Graph Attention Multi-Layer Perceptron

Wentao Zhang[1], **Ziqi Yin**[3], **Zeang Sheng**[1], **Wen Ouyang**[2], **Xiaosen Li**[2]
Yangyu Tao[2], **Zhi Yang**[1], **Bin Cui**[1]
[1]EECS, Peking University [2]Tencent Inc. [3]Beijing Institute of Technology

Abstract

Graph neural networks (GNNs) have recently achieved state-of-the-art performance in many graph-based applications. Despite the high performance, they typically need to perform expensive recursive neighborhood expansions during every training epoch and has the scalability issue. Moreover, most of them are inflexible since they are restricted to fixed-hop neighborhoods and insensitive to actual receptive field requirements for each node. We circumvent these limitations by introducing a scalable and flexible method: Graph Attention Multi-Layer Perceptron (GAMLP). Following the routine of decoupled GNNs, the feature propagation in GAMLP is executed during pre-computation, which helps it maintain high scalability. With three proposed receptive field attention, each node in GAMLP is flexible in leveraging the propagated features over the different sizes of reception field. We conduct extensive evaluations on two large open graph benchmarks (ogbn-products and ogbn-papers100M), demonstrating that GAMLP not only achieves state-of-the-art performance, but also enjoys high scalability and efficiency.

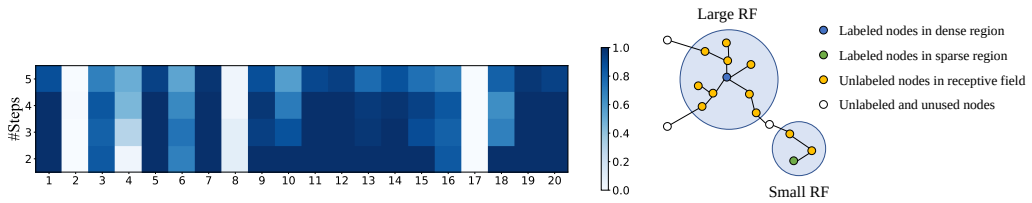
1 Introduction

Graph Neural Networks (GNNs) are powerful deep neural networks for graph-structured data, becoming the *de facto* method in many semi-supervised and unsupervised graph representation learning scenarios such as node classification, link prediction, recommendation, and knowledge graphs [15, 10, 1, 6, 7, 22]. Through stacking K graph convolution layers, GNNs can learn node representations by utilizing information from the K-hop neighborhood and thus enhance the model performance by getting more unlabeled nodes involved in the training process.

Unlike the images, text, or tabular data, where training data are independently distributed, graph data contains extra relationship information between nodes. Besides, the real-world graph is usually huge. For example, the users and their relationships in Wechat can be formed as a graph, and this graph has billions of nodes and ten billion edges. Every node in a K-layer GNN will incorporate a set of nodes, including the node itself and its K-hop neighbors. This set is called Receptive Field(RF). As the size of RF grows exponentially to the number of GNN layers, the rapidly expanding RF introduces high computation and memory cost in a single machine. Besides, even in a distributed environment, GNN has to read great amount of data of neighboring nodes to compute the single target node representation, leading to high communication cost. Despite their effectiveness, the utilization of neighborhood information in GNNs leads to the scalability issue for training on large graphs.

A commonly used approach to tackle this issue is sampling, such as node sampling [10, 3], layer sampling [14, 2] and graph sampling [5, 29]. However, the sampling-based methods are imperfect because they still face high communication costs, and the sampling quality highly influences the model performance. Besides, a recent direction for scalable GNNs is based on model simplification. For example, Simplified GCN (SGC) [23] decouples the feature propagation and the non-linear

Under improvement.
GAMLP is adapted from our previous work GMLP: https://arxiv.org/abs/2104.09880
Codes are avaliable at https://github.com/PKU-DAIR/GAMLP



(a) Inconsistent optimal steps (b) Inconsistent RF expansion speed

Figure 1: (Left) Test accuracy of SGC on 20 randomly sampled nodes of Citeseer. The X-axis is the node id, and Y-axis is the propagation steps (layers). The color from white to blue represents the ratio of being predicted correctly in 50 different runs. (Right) The local graph structures for two nodes in different regions; the node in the dense region has larger RF within two iterations of propagation.

transformation process, and the former is executed during pre-processing. Unlike the sampling-based methods, which still need feature propagation in each training epoch, this time-consuming process in SGC is only executed once, and only the nodes of training set get involved in the model training. As a result, SGC is computation and memory efficient in a single machine and scalable in distributed settings since it does not require fetch features of neighboring nodes in the model training process. Despite the scalability, SGC adopts fixed layers of feature propagation, leading to a fixed RF of all nodes. Such graph-wise propagation lacks the flexibility to model the interesting correlations on node features under different reception fields. This either makes that long-range dependencies cannot be fully leveraged due to the undersized RF, or loses local information due to introducing over-smoothed noise with the oversized RF. Both results in non-optimal discriminative node representations.

