above two situtations, no local context can be used, or will bring negative effect to the GNN used, for GNN's idea is 'smoothing' in K-hop.

From the aspect of data augmentation, some other information need to be added to overcome the limitation of original citation network. Papers in the same category may use the same "bag" of words to describe their ideas, we usually call these 'bags' as glossary. So papers use the same "bag" of words are likely to be in the same category, although they do not cite each other.

We use the node features extracted from the title and abstract of paper as the "bag", and build the linkage between them via a KNN Graph K_G, which captures the **global context** supplement to the **local context** from G.

The node similarity is quantified as node feature similarity (e.g. Cosine Similarity):

$$S(u, v) = \cos(X_u, X_v) = \frac{< X_u, X_v >}{||X_u||_2||X_v||_2} \tag{2}$$

For the original features X in **ogbn-arxiv**, they just use the simple average of word vectors of the title and abstract, result in over high $(0.83+)$ simiarity in average edges. We use the GIANT-XRT [Chien et al., 2022] embeddings X_{emb} to calculate the similarity.

The fast algorithm to find Top-K neighbors, we follows the method in GAS [Li et al., 2019], and python toolkit **pynndescent** helps to build the KNN Graph in advance.

The hyper-parameter K is chosen to match the **ogbn-arxiv** dataset, for the average degree showing in Table 1, we choose $K = 20, 30$ in experiments. The Top-K similar neighbors may also be connected in original graph G, replicated edges will be dropped in KNN Graph, make sure that K_G contains purely global information.

More details in building the KNN Graph will be find in Appendix B

2.2 Two Tower GNN Model

Combine the local context learn with $\mathbf{GNN}_L$ and global context learn with $\mathbf{GNN}_G$ to make the final prediction:

$$\begin{aligned} Z_L &= \mathbf{GNN}_L(G, X_{emb}) \\ Z_G &= \mathbf{GNN}_G(K_G, X_{emb}) \\ Y &= \mathbf{MLP}(Concat(Z_L, Z_G)) \end{aligned} \tag{3}$$

The local tower and global tower GNN model can be different type or same type with different structure, here we use GraphSAGE with different convolution layers for simplicity.

As other multi-class node classification task, Cross Entropy Loss is used to train the two-tower model end to end.

$$\mathcal{L} = -\sum_{v \in D_{train}} y_v \log(\hat{y}_v) + (1 - y_v) \log(1 - \hat{y}_v) \tag{4}$$

2.3 Training Strategis

2.3.1 Label Reuse

For some nodes with very low homophily in labels, it's hard to distinguish them only use feature and graph structure. Commonly, nodes with certain labels linked to each other, so heterogeneity in labels can be used to predict the labels of node as well.

In order to predict the label of a central node Y_c, its features X_c, features of neighbors in K-hop $\{X_n : n \in \mathcal{N}^K_{hop}(c)\}$ and labels of neighbors in K-hop $\{Y_n : n \in \mathcal{N}^K_{hop}(c)\}$ all can be used. Followed **label reuse**:

- randomly mask the training set: $D_{train} \rightarrow D^0_{train}, D^1_{train}$
- use one hot labels $\tilde{Y}$ as extra features in D^1_{train} and set others($D^0_{train}, D_{val}, D_{test}$) to zero.
- $X_{new} = [X, \tilde{Y}]$, train model with $G, K_G, X_{new}; Y$

we train the LG-GNN model in a label-reuse fashion.

