2021

DataFunSummit

图机器学习

在线峰会

GNN基础模型论坛

2021.10.10 , 09:00 - 17:30

Graph Attention Multi-Layer Perceptron

张文涛 腾讯angel graph/北京大学 博士生

目录

CONTENTS

01 Motivation

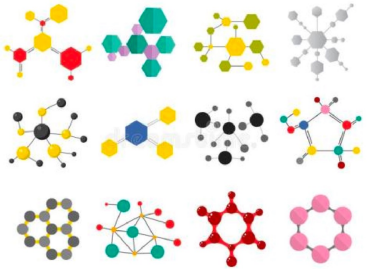
—— GNN的应用和局限

Applications of GNN

Many real-world data are graphs.

Social Network

Knowledge Graph



Drugs and New materials

- Graph Neural Network (GNN) has achieved great success in many graph-based applications.

- GNN has the ability to enhance its node feature (or representation in each layer) with the adjacent nodes, and thus improve the model performance.

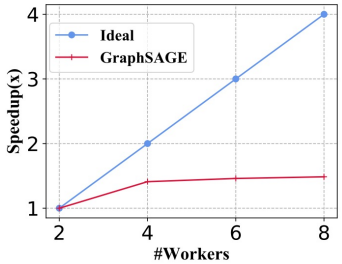
Low Scalability of GNN

1. High Memory/Time Cost in Single Machine

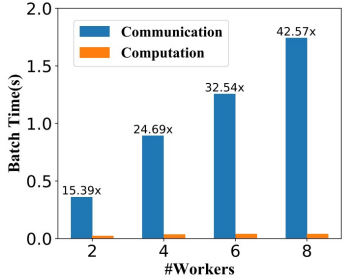
- Hard to load the feature and adjacent matrix due to limited GPU memory.
- The matrix multiplication is time-consuming.

2. High Communication Cost in Distributed Setting

- k-layer GNNs have to pull and aggregate the graph embedding of the k-hop neighbors of nodes in each batch training.
- The speedup is unsatisfactory due to the high communication cost.



Speedup

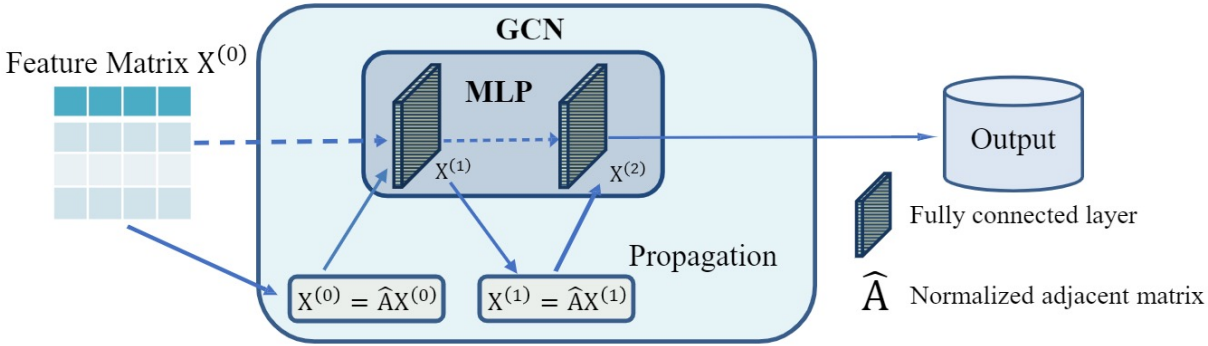


Bottleneck

Zhang W, Shen Y, Lin Z, et al. *GMLP: Building Scalable and Flexible Graph Neural Networks with Feature-Message Passing*[J]. arXiv preprint arXiv:2104.09880, 2021.

Low Flexibility of GNN

1. Restricting $D_p = D_t$

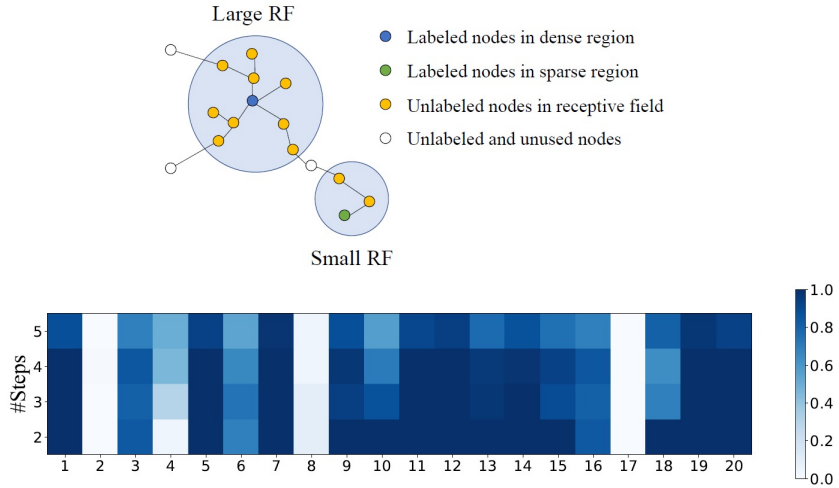


- GNN has two depth, the propagation depth D_{p} and the transformation depth D_t.
- We need large D_{p} for sparse graphs and large D_t for large graphs.

Zhang W, Sheng Z, Jiang Y, et al. *Evaluating deep graph neural networks*[J]. arXiv preprint arXiv:2108.00955, 2021.

Low Flexibility of GNN

2. Inconsistent propagation speed



- Nodes in the dense region have large RF with 2-step propagation.
- Different nodes require different propagation steps.

Zhang W, Yin Z, Sheng Z, et al. *Graph Attention Multi-Layer Perceptron*[J]. arXiv preprint arXiv:2108.10097, 2021.

02 Related Works

Scalable GNNs

1. Sampling

- Graph-wise sampling: Cluster-GCN, GraphSAINT
- Layer-wise sampling: Fast-GCN, AS-GCN
- Node-wise sampling: GraphSAGE, VR-GCN

2. Model decoupling

- SGC
- SIGN
- GBP
- S^2GC

Our method follow on the second type since it is more efficient and scalable.