Lines of simplified models have been proposed to better use propagated features under different propagation layers and RF. As graph-wise propagation only considers features under a fixed layer, SIGN [9] proposes to concatenate all these features without information loss, while S^2GC [32] averages all these features to generate the combined feature with the same dimension. Although they have considered the influence of different layers, the importance of different propagated features is ignored. Under a large propagation layer, some over-smoothed features with oversized RF will introduce feature noise and degrade the model performance. GBP [4] tackles this issue by adopting a constant decay factor for weighted average in the propagated features. Motivated by Personalized PageRank, the propagated features with larger propagation layers face a higher risk of over-smoothing, and they will contribute less to the final averaged features in GBP. All these methods adopt a layer-wise propagation mechanism and consider the features after different layers of propagation. Despite their effectiveness, they fail to consider the feature combination from a node-wise level.

As shown in Figure 1(a), different nodes require different propagation steps and corresponding smoothness levels. Besides, the homogeneous and non-adaptive feature averaging may be unsuitable for all nodes due to the inconsistent RF expansion speed shown in Figure 1(b). To support scalable and node-adaptive graph learning, we propose a novel MLP with three RF attention, abbreviated as GAMLP. Experimental results demonstrate that GAMLP achieves the state-of-the-art performance on the three largest ogbn datasets, while maintains high scalability and efficiency.

2 Preliminaries

In this section, we introduce the notations and review some current works tackling GNN scalability.

Notations. We consider an undirected graph $\mathcal{G} = (\mathcal{V},\mathcal{E})$ with $|\mathcal{V}| = N$ nodes and $|\mathcal{E}| = M$ edges. We denote by $\mathbf{A}$ the adjacency matrix of $\mathcal{G}$, weighted or not. Nodes can possibly have features vector of size d, stacked up in an $N \times d$ matrix $\mathbf{X}$. $\mathbf{D} = \mathrm{diag}\,(d_1, d_2, \cdots, d_N) \in \mathbb{R}^{N\times N}$ denotes the degree matrix of $\mathbf{A}$, where $d_i = \sum_{v_j\in\mathcal{V}} \mathbf{A}_{ij}$ is the degree of node v_i. Suppose $\mathcal{V}_l$ is the labeled set, and our goal is to predict the labels for nodes in the unlabeled set $\mathcal{V}_u$ with the supervision of $\mathcal{V}_l$.

Sampling. A commonly used method to tackle the scalability issue (i.e., the recursive neighborhood expansion) in GNN is sampling. As a node-wise sampling method, GraphSAGE [10] randomly samples a fixed size set of neighbors for computing in each mini-batch. VR-GCN [3] analyzes the

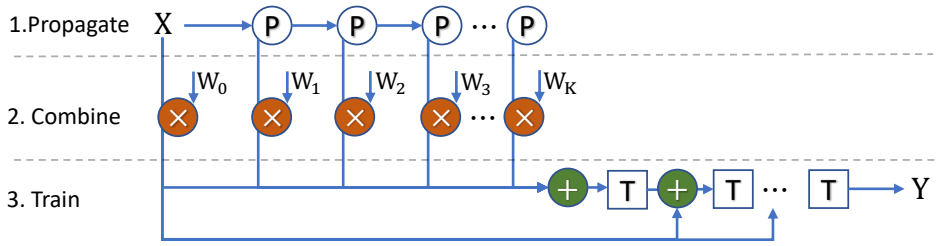


Figure 2: Overview of the proposed GAMLP, including (1) feature propagation, (2) feature combination with RF attention, and (3) MLP training. The feature propagation can be pre-processed.

variance reduction so that it can reduce the size of samples with an additional memory cost. For the layer-wise sampling, Fast-GCN [2] samples a fixed number of nodes at each layer, and ASGCN [14] proposes the adaptive layer-wise sampling with better variance control. In the graph level, Cluster-GCN [5] firstly clusters the nodes and then samples the nodes in the clusters, and GraphSAINT [29] directly samples a subgraph for mini-batch training. As an orthogonal way to model simplification, sampling has already been widely used in many GNNs and GNN systems [31, 33, 8].