2.3.2 Self-KD

Knowledge distillation is often being used in the context of model compression, as transferring "knowledge" from a large model(teacher) to a more compact one(student). Several works [Zhang et al., 2019, Kim et al., 2020] have shown that a self-distilled student can outperform the teacher on held-out data. Benefit from this, self-distillation are frequently used to boost the performance on test set in OGB Node Property Prediction Tasks

The procedure of self-distillation is shown in Figure1 as an example:

- First, pre-train the 'teacher' model, use the true labels, and save the predictions.
- Then, train(distill) the 'student' model use both the hard target and soft target from 'teacher'

Loss function in Self-KD

$$\mathcal{L} = \alpha \mathcal{L}_{KD} + (1 - \alpha)\mathcal{L}_{CE} \tag{5}$$

$\mathcal{L}_{CE}$ is the usual Cross Entropy loss, and $\mathcal{L}_{KD}$ is the Kullback-Leibler divergence between the scaled student and teacher logits Eq(6)

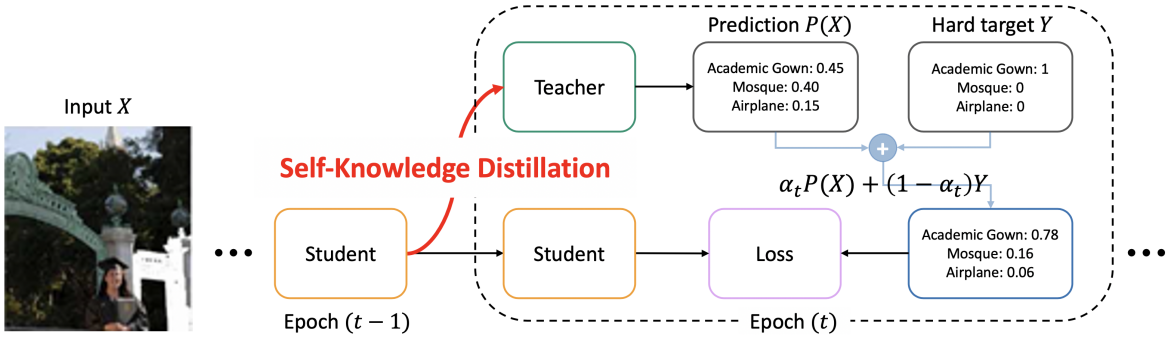


Figure 1: Self-Knowledge Distillation [Kim et al., 2020]

$$\mathcal{L}_{KD}(P||Q) = \tau^2 \sum_j \sigma_j(P/\tau) \log \frac{\sigma_j(P/\tau)}{\sigma_j(Q/\tau)} \tag{6}$$

2.3.3 Correct & Smooth

Correct & Smooth[Huang et al., 2021] is a another useful technique for post-process, that could correct the predictions of GNNs in a 'label propagation' fashion.

Algorithm 1: Correct & Smooth

Input: Z, Y, S, μ, N_C, N_S

Output: $\hat{Y}$

Error Correlation;
$E_{L_t} \leftarrow Y_{L_t} - Z_{L_t}$;
$E_{L_v} \leftarrow 0$;
$E_U \leftarrow 0$;
// Solve $\hat{E} = \arg\min_{W \in \mathbb{R}^{n \times c}} \mathrm{tr}(W^T(I-S)W) + \mu||W-E||_F^2$ via iteration
$K \leftarrow 0$;
$\alpha \leftarrow 1/(1+\mu)$;
while $K \leq N_C$ **do**
  $E \leftarrow (1-\alpha)E + \alpha SE$;
  $K \leftarrow K+1$
end
$Z^{(r)} \leftarrow Z + \hat{E}$;
Prediction Correlation;
$H_{L_t} \leftarrow Y_{L_t}$;
$H_{L_v \cup U} \leftarrow Z^{(r)}_{L_v \cup U}$;
// Solve $\hat{H} = \arg\min_{W \in \mathbb{R}^{n \times c}} \mathrm{tr}(W^T(I-S)W) + \mu||W-H||_F^2$ via iteration
$K \leftarrow 0$;
while $K \leq N_S$ **do**
  $H \leftarrow (1-\alpha)H + \alpha SH$;
  $K \leftarrow K+1$
end
$\hat{Y} \leftarrow H^{(t)}$

As mentioned in [Huang et al., 2021], using validation labels boosts performance even further, that is not written in the Algorithm1, we will show more details in experiments.