SGC(ICML 19)

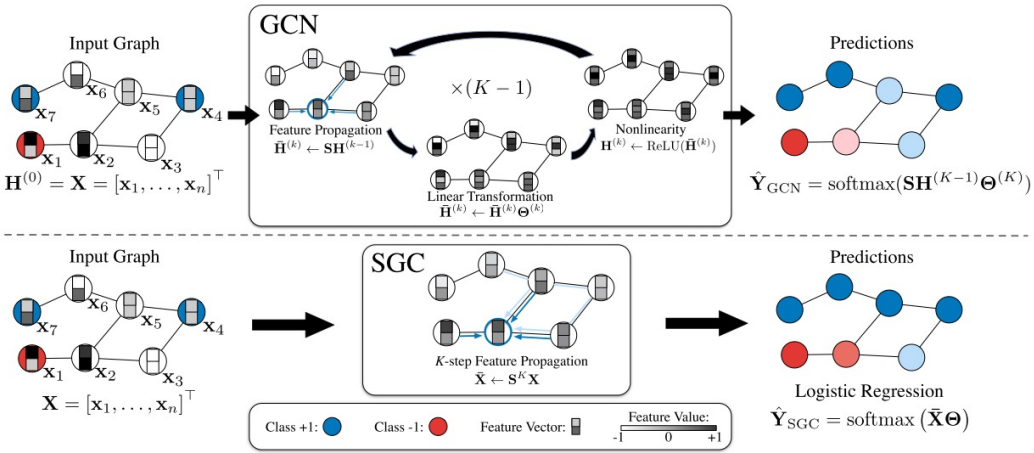


Figure 1. Schematic layout of a GCN v.s. a SGC. *Top row:* The GCN transforms the feature vectors repeatedly throughout K layers and then applies a linear classifier on the final representation. *Bottom row:* the SGC reduces the entire procedure to a simple feature propagation step followed by standard logistic regression.

Graph-wise Propagation

- SGC first propagates the feature and then adopt a LR to generate the predictions.
- It would encounter the over-smoothing issue when the propagation step is large.

Wu F, Souza A, Zhang T, et al. *Simplifying graph convolutional networks*[C]//International conference on machine learning. PMLR, 2019: 6861-6871.

SIGN (Twitter)

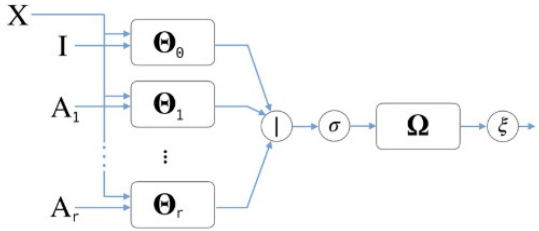


Figure 1: The *SIGN* architecture for r generic graph filtering operators. $\mathbf{\Theta}_k$ represents the k-th dense layer transforming node-wise features downstream the application of operator k, $|$ is the concatenation operation and $\mathbf{\Omega}$ refers to the dense layer used to compute final predictions.

Layer-wise Propagation

SIGN further improve SGC by adding more diffusion operators, such as PageRank-based and Triangle-based normalized adjacency matrices.

- Pros: features with different propagation steps can be fully utilized.
- Cons: some nodes need smaller receptive field while other nodes need it to be larger. SIGN simply concatenates the global and local representations, ignoring the individual properties of each node.

Frasca F, Rossi E, Eynard D, et al. *Sign: Scalable inception graph neural networks*[J]. arXiv preprint arXiv:2004.11198, 2020.

DAGNN(KDD 20)

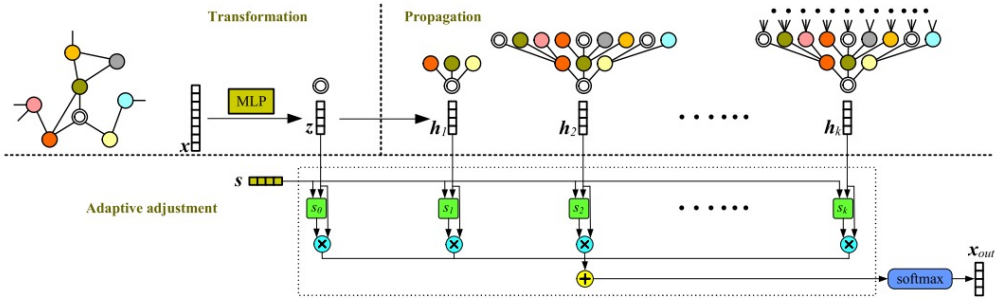


Figure 5: An illustration of the proposed Deep Adaptive Graph Neural Network (DAGNN) . For clarity, we show the pipeline to generate the prediction for one node. Notation letters are consistent with Eq.(8) but bold lowercase versions are applied to denote representation vectors. s is the projection vector that computes retainment scores for representations generating from various receptive fields. s_0, s_1, s_2, and s_k represent the retainment scores of z, h_1, h_2, and h_k, respectively.

Node-wise Propagation

It can be seen as a particular case of label propagation since it propagates the MLP predictions and weighted add the multiple predictions under different propagation depths.

- The MLP prediction and the label propagation can not be decoupled and DAGNN is trained in an end-to-end manner. Correspondingly, its scalability is limited.
- One global vector is adopted to train the weight of different propagated label, and this simple vector may be hard to learn the interactions between the propagated labels.

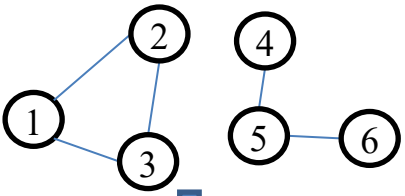
Liu M, Gao H, Ji S. *Towards deeper graph neural networks*[C]//Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. 2020: 338-348.

03 Node Dependent Local Smoothing

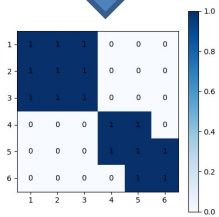
Over-Smoothing issue

➢ GNN smooths the representation of each node via node propagation.