Graph-wise Propagation. Recently studies have observed that non-linear feature transformation contributes little to the performance of the GNNs as compared to feature propagation. Thus, a new direction recently emerging for scalable GNN is based on the *simplified* GCN (SGC) [23], which successively removes nonlinearities and collapsing weight matrices between consecutive layers. This reduces GNNs into a linear model operating on K-layers propagated features:

$$\mathbf{X}^{(K)} = \hat{\mathbf{A}}^K \mathbf{X}^{(0)}, \qquad \mathbf{Y} = \text{softmax}(\mathbf{\Theta} \mathbf{X}^{(K)}), \tag{1}$$

where $\mathbf{X}^{(0)} = \mathbf{X}$, $\mathbf{X}^{(K)}$ is the K-layers propagated feature, and $\hat{\mathbf{A}} = \widetilde{\mathbf{D}}^{r-1} \widetilde{\mathbf{A}} \widetilde{\mathbf{D}}^{-r}$. By setting $r =$ 0.5, 1 and 0, $\hat{\mathbf{A}}$ represents the symmetric normalization adjacency matrix $\widetilde{\mathbf{D}}^{-1/2} \widetilde{\mathbf{A}} \widetilde{\mathbf{D}}^{-1/2}$ [16], the transition probability matrix $\widetilde{\mathbf{A}} \widetilde{\mathbf{D}}^{-1}$ [29], or the reverse transition probability matrix $\widetilde{\mathbf{D}}^{-1} \widetilde{\mathbf{A}}$ [25], respectively. As the propagated features $\mathbf{X}^{(K)}$ can be precomputed, SGC is more scalable and efficient for the large graph. However, such graph-wise propagation restricts the same propagation steps and a fixed RF for each node. Therefore, some nodes' features may be over-smoothed or under-smoothed due to the inconsistent RF expansion speed, leading to non-optimal performance.

Layer-wise Propagation. Following SGC, some recent methods adopt layer-wise propagation to combine the features with different propagation layers. SIGN [9] proposes to concatenate the different iterations of propagated features with linear transformation: $[\mathbf{X}^{(0)}\mathbf{W}_0, \mathbf{X}^{(1)}\mathbf{W}_1, ..., \mathbf{X}^{(K)}\mathbf{W}_K]$. S^2GC [32] proposes the simple spectral graph convolution to average the propagated features in different iterations as $\mathbf{X}^{(K)} = \sum_{l=0}^{K} \hat{\mathbf{A}}^l \mathbf{X}^{(0)}$. In addition, GBP [4] further improves the combination process by weighted averaging as $\mathbf{X}^{(K)} = \sum_{l=0}^{K} w_l \hat{\mathbf{A}}^l \mathbf{X}^{(0)}$ with the layer weight $w_l = \beta(1-\beta)^l$. Similar to these works, we also use a linear model for higher training scalability. The difference lies in that we consider the propagation from a node-wise perspective and each node in GAMLP has a personalized combination of different steps of the propagated features.

3 The GAMLP Model

3.1 Overview

As shown in Figure 2, GAMLP decomposes the end-to-end GNN training into three parts: feature propagation, feature combination with RF attention, and the MLP training. As the feature propagation is pre-processed only once, and MLP training is efficient and salable, we can easily scale GAMLP to large graphs. Besides, with the RF attention, each node in GAMLP can adaptively get the suitable combination weights for propagated features under different RF, thus boosting model performance.

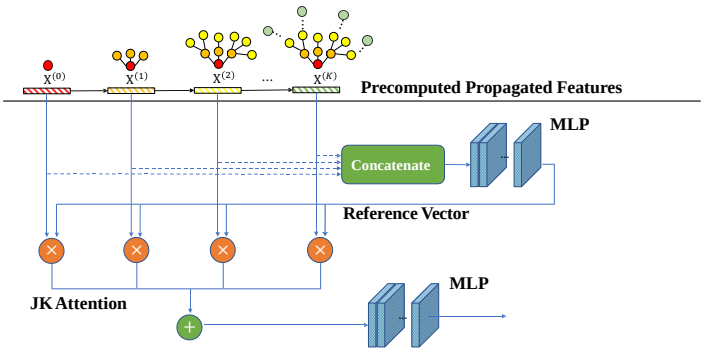


Figure 3: The architecture of GAMLP with JK Attention.

3.2 Establishment of GAMLP

3.2.1 Feature Propagation

We separate the essential operation of GNNs — feature propagation by removing the neural network $\mathbf{\Theta}$ and nonlinear activation δ for feature transformation. Specifically, we construct a parameter-free K-step feature propagation as:

$$\mathbf{X}^{(l)} \leftarrow \mathbf{T}\mathbf{X}^{(l-1)}, \ \forall l = 1, \ldots, K, \tag{2}$$

where $\mathbf{X}^{(l)}$ contains the features of a fixed RF: the node itself and its l-hop neighborhoods.