Table 2: Best Performance of Model

Model	Test Acc	Val Acc	#Params
LG	0.7523 ± 0.0020	0.7652 ± 0.0005	$1,120,680$
LG+LabelReuse(1)	0.7561 ± 0.0020	0.7684 ± 0.0005	$1,161,640$
LG+LabelReuse(2)	0.7564 ± 0.0019	0.7681 ± 0.0005	$1,161,640$
LG+LabelReuse(1)+CS	$\mathbf{0.7570 \pm 0.0018}$	0.7687 ± 0.0005	$1,161,640$
LG+LabelReuse(1)+Self-KD	0.7555 ± 0.0010	0.7653 ± 0.0007	$1,161,640$

3 Experiments

3.1 Experiment Setup

Environment: DGL 0.9, PyTorch 1.10.2, CUDA 11.3, pynndescent 0.5.6, Numpy 1.21.5, Scipy 1.7.3

For the C&S module, we use the DGL official C&S Example , and for the Self-KD module, we reuse "GAT+label reuse+self KD" instance.

In order to full utilize the GPU, full batch training for **ogbn-arxiv** is used to train the two-tower model.

Label distribution drift between train and validation/test set, we add weights to adjust the loss function in experiments, see AppendixA.3 for more details. But in the formal runs in Table2, to following the rules of OGB[3], no sample weights based on labels in validation set is added.

3.2 Best Performance

Our experiment results are shown in Table2 with the hyper-parameters tuned in advance via a greedy Hyper-Parameter Search in AppendixC.3.

Hyper-parameters used: $L\_Layers = 3$, $G\_Layers = 1$, $HD = 256$, $LR = 2e - 4$

The best performance got from Local&Global Model with Label Reuse(just 1 iterations) and post processing with C&S in train data set. We run the model 10 times with seed from $0 - 9$ to get the mean and standard deviation of test accuracy with ogb evaluator.

4 Conclusion

In this technical report, we study the node classification task by comprehensively exploring citation topologies and labels relationship in academic scenario. For the heterogeneity and sparsity commonly occur in practical situation, a **KNN graph** is built as an auxiliary structure to capture the global context which supplies to the local context using the original graph, we call this two-tower model structure **LGGNN**. Several training techniques have also been used to boost the performance, such as **Label Reuse**, **Self-KD** and **Correct&Smooth**.

This work shows that data augmentation is usually useful in solving practical problem, for the observation data always with noise.

References

Kuansan Wang, Zhihong Shen, Chiyuan Huang, Chieh-Han Wu, Yuxiao Dong, and Anshul Kanakia. Microsoft academic graph: When experts are not enough. *Quantitative Science Studies*, 1:396–413, 2020.

Chuxiong Sun and Guoshi Wu. Scalable and adaptive graph neural networks with self-label-enhanced training. *arXiv: Learning*, 2021.

Lingfan Yu, Jiajun Shen, Jinyang Li, and Adam Lerer. Scalable graph neural networks for heterogeneous graphs. *arXiv: Learning*, 2021.

Wentao Zhang, Ziqi Yin, Zeang Sheng, Wen Ouyang, Xiaosen Li, Yangyu Tao, Zhi Yang, and Bin Cui. Graph attention multi-layer perceptron. *knowledge discovery and data mining*, 2021a.

[3] `https://github.com/snap-stanford/ogb/issues/73#issuecomment-707258886`

Xiaocheng Yang, Mingyu Yan, Shirui Pan, Xiaochun Ye, and Dongrui Fan. Simple and efficient heterogeneous graph neural network. *ArXiv*, abs/2207.02547, 2022.

Eli Chien, Wei-Cheng Chang, Cho-Jui Hsieh, Hsiang-Fu Yu, Jiong Zhang, Olgica Milenkovic, and Inderjit S. Dhillon. Node feature extraction by self-supervised multi-scale neighborhood prediction. *ArXiv*, abs/2111.00064, 2022.