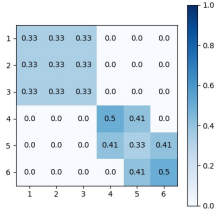
$\widetilde{D}$: the diagonal node degree matrix with added self loops



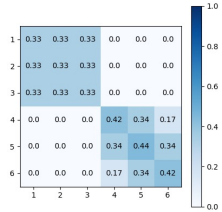
$$\widehat{A} = \widetilde{D}^{r-1}\widetilde{A}\widetilde{D}^{-r}, \quad X^{(\infty)} = \widehat{A}^{\infty}X, \quad \widehat{A}^{\infty}_{i,j} = \frac{(d_i+1)^r(d_j+1)^{1-r}}{2m+n}$$



A

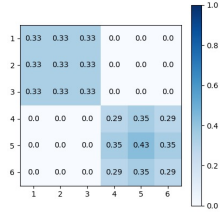


$\left(\widetilde{D}^{-1/2}A\widetilde{D}^{-1/2}\right)^1$



$\left(\widetilde{D}^{-1/2}A\widetilde{D}^{-1/2}\right)^2$

…



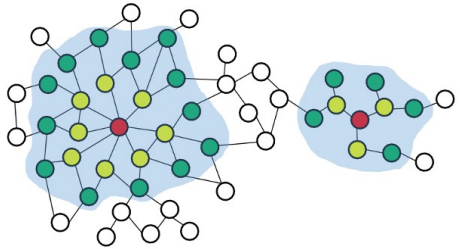
$\left(\widetilde{D}^{-1/2}A\widetilde{D}^{-1/2}\right)^{26}$

The final smoothed feature is over-smoothed (indistinguishable) and unable to capture the full graph structure information since it only relates with the node degrees of target nodes and source nodes.

An explanation from the receptive field

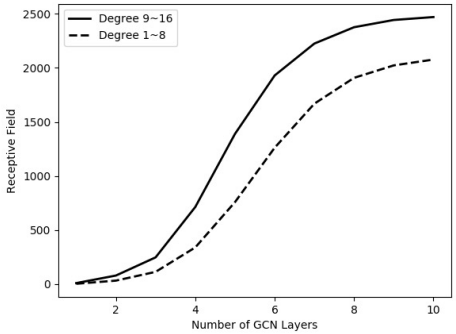
Definition:

The Receptive Field (RF) of a node refers to a set of nodes including the node itself and the neighboring nodes it has access to.



Expansion speed of the receptive field:

The expansion speed of RF of nodes in the dense region (in Cora) is much faster than the one of nodes in the sparse region.



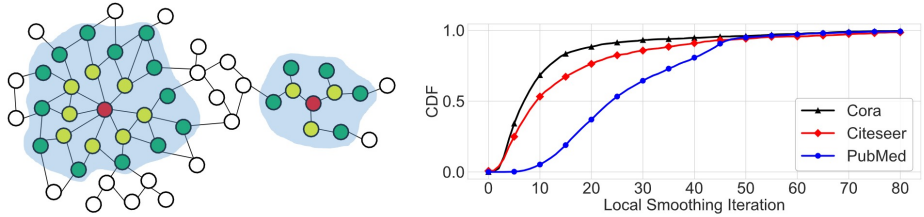
limitations

Under-smoothing vs Over-smoothing

- k-layer GNN can only capture the information within k-hop neighborhood. When adopting a shallow architecture, GNNs neglect plenty of helpful information existed in long-range dependencies.

- However, simply stacking multiple existing GNN layers leads to the over-smoothing issue while nodes in the dense region receive much noise from unrelated nodes in the excessively large receptive field.

NDLS(NeurIPS 2021 , Spotlight)



(a) Two nodes with different local structures (b) The CDF of LSI in different graphs

Figure 1: (Left) The local graph structures for two nodes in different regions; the node in dense region has larger smoothed area within two iterations of propagation. (Right) The CDF of LSI in three citation networks.

Challenge: how to control the smoothness in a node dependent way ?

Definition: Local-Smoothing Iteration(LSI, parameterized by ϵ) is defined as

$$K(i,\epsilon) = \min\left\{k: \left\|\tilde{I}_i - I(k)_i\right\|_2 < \epsilon\right\},$$

Where ϵ is an arbitrary small constant with $\epsilon > 0$,

$$I(k)_{i,j} = \frac{\partial \widehat{\mathrm{X}}_{ih}^{(k)}}{\partial \widehat{\mathrm{X}}_{jh}^{(0)}}, \forall h \in \{1,2,\cdots,f\}, \text{and } \tilde{I} = I(\infty).$$

Zhang W, Yang M, Sheng Z, et al. *Node Dependent Local Smoothing for Scalable Graph Learning*[C]. NeurIPS, 2021.

Node Dependent Local Smoothing

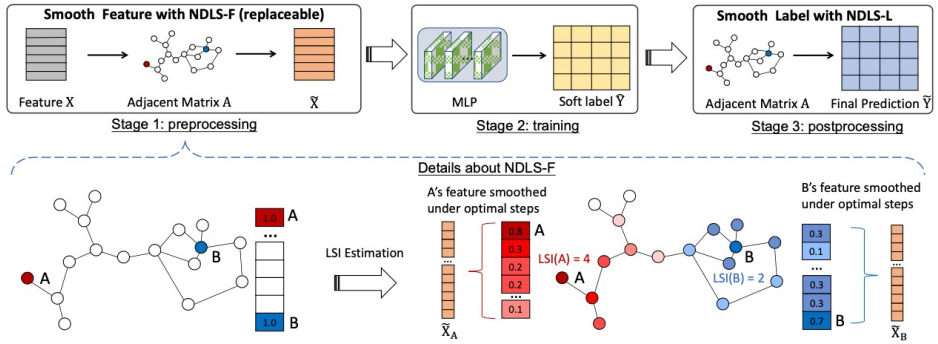


Figure 2: Overview of the proposed NDLS method, containing 1) feature smoothing with NDLS (NDLS-F); 2) model training with smoothed features; 3) label smoothing with NDLS (NDLS-L). NDLS-F and NDLS-L are pre-processing and post-processing steps, respectively.