After K-step feature propagation shown in E.q. 9, we correspondingly get a list of propagated features under different propagation steps: $[\mathbf{X}^{(0)}, \mathbf{X}^{(1)}, \mathbf{X}^{(k)}, ..., \mathbf{X}^{(K)}]$. For a node-wise propagation, we propose to average these propagated features in a weighted manner:

$$\mathbf{H} = \sum_{l=0}^{K} \mathbf{W}_l \mathbf{X}^{(l)}, \tag{3}$$

where $\mathbf{W}_l = Diag(\eta_l)$ is the diagonal matrix derived from vector η_l, and η_l is an N-dimension vector derived from vector $\eta_l = w_i(l), 1 \leq i \leq N$, and $w_i(l)$ measures the importance of the l-step propagated feature for node v_i. To satisfy different RF requirements for each node, we introduce three RF attention mechanisms to get $w_i(l)$.

3.2.2 Receptive Field Attention

Smoothing Attention. Suppose we execute feature propagation for infinite times. In that case, the node embedding within the same connected component will reach a stationary state, and it is hard to distinguish a specific node from others. This issue is referred to as over-smoothing [18]. Concretely, when applying $\widetilde{\mathbf{D}}^{r-1}\widetilde{\mathbf{A}}\widetilde{\mathbf{D}}^{-r}$ as adjacency matrix $\hat{\mathbf{A}}$, the stationary state follows

$$\hat{\mathbf{A}}^{\infty}_{i,j} = \frac{(d_i + 1)^r (d_j + 1)^{1-r}}{2m + n}, \tag{4}$$

To avoid the over-smoothing issue introduced by large RF, the weight $w_i(k)$ parameterized by node i and aggregation step k is defined as:

$$\widetilde{\mathbf{X}}^{(l)}_i = \mathbf{X}^{(l)}_i \parallel \mathbf{X}^{(\infty)}_i, \quad \widetilde{w}_i(l) = \delta(\widetilde{\mathbf{X}}^{(l)}_i \cdot s), \quad w_i(l) = e^{\widetilde{w}_i(l)} / \sum_{k=0}^{K} e^{\widetilde{w}_i(k)}, \tag{5}$$

where $\parallel$ stands for concatenation, and $s \in \mathbb{R}^{1 \times d}$ is a trainable vector. Larger $w_i(l)$ means that the l-step propagated feature of node v_i is more distant from the stationary state, and has less risk of being noise. Therefore, propagated feature with larger $w_i(k)$ should contribute more to the feature combination.

Recursive Attention At each propagation step l, we recursively measure the feature information gain compared with the previous combined feature as:

$$\widetilde{\mathbf{X}}_i^{(l)} = \mathbf{X}_i^{(l)} \parallel \sum_{k=0}^{l-1} w_i(k)\mathbf{X}_i^{(k)}, \quad \widetilde{w}_i(l) = \delta(\widetilde{\mathbf{X}}_i^{(l)} \cdot s), \quad w_i(l) = e^{\widetilde{w}_i(l)} / \sum_{k=0}^{K} e^{\widetilde{w}_i(k)}, \tag{6}$$

As $\widetilde{\mathbf{X}}_i^{(l-1}$ combines the graph information under different propagation steps and RF, large proportion of the information in $\widetilde{\mathbf{X}}_i^{(l)}$ may have already existed in $\sum_{k=0}^{l-1} w_i(k)\mathbf{X}_i^{(k)}$, leading to small information gain . larger $w_i(l)$ means the feature $\mathbf{X}_i^{(l)}$ is more important to the current state of node v_i since combining $\widetilde{\mathbf{X}}_i^{(l)}$ will introduce higher information gain.

JK Attention Jumping Knowledge Network (JK-Net) [25] adopts layer aggregation to combine the node embeddings of different GCN layers, and thus it can leverage the propagated nodes information with different RF. Motivated by JK-Net, we propose to guide the feature combination process with the model prediction trained on all the propagated features. Figure 3 shows the corresponding model architecture of GAMLP with JK attention, which includes two branches: the concatenated JK branch and the attention-based combination branch. We define the MLP prediction of the JK branch as $\mathbf{E}_i = \text{MLP}(\mathbf{X}_i^{(1)} \parallel \mathbf{X}_i^{(2)} \parallel ... \parallel \mathbf{X}_i^{(K)})$, and then define the combination weight as:

$$\widetilde{\mathbf{X}}_i^{(l)} = \mathbf{X}_i^{(l)} \parallel \mathbf{E}_i, \quad \widetilde{w}_i(l) = \delta(\widetilde{\mathbf{X}}_i^{(l)} \cdot s), \quad w_i(l) = e^{\widetilde{w}_i(l)} / \sum_{k=0}^{K} e^{\widetilde{w}_i(k)}, \tag{7}$$