Qian Huang, Horace He, Abhay Singh, Ser-Nam Lim, and Austin R. Benson. Combining label propagation and simple models out-performs graph neural networks. *ArXiv*, abs/2010.13993, 2021.

Chenhui Zhang, Yufei He, Yukuo Cen, Zhenyu Hou, and Jie Tang. Improving the training of graph neural networks with consistency regularization, 2021b.

Yangkun Wang, Jiarui Jin, Weinan Zhang, Yong Yu, Zheng Zhang, and David Wipf. Bag of tricks for node classification with graph neural networks. *arXiv: Learning*, 2021.

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *neural information processing systems*, 2020.

Ao Li, Zhou Qin, Runshi Liu, Yiqun Yang, and Dong Li. Spam review detection with graph convolutional networks. *conference on information and knowledge management*, 2019.

Linfeng Zhang, Jiebo Song, Anni Gao, Jingwei Chen, Chenglong Bao, and Kaisheng Ma. Be your own teacher: Improve the performance of convolutional neural networks via self distillation. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 3712–3721, 2019.

Kyungyul Kim, Byeongmoon Ji, Doyoung Yoon, and Sangheum Hwang. Self-knowledge distillation: A simple way for better generalization. *arXiv e-prints*, 2020.

A Data Analysis

A.1 Node Label Similarity

Nodes with labels $Y \in \mathbb{R}^{n\times c}$, we calculate the same label ratio μ in each node's 1st order neighbors(1-hop), $\mu \in [0, 1]$ represent the homophily – when it's close to 1.0, the same label nodes surround the central node, when it's close to 0.0, little same label node exists.

For each node is also divided into three data set – train, validation and test by time, we group those nodes to discover the distribution of μ, showing in Figure2. Nodes with zero in degrees are removed from this statistics.

Three kurtosis shows in the distribution – 0.0, 0.5 and 1.0. Many nodes(near $14,000$ in train set, $4,000$ in validation set, and $6,000$ in test set) got $\mu = 0.0$, no nodes with the same label in their 1st order neighbors, it's hard to use usual GNN for the prediction.

A.2 Feature Similarity

Node features $X \in \mathbb{R}^{n\times d}$, for every edge in citation network, we calculate the cosine similarity of features of end nodes to get λ. $\lambda \in [-1, 1]$, but in the citation network, $\lambda > 0$. For extracted by the simple average of word vectors, the original node features are over alike to each other showing in Figure3(a), the mean is $\bar{\lambda} = 0.83$.

Over similar node features, combined with low homophily, makes it more difficult to distinguish the node labels. So GIANT-XRT embeddings is more appropriate to use to boost the performance, the distribution of simiarity λ with larger variance and lower mean Figure3(b).