Node Dependent Feature Propagation → Training → Node Dependent Label Propagation

All ML models (MLP, LR, XGB, etc..) can be applied

Smooth Features and labels

NDLS-F

We smooth the initial input feature X_i of node v_i with node-dependent LSI as:

$$\widetilde{\mathbf{X}}_i(\epsilon) = \frac{1}{K(i,\epsilon)+1} \sum_{k=0}^{K(i,\epsilon)} \mathbf{X}_i^{(k)}.$$

NDLS-L

Similarly, we smooth the model prediction $\widehat{\mathrm{Y}}_i^{(k)}$ of node v_i with node-dependent LSI as:

$$\widetilde{\mathbf{Y}}_i(\epsilon) = \frac{1}{K(i,\epsilon)+1} \sum_{k=0}^{K(i,\epsilon)} \hat{\mathbf{Y}}_i^{(k)}.$$

Accuracy and Efficiency Comparison

Table 3: Results of transductive settings. OOM means "out of memory".

Type	Models	Cora	Citeseer	PubMed	Industry	ogbn-papers100M
Coupled	GCN	81.8±0.5	70.8±0.5	79.3±0.7	45.9±0.4	OOM
	GAT	83.0±0.7	72.5±0.7	79.0±0.3	46.8±0.7	OOM
	JK-Net	81.8±0.5	70.7±0.7	78.8±0.7	47.2±0.3	OOM
Decoupled	APPNP	83.3±0.5	71.8±0.5	80.1±0.2	46.7±0.6	OOM
	AP-GCN	83.4±0.3	71.3±0.5	79.7±0.3	46.9±0.7	OOM
	PPRGo	82.4±0.2	71.3±0.5	80.0±0.4	46.6±0.5	OOM
	DAGNN (Gate)	84.4±0.5	73.3±0.6	80.5±0.5	47.1±0.6	OOM
	DAGNN (NDLS-L)*	84.4±0.6	73.6±0.7	80.9±0.5	47.2±0.7	OOM
Linear	MLP	61.1±0.6	61.8±0.8	72.7±0.6	41.3±0.8	47.2±0.3
	SGC	81.0±0.2	71.3±0.5	78.9±0.5	45.2±0.3	63.2±0.2
	SIGN	82.1±0.3	72.4±0.8	79.5±0.5	46.3±0.5	64.2±0.2
	S^2GC	82.7±0.3	73.0±0.2	79.9±0.3	46.6±0.6	64.7±0.3
	GBP	83.9±0.7	72.9±0.5	80.6±0.4	46.9±0.7	65.2±0.3
Linear	NDLS-F+MLP*	84.1±0.6	73.5±0.5	81.1±0.6	47.5±0.7	65.3±0.5
	MLP+NDLS-L*	83.9±0.6	73.1±0.8	81.1±0.6	46.9±0.7	64.6±0.4
	SGC+NDLS-L*	84.2±0.2	73.4±0.5	81.1±0.4	47.1±0.6	64.9±0.3
	NDLS*	**84.6±0.5**	**73.7±0.6**	**81.4±0.4**	**47.7±0.5**	**65.6±0.3**

Models	Flickr	Reddit
GraphSAGE	50.1±1.3	95.4±0.0
FastGCN	50.4±0.1	93.7±0.0
ClusterGCN	48.1±0.5	95.7±0.0
GraphSAINT	51.1±0.1	96.6±0.1
NDLS-F+MLP*	51.9±0.2	96.6±0.1
GraphSAGE+NDLS-L*	51.5±0.4	96.3±0.0
NDLS*	**52.6±0.4**	**96.8±0.1**

Table 4: Results of inductive settings.

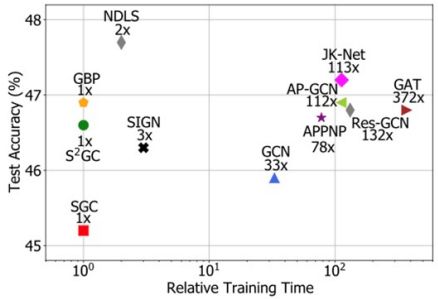
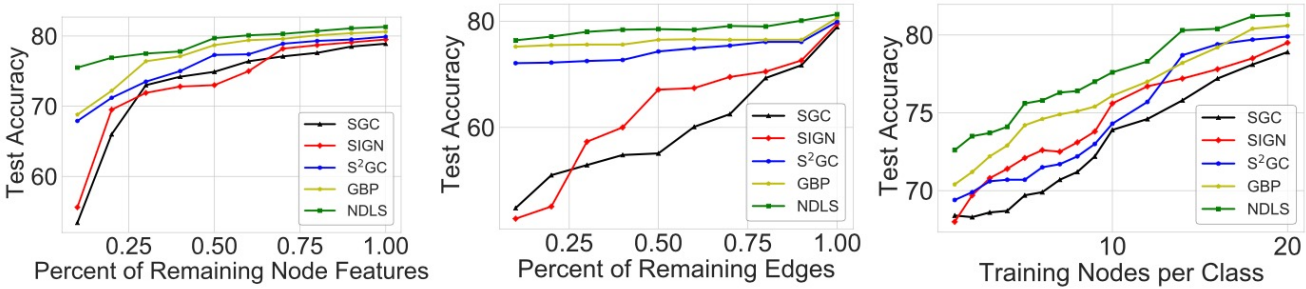


Figure 3: Performance along with training time on the Industry dataset.

- MLP with FS or LS achieve competitive performance.
- LS can be cooperated into existing methods and improve their performance, such as SGC and DAGNN.
- Both FS and LS contributes to the performance.
- NDLS gets the best accuracy while keep good efficiency.

Other Experiments



(a) Feature Sparsity (b) Edge Sparsity (c) Label Sparsity

Figure 4: Test accuracy on PubMed dataset under different levels of feature, edge and label sparsity.