The JK branch aims to create a multi-scale feature representation for each node, which helps the attention mechanism learn the weight $w_i(k)$. The learned weights are then fed into the attention-based combination branch to generate each node's refined attention feature representation. As the training process continues, the attention-based combination branch will gradually emphasize those neighborhood regions that are more helpful to the target nodes. The JK attention can model a wider neighborhood while enhancing correlations, bringing a better feature representation for each node.

3.2.3 Incorporating Label Propagation.

To reinforce the model performance, we propose a simple and scalable way to take advantage of the node labels of the training set. First, the label embedding matrix $\hat{\mathbf{Y}}^{(0)}$ is initialized as all zero. Then, we use the hard training labels, $\mathbf{Y}_{\mathcal{V}_l}$, to fill in the all zero matrix and propagate it with the normalized adjacency matrix $\hat{\mathbf{A}}$:

$$\hat{\mathbf{Y}}_{\mathcal{V}_l}^{(0)} = \mathbf{Y}_{\mathcal{V}_l}, \hat{\mathbf{Y}}^{(k+1)} = \hat{\mathbf{A}}\hat{\mathbf{Y}}^{(k)}, \tag{8}$$

where $\mathcal{V}_l$ is the labeled node set. After K-steps Label Propagation, we get the final label embedding $\hat{\mathbf{Y}}^{(K)}$ and then use it to enhance the model prediction.

3.2.4 Model Training

Previous work [30] shows that the main limitations of deep GNNs are the *over-smoothing* introduced by large steps of propagation and *model degradation* introduced by too many layers of non-linear transformation. The proposed attention-based feature propagation can adaptively leverage the propagated features over the different sizes of reception field and avoid the *over-smoothing* issue.

Large graph [30] require large steps of non-liner transformation. To tackle the *model degradation* problem, we propose to use initial residual as:

$$\hat{\mathbf{H}}^{(l)} \leftarrow \delta(\mathbf{W}^{(l)}\hat{\mathbf{H}}^{(l-1)} + \mathbf{X}^{(0)}), \; l = 1, \ldots, L, \tag{9}$$

where L is the MLP layers, and $\mathbf{H}^{(0)} = \hat{\mathbf{H}}$ is the combined feature matrix.

Then the output of this L layers MLP is added with the label embedding from Sec. 3.2.3 to get the final output embedding:

$$\widetilde{\mathbf{H}} = \hat{\mathbf{H}}^{(L)} + \text{MLP}(\hat{\mathbf{Y}}^{(K)}). \tag{10}$$

The MLP here is used to map the embedding $\hat{\mathbf{Y}}^{(K)}$ to the same space as $\hat{\mathbf{H}}^{(L)}$.

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

$$\mathcal{L}_{CE} = -\sum_{i \in \mathcal{V}_l} \sum_j \mathbf{Y}_{ij} \log(\text{softmax}(\widetilde{\mathbf{H}})_{ij}) \tag{11}$$

where $\mathbf{Y}_i$ is the one-hot label indicator vector.

3.2.5 Reliable Label Utilization (RLU)

Reliable Label Propagation. To better utilize the predicted soft label (i.e., softmax outputs), we split the whole training process into multiple stages, each containing a full training procedure of the GAMLP model. At the first stage, the GAMLP model is trained according to the above-mentioned procedure. However, at later stages, we take advantage of the predicted reliable soft label of the last stage to improve the label embedding $\hat{\mathbf{Y}}^{(0)}$. Here, we denote the prediction results of m-th stage as $\mathbf{P}^{(m)}$:

$$\mathbf{P}^{(m)} = \text{softmax}(\hat{\mathbf{H}}/T), \quad T \in (0, 1], \tag{12}$$

where the parameter T controls the softness of the softmax distribution. Lower values of T leads to more hardened distribution.

Suppose we are now at the beginning of the m-th stage ($m > 1$) of the whole training process. Rather than just using the training labels to construct the initial label embedding $\hat{\mathbf{Y}}^{(0)}$, we adopt the predicted results for the nodes in the validation set and the test set at the last stage as well. To ensure the reliability of the predicted soft label, we use a threshold ϵ to filter out the low-confident nodes in the validation set and the test set. The formulation for the reliable label is as follows:

$$\hat{\mathbf{Y}}_i^{(0)} = \begin{cases} \mathbf{Y}