A.3 Labels Distribution Cross Time

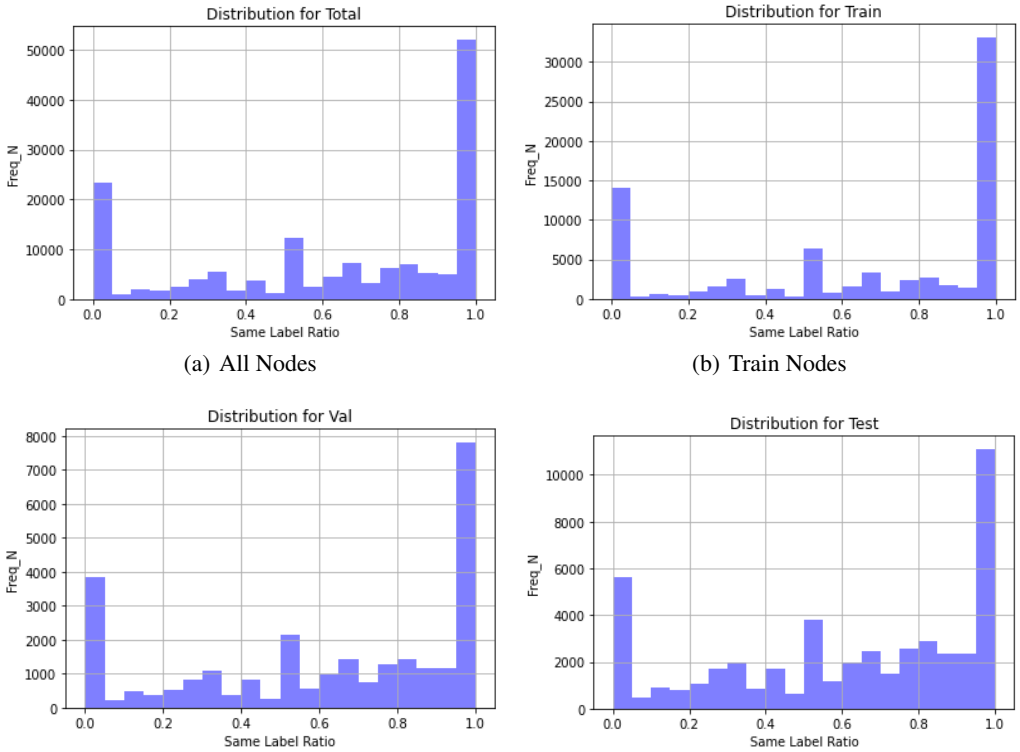
In **ogbn-arxiv**, 40 subject areas are needed to be predicted. The labels distribution over 40 classes is imbalanced, and the distributions cross three data sets are not identical Figure4. The hottest subjects in train set(before 2017) are 28, 16 and 34, but in validation set(2018) and test set(2019) they changeed to 16, 24 and 30.

We use TGI(Target Group Index) to adjust the influence of distribution dirft of labels,

$$TGI(X, Y) = X \oslash Y \tag{7}$$

X, Y are two discrete distributions over the same sample space, Y is used for the baseline, $\oslash$ is element-wise division.

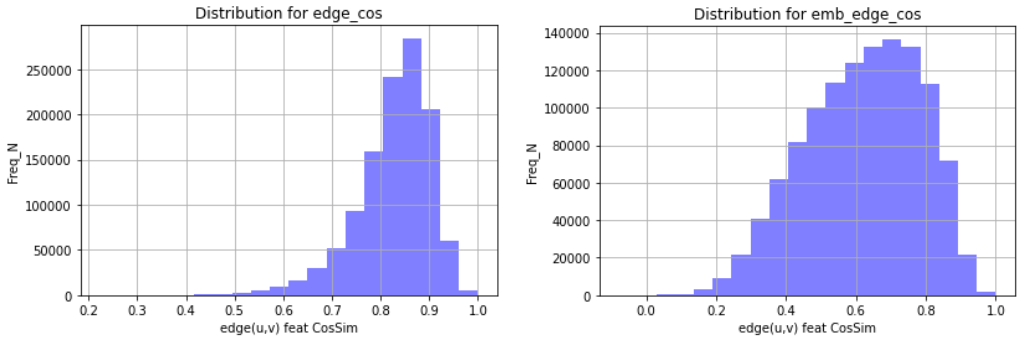
Five runs experiments Table3 have been done to validate the efficiency of weight adjustment for loss. We just use pure "LGGNN" model without any training techniques.