Table 1: Results of different base models on PubMed.

Base Models	Models	Accuracy	Gain
MLP	Base	72.7±0.6	-
	+ NDLS-F	81.1±0.6	+ 8.4
	+ NDLS-L	81.1±0.6	+ 8.4
	+ NDLS (both)	**81.4±0.4**	+ 8.7
RF	Base	74.4±0.2	-
	+ NDLS-F	80.3±0.1	+ 5.9
	+ NDLS-L	80.0±0.2	+ 5.6
	+ NDLS (both)	**80.5±0.4**	+ 6.1
XGB	Base	74.1±0.2	-
	+ NDLS-F	81.0±0.3	+ 6.9
	+ NDLS-L	79.8±0.2	+ 5.7
	+ NDLS (both)	**81.6±0.3**	+ 7.5

- NDLS performs well and is robust on sparse graphs.
- NDLS works as a plug-and-play module, and decision tree based models can also get performance gains.

What influences LSIs?

Theorem 3.1. *Given feature smoothing* $\mathbf{X}^{(k)} = \hat{\mathbf{A}}^k\mathbf{X}$ *with* $\hat{\mathbf{A}} = \tilde{\mathbf{D}}^{-1}\tilde{\mathbf{A}}$*, we have*

$$K(i,\epsilon) \leq \log_{\lambda_2}\left(\epsilon\sqrt{\frac{\tilde{d}_i}{2m+n}}\right),$$

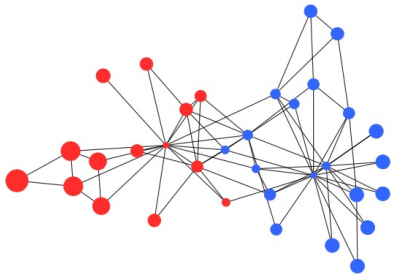
where λ_2 *is the second largest eigenvalue of* $\hat{\mathbf{A}}$*,* $\tilde{d}_i$ *denotes the degree of node* v_i *plus 1 (i.e.,* $\tilde{d}_i = d_i + 1$*), and* m*,* n *denote the number of edges and nodes respectively.*

Positively related with

- The scale of graph (m, n)
- The sparsity of graph (λ_2)

Negatively related with

- The node degree (d_i)



(b) The visualization of LSI

Theorem 3.2. *For any nodes* i *in a graph* $\mathcal{G}$*,*

$$K(i,\epsilon) \leq \max\{K(j,\epsilon), j \in N(i)\} + 1,$$

where $N(i)$ *is the set of node* v_i*'s neighbours.*

- Adjacent nodes have similar LSIs
- Nodes with super-nodes as neighbors (or neighbor's neighbors) may have small LSIs

Analysis of NDLS

Advantages:

- Scalable: less memory/communication cost
- Flexible: node-adaptive, can be incorporated into any models

Limitations:

- Despite of its high generalization ability, the generated smoothed features may be unsuitable for the downstream tasks (e.g., classification or link prediction) since NDLS cannot be trained in an end-to-end manner.
- Directly averaging the propagated features with the hop less than LSI may be unsuitable and a better alternative is adaptive weighted averaging.

GAMLP is also scalable and flexible, and solves the limitations above!

04 GAMLP

Overview of framework

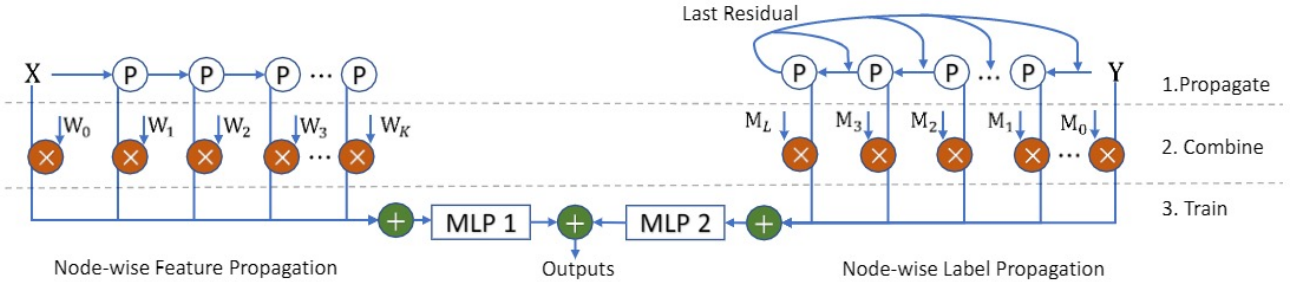


Figure 2: Overview of the proposed GAMLP, including (1) feature and label propagation, (2) combine the propagated features and labels with RF attention, and (3) MLP training. Note that both the feature and label propagation can be pre-processed.

GAMLP decomposes the end-to-end GNN training into three parts:

1. Feature/Label propagation
2. Feature/Label combination with RF attention
3. MLP training

1. Propagation

1. Node-wise Feature Propagation

We construct a parameter-free K-step feature propagation as

$$\mathbf{X}^{(k)} \leftarrow \hat{\mathbf{A}}\mathbf{X}^{(k-1)}, \ \forall k = 1, \ldots, K,$$

where $\mathrm{X}^{(k)}$ contains the features of a fixed RF of each node.

We average these propagated features in a weighted manner:

$$\mathbf{H}_{\mathbf{X}} = \sum_{k=0}^{K} \mathbf{W}_k \mathbf{X}^{(k)}$$

where $\mathbf{W}_k = Diag(\eta_k)$ is the diagonal matrix derived from vector η_k, and η_k is an N-dimension vector derived from vector $\eta_k = w_i(k), 1 \leq i \leq N$, and $w_i(k)$ measures the importance of the k-step propagated feature for node v_i.

1. Propagation

1. Node-wise Label Propagation

We construct a parameter-free K-step label propagation as

$$\mathbf{Y}^{(l)} \leftarrow \hat{\mathbf{A}}\mathbf{Y}^{(l-1)}, \ \forall l = 1, \ldots, L,$$

The propagated label $\mathrm{Y}^{(l)}$ is closer to the original label matrix $\mathrm{Y}^{(0)}$ with smaller propagation step l, and thus face a higher risk of the label leakage issue if it is directly used as the model input.

Definition：Last Residual Connection

To alleviate this, given the propagation step l, and the list of propagated labels: [$\mathrm{Y}^{(0)}$, $\mathrm{Y}^{(1)}$, $\mathrm{Y}^{(2)}$,…, $\mathrm{Y}^{(L)}$], we set

$$\hat{\mathbf{Y}}^{(l)} \leftarrow \mathbf{Y}^{(l)} + \alpha_l \mathbf{Y}^{(L)}, \ l = 1, \ldots, L, \ \ \alpha_l = \cos\left(\frac{\pi l}{2L}\right)$$

We then average these propagated labels in a weighted manner:

$$\mathbf{H}_{\mathbf{Y}} = \sum_{l=0}^{L} \hat{\mathbf{W}}_l \hat{\mathbf{Y}}^{(l)}.$$

2. Combine

Recursive Attention

We recursively measure the feature information gain compared with the previous combined features/labels as:

$$\widetilde{\mathbf{X}}_i^{(l)} = \mathbf{X}_i^{(l)} \parallel \sum_{k=0}^{l-1} w_i(k)\mathbf{X}_i^{(k)}, \quad \widetilde{w}_i(l) = \delta(\widetilde{\mathbf{X}}_i^{(l)} \cdot s), \quad w_i(l) = e^{\widetilde{w}_i(l)} / \sum_{k=0}^{K} e^{\widetilde{w}_i(k)}$$

JK Attention

The feature combination process is guided with the model prediction trained on all the propagated features/labels as:

$$\widetilde{\mathbf{X}}_i^{(l)} = \mathbf{X}_i^{(l)} \parallel \mathbf{E}_i, \quad \widetilde{w}_i(l) = \delta(\widetilde{\mathbf{X}}_i^{(l)} \cdot s), \quad w_i(l) = e^{\widetilde{w}_i(l)} / \sum_{k=0}^{K} e^{\widetilde{w}_i(k)} \qquad \mathbf{E}_i = \mathrm{MLP}(\mathbf{X}_i^{(1)} \parallel \mathbf{X}_i^{(2)} \parallel \ldots \parallel \mathbf{X}_i^{(K)})$$

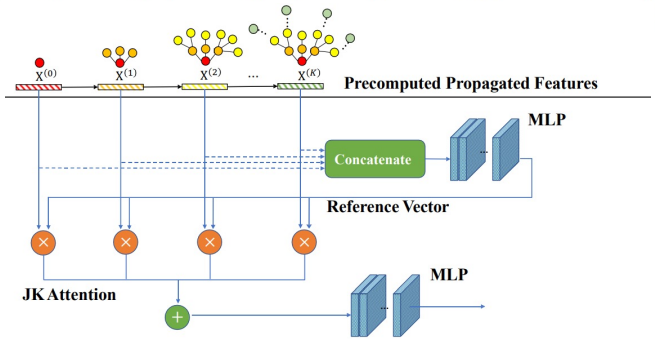


Figure 3: The architecture of GAMLP with JK Attention.

3. Training

Both the combined feature H_X and combined label H_X are transformed with MLP, and then be added to get the final output embedding:

$$\widetilde{\mathbf{H}} = \text{MLP}(\mathbf{H_X}) + \beta\text{MLP}(\mathbf{H_Y}),$$

where β is a hyper-parameter that measures the importance of the combined label.

For graphs with good features but low-quality labels (e.g., label noise or low label rate), and we should decrease β so that more attention is paid to the former one.

We adopt the Cross-Entropy (CE) measurement between the predicted softmax outputs and the one-hot ground-truth label distributions as the objective function:

$$\mathcal{L}_{CE} = -\sum_{i\in\mathcal{V}_l}\sum_{j}\mathbf{Y}_{ij}\log(\text{softmax}(\widetilde{\mathbf{H}})_{ij}),$$

05 Experiments

Setups

➢ Datasets:

Table 10: Overview of the Datasets

Dataset	#Nodes	#Features	#Edges	#Classes	#Train/Val/Test	Task type	Description
Cora	2,708	1,433	5,429	7	140/500/1000	Transductive	citation network
Citeseer	3,327	3,703	4,732	6	120/500/1000	Transductive	citation network
Pubmed	19,717	500	44,338	3	60/500/1000	Transductive	citation network
Amazon Computer	13,381	767	245,778	10	200/300/12881	Transductive	co-purchase graph
Amazon Photo	7,487	745	119,043	8	160/240/7,087	Transductive	co-purchase graph
Coauthor CS	18,333	6,805	81,894	15	300/450/17,583	Transductive	co-authorship graph
Coauthor Physics	34,493	8,415	247,962	5	100/150/34,243	Transductive	co-authorship graph
ogbn-products	2,449,029	100	61,859,140	47	196k/49k/2204k	Transductive	co-purchase graph
ogbn-papers100M	111,059,956	128	1,615,685,872	172	1207k/125k/214k	Transductive	citation network
ogbn-mag	1,939,743	128	21,111,007	349	626k/66k/37k	Transductive	citation network
PPI	56,944	50	818,716	121	45k / 6k / 6k	Inductive	protein interactions network
Flickr	89,250	500	899,756	7	44k/22k/22k	Inductive	image network
Reddit	232,965	602	11,606,919	41	155k/23k/54k	Inductive	social network

➢ Settings:

Please see https://github.com/PKU-DAIR/GAMLP/.

Transductive Performance

Table 1: Performance comparison on seven transductive datasets.