(a) All Nodes (b) Train Nodes

(c) Val Nodes (d) Test Nodes

Figure 2: Node Label Local Similarity

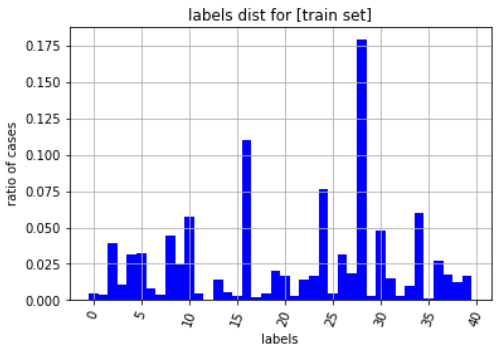


(a) Original Features (b) GIANT-XRT Embeddings

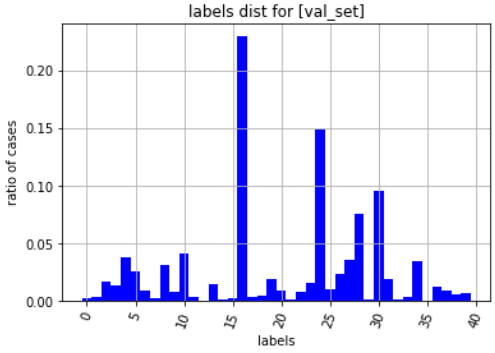
Figure 3: Node Feature Local Similarity

Table 3: TGI Weight Adjustment for Loss

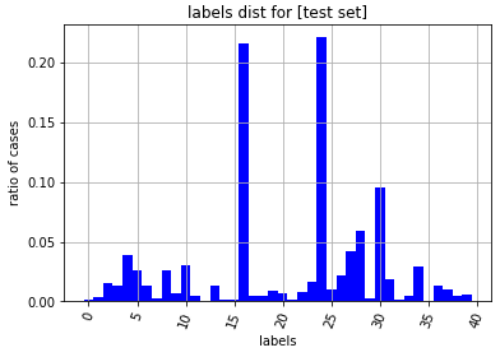
Weight Adj	Run	Val Acc	Test Acc
with	1	0.7584	0.7489
	2	0.7587	0.7490
	3	0.7611	0.7507
	4	0.7608	0.7491
	5	0.7602	0.7481
	mean	0.7598	**0.7492**
	std	0.0011	0.0008
without	1	0.7622	0.7506
	2	0.7620	0.7515
	3	0.7619	0.7472
	4	0.7610	0.7485
	5	0.7618	0.7460
	mean	0.7618	0.7488
	std	0.0004	0.0020



(a) Train Set Labels(before 2017)



(b) Val Set Labels(2018)



(c) Test Set Labels(2019)

Figure 4: Label Distribution Cross Time

Table 4: Self-KD for LGGNN

α	τ	Val Acc	Test Acc
0.95	0.7	0.7590 ± 0.0010	0.7502 ± 0.0022
0.60	0.7	0.7615 ± 0.0008	0.7515 ± 0.0021
0.20	0.7	0.7629 ± 0.0010	0.7523 ± 0.0014

Table 5: Models Performance without 'Self Loop'($L\_Layers = 3$, $G\_Layers = 1$, $HD = 256$, $LR = 2e-3$)

Model	Test Acc	Val Acc	#Params
LG	0.7478 ± 0.0013	0.7621 ± 0.0007	$1,120,680$
LG+LabelReuse(1)	0.7527 ± 0.0018	0.7639 ± 0.0005	$1,161,640$
LG+LabelReuse(2)	0.7531 ± 0.0016	0.7636 ± 0.0004	$1,161,640$
LG+LabelReuse(1)+CS	$\mathbf{0.7527 \pm 0.0016}$	0.7646 ± 0.0006	$1,161,640$
LG+LabelReuse(1)+Self-KD	0.7502 ± 0.0022	0.7590 ± 0.0010	$1,161,640$

B KNN Graph

Adjacency matrix A_{knn} is built with the output of k-nearest neighbor index, generated from pynndescent. Use the matrices operations, easily remove the duplicate edges exist in original graph, which is also represented by the adjacency matrix A_o.

$$A_{new} = A_{knn} \odot (A_{knn} - A_o) \tag{8}$$

$\odot$ for element-wise multiplication.

C Experiment Details

C.1 Self-KD

In Self-KD, two hyper-parameters α and τ influence the performance. We choose $\alpha = 0.2, 0.6, 0.95$ and $\tau = 0.7$ in our experiments when train student model from teacher "LG+LabelReuse(1)"

But in our experiments, Self-KD did not boost the performance in test set. Table4 shows the results with Self-KD.