Methods	Cora	Citeseer	PubMed	Amazon Computer	Amazon Photo	Coauthor CS	Coauthor Physics
GCN	81.8±0.5	70.8±0.5	79.3±0.7	82.4±0.4	91.2±0.6	90.7±0.2	92.7±1.1
GAT	83.0±0.7	72.5±0.7	79.0±0.3	80.1±0.6	90.8±1.0	87.4±0.2	90.2±1.4
JK-Net	81.8±0.5	70.7±0.7	78.8±0.7	82.0±0.6	91.9±0.7	89.5±0.6	92.5±0.4
ResGCN	82.2±0.6	70.8±0.7	78.3±0.6	81.1±0.7	91.3±0.9	87.9±0.6	92.2±1.5
APPNP	83.3±0.5	71.8±0.5	80.1±0.2	81.7±0.3	91.4±0.3	92.1±0.4	92.8±0.9
AP-GCN	83.4±0.3	71.3±0.5	79.7±0.3	83.7±0.6	92.1±0.3	91.6±0.7	93.1±0.9
SGC	81.0±0.2	71.3±0.5	78.9±0.5	82.2±0.9	91.6±0.7	90.3±0.5	91.7±1.1
SIGN	82.1±0.3	72.4±0.8	79.5±0.5	83.1±0.8	91.7±0.7	91.9±0.3	92.8±0.8
S^2GC	82.7±0.3	73.0±0.2	79.9±0.3	83.1±0.7	91.6±0.6	91.6±0.6	93.1±0.8
GBP	83.9±0.7	72.9±0.5	80.6±0.4	83.5±0.8	92.1±0.8	92.3±0.4	93.3±0.7
GAMLP(JK)	**84.3±0.8**	**74.6±0.4**	80.7±0.4	**84.5±0.7**	**92.8±0.7**	92.6±0.5	**93.6±1.0**
GAMLP(R)	83.9±0.6	73.9±0.6	**80.8±0.5**	84.2±0.5	92.6±0.8	**92.8±0.7**	93.2±1.0

Table 2: Performance comparison on the ogbn-products dataset.

Methods	Val Accuracy	Test Accuracy
GCN	92.00±0.03	75.64±0.21
SGC	92.13±0.02	75.87±0.14
GraphSAGE	92.24±0.07	78.50±0.14
GraphSAINT	92.52±0.13	80.27±0.26
SIGN	92.99±0.04	80.52±0.16
DeeperGCN	92.38±0.09	80.98±0.20
SAGN	93.09±0.04	81.20±0.07
UniMP	93.08±0.17	82.56±0.31
GAMLP(JK)	93.19±0.03	83.54±0.25
GAMLP(R)	93.11±0.05	**83.59±0.09**

Table 3: Performance comparison on the ogbn-papers100M dataset.

Methods	Val Accuracy	Test Accuracy
SGC	66.48±0.20	63.29±0.19
SIGN	69.32±0.06	65.68±0.06
SIGN-XL	69.84±0.06	66.06±0.19
SAGN	70.34±0.99	66.75±0.84
GAMLP(JK)	71.92±0.04	**68.07±0.10**
GAMLP(R)	71.21±0.03	67.46±0.02

- Both variants of GAMLP outperform all the baseline methods.
- On the two large OGB datasets, GAMLP takes the lead by 1.03% and 1.32% on the ogbn-products dataset and the ogbn-papers100M dataset, respectively.
- The contest between the two variants of GAMLP is not a one-horse race, which reveals that the two different attention mechanisms both have their irreplaceable sense in some ways.

Inductive Performance

Table 4: Performance comparison on three inductive datasets.

Methods	PPI	Flickr	Reddit
SGC	65.7±0.01	50.2±0.12	94.9±0.00
GraphSAGE	61.2±0.05	50.1±0.13	95.4±0.01
Cluster-GCN	99.2±0.04	48.1±0.05	95.7±0.00
GraphSAINT	99.4±0.03	51.1±0.10	96.6±0.01
GAMLP(JK)	**99.82±0.01**	**54.12±0.01**	**97.04±0.01**
GAMLP(R)	99.66±0.01	53.12±0.00	96.62±0.01

- The leading advantage of GAMLP(JK) over SOTA inductive method – GraphSAINT is more than 3.0% on the widely-used dataset – Flickr.
- GAMLP is powerful in predicting the properties of unseen nodes.

Ablation Study

Table 5: Ablation study on label utilization.

Methods	Val Accuracy	Test Accuracy
GAMLP(R)	93.11±0.05	**83.59±0.05**
-no_label	92.29±0.06	81.43±0.18
-plain_label	92.53±0.21	81.12±0.45
-uniform	92.72±0.15	81.28±0.93

Table 6: Ablation study on reference vector.

Methods	Val Accuracy	Test Accuracy
GAMLP(JK)	82.5±0.5	**80.7±0.4**
-origin_feature	82.2±0.4	80.5±0.4
-normal_noise	81.8±0.4	79.8±0.5
-no_reference	81.5±0.5	79.9±0.3

The predictive accuracy of GAMLP(R) is evaluated on the ogbn-products dataset along with its three variants: "-no_label", "-plain_label", and "-uniform", which stands for not using labels, removing last residual connections, and replacing last residual connections with uniform distributions, respectively.

- Utilizing labels brings huge performance gain to GAMLP: from 81.43% to 83.59%.
- The performance drop from removing the last residual connections is significant since directly adopting the raw training labels leads to the label leakage problem.
- The fact that "-uniform" performs worse than "-no_label" illustrates that intuitively fusing the original label distribution with the uniform distribution would harm the predictive accuracy.

Ablation Study

Table 5: Ablation study on label utilization.

Methods	Val Accuracy	Test Accuracy
GAMLP(R)	93.11±0.05	**83.59±0.05**
-no_label	92.29±0.06	81.43±0.18
-plain_label	92.53±0.21	81.12±0.45
-uniform	92.72±0.15	81.28±0.93

Table 6: Ablation study on reference vector.

Methods	Val Accuracy	Test Accuracy
GAMLP(JK)	82.5±0.5	**80.7±0.4**
-origin_feature	82.2±0.4	80.5±0.4
-normal_noise	81.8±0.4	79.8±0.5
-no_reference	81.5±0.5	79.9±0.3

We evaluate the three variants of GAMLP(JK): "-origin_feature", "-normal_noise", and "-no_reference", which changes the reference vector to the original node feature, noise from the normal distribution, and nothing, respectively.

- The superiority of the concatenated features from different propagation steps comes from the fact that it allows the model to capture the interactions between the propagated features over the different receptive fields.

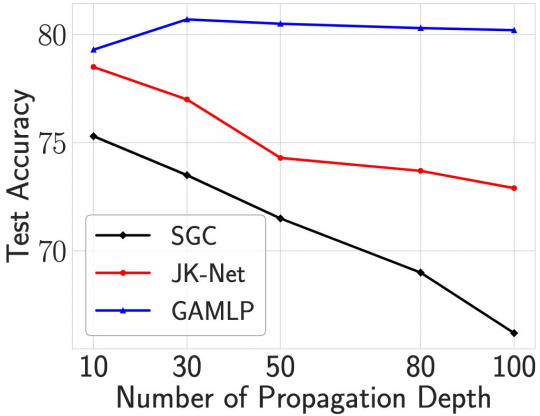
Efficiency Comparison

Table 7: Efficiency comparison on the ogbn-products dataset.

Methods	SGC	SIGN	GAMLP(JK)	GAMLP(R)	GraphSAINT	Cluster-GCN
Training time	1.0	4.0	8.0	9.3	364	503
Test accuracy	75.87	80.52	83.54	**83.59**	79.08	78.97

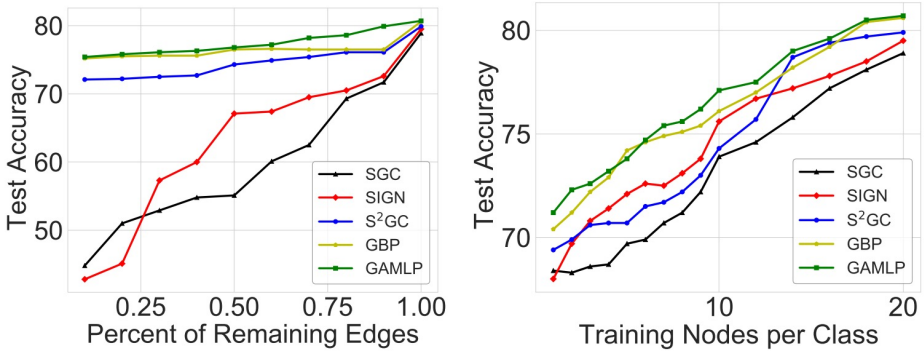
- Sampling-based methods (e.g., GraphSAINT) consume much more time than the simplified GNNs (i.e., SGC and SIGN).
- The two variants of GAMLP achieve the best predictive accuracy while requiring comparable training time with SGC.

Experiments on propagation step



- Even at propagation depth equals 100, the predictive accuracy of GAMLP(JK) still exceeds 80.0%, higher than the predictive accuracy of most baselines.

Experiments on Sparse Graph



(a) Edge Sparsity (b) Label Sparsity

- GAMLP consistently outperforms all the baselines when faced with different levels of edge and label sparsity.
- The node-wise propagation enables GAMLP to better capture long-range dependencies, which is crucial when applying GNN methods to highly sparse graphs.

06 Conclusion

Conclusion

Advantages:

1. Deep:

GAMLP is able to go extremely deep without over-smoothing;

2. Scalable:

GAMLP is easy for distributed learning (no feature pulling communication);

3. Efficient:

GAMLP is simple and acquires less computation cost.

Applications:

1. Sparse Graphs:

GAMLP can help to propagate the graph information of sparse graph by increasing the propagation steps;

2. Large Graphs

GAMLP is scalable in distributed settings for extremely large graph in industrial production environment.

Reference

1. **Zhang W**, Yang M, Sheng Z, et al. *Node Dependent Local Smoothing for Scalable Graph Learning*[C]. NeurIPS, 2021
2. **Zhang W**, Sheng Z, Jiang Y, et al. *Evaluating deep graph neural networks*. arXiv, 2021.
3. **Zhang W**, Shen Y, Lin Z, et al. *GMLP: Building Scalable and Flexible Graph Neural Networks with Feature-Message Passing*. arXiv, 2021.
4. **Zhang W**, Yin Z, Sheng Z, et al. *Graph Attention Multi-Layer Perceptron*. arXiv, 2021.
5. Chen M, Wei Z, Ding B, et al. *Scalable graph neural networks via bidirectional propagation*[J]. NeurIPS, 2020.
6. Wu F, Souza A, Zhang T, et al. *Simplifying graph convolutional networks*[C]. ICML 2018.
7. Liu M, Gao H, Ji S. *Towards deeper graph neural networks*[C]. KDD 2020.
8. Chiang W L, Liu X, Si S, et al. *Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks*[C]. KDD 2019.
9. Huang Q, He H, Singh A, et al. *Combining label propagation and simple models out-performs graph neural networks*[J]. ICLR 2021.
10. Klicpera J, Bojchevski A, Günnemann S. *Predict then propagate: Graph neural networks meet personalized pagerank*[J]. ICLR 2019.
11. Li Q, Han Z, Wu X M. *Deeper insights into graph convolutional networks for semi-supervised learning*[C]//AAAI, 2018.
12. Shi Y, Huang Z, Wang W, et al. *Masked label prediction: Unified message passing model for semi-supervised classification*[J]. IJCAI, 2021.
13. Xu K, Li C, Tian Y, et al. *Representation learning on graphs with jumping knowledge networks*[C], ICML 2018.
14. Zeng H, Zhou H, Srivastava A, et al. *Graphsaint: Graph sampling based inductive learning method*[J]. ICLR 2020.
15. Zhu H, Koniusz P. *Simple spectral graph convolution*[C]. ICLR, 2021.
16. Sun C, Wu G. *Scalable and Adaptive Graph Neural Networks with Self-Label-Enhanced training*[J]. arXiv, 2021.
17. Bojchevski A, Klicpera J, Perozzi B, et al. *Scaling graph neural networks with approximate pagerank*[C]. KDD 2020.

Logo | DataFunSummit

THANKS!

Ending