C.2 Some Experiments Records

Some conclusion in our experiments:

- 'Self Loop' is a common trick, always helps Table6
- Label Reuse and C&S are helpful in boosting model performance showing in Table5 and Table6
- Self-KD do not work well in our case, under performance of C&S
- 'Learning Rate' influence the performance out of our expectation, compared by the formal result in Table2($LR = 2e-4$) and experiment result in Table6($LR = 2e-3$)

C.3 Hyper-Parameter Tuning

The hyper-parameters of the LGGNN model is tuned in a greedy manner – fix the other parameters and search for one, then next. The search space is listed in Table7, we only search parameters for the original model – without label reuse and other techniques.

C.3.1 GNN Layers

Local Tower GNN Layers(L_Layers) v.s. Global Tower GNN Layers(G_Layers)

Table 6: Models Performance with 'Self Loop'($L\_Layers = 3$, $G\_Layers = 1$, $HD = 256$, $LR = 2e-3$)

Model	Test Acc	Val Acc	#Params
LG	0.7500 ± 0.0020	0.7601 ± 0.0009	$1,120,680$
LG+LabelReuse(1)	0.7539 ± 0.0023	0.7639 ± 0.0005	$1,161,640$
LG+LabelReuse(2)	0.7510 ± 0.0024	0.7635 ± 0.0007	$1,161,640$
LG+LabelReuse(1)+CS(T)	$\mathbf{0.7541 \pm 0.0018}$	0.7648 ± 0.0005	$1,161,640$
LG+LabelReuse(1)+Self-KD	0.7500 ± 0.0009	0.7589 ± 0.0009	$1,161,640$

Table 7: The search space of hyper-parameters of model

Hyper-Parameters	Candidates
Local GNN Layers(L_Layers)	$\{1, 2, 3^*\}$
Global GNN Layers(G_Layers)	$\{1^*, 2, 3\}$
Hidden Dimension(HD)	$\{128, 192, 256^*\}$
Learning Rate(LR)	$\{2e-2, 2e-3, 2e-4^*\}$

Since the global part(KNN Graph) is more dense, and it's auxiliary to the local part, we tuned the GNN Layers parameters with the constraints: $L\_Layers \geq G\_Layers$.

The performance is shown in Table8, large local GNN layers($L\_Layers = 3$) gives sufficient message passing in K-hop, KNN Graph supplies the global information that do not need too much aggregation($G\_Layers = 1$)

C.3.2 Hidden Dimension

Table9

C.3.3 Learning Rate

Table10

Table 8: L_layers vs G_layers

L_layers	G_layers	Test Acc	Val Acc	#Params
3	**1**	$\mathbf{0.7500 \pm 0.0020}$	$\mathbf{0.7601 \pm 0.0009}$	$1,120,680$
3	2	0.7476 ± 0.0042	0.7585 ± 0.0008	$1,252,008$
3	3	0.7463 ± 0.0037	0.7557 ± 0.0008	$1,383,336$
2	1	0.7480 ± 0.0031	0.7596 ± 0.0008	$989,352$
2	2	0.7481 ± 0.0038	0.7580 ± 0.0009	$1,120,680$
1	1	0.7470 ± 0.0009	0.7579 ± 0.0007	$858,024$

Table 9: Hidden Dimension

HD	Test Acc	Val Acc	#Params
256	$\mathbf{0.7500 \pm 0.0020}$	$\mathbf{0.7601 \pm 0.0009}$	$1,120,680$
192	0.7487 ± 0.0041	0.7597 ± 0.0011	$779,080$
128	0.7503 ± 0.0028	0.7600 ± 0.0008	$478,440$

Table 10: Learning Rate

LR	Test Acc	Val Acc
$2e-2$	0.7150 ± 0.0020	0.7232 ± 0.0031
$2e-3$	0.7500 ± 0.0020	0.7601 ± 0.0009
$\mathbf{2e-4}$	$\mathbf{0.7523 \pm 0.0020}$	$\mathbf{0.7652 \pm 0.0005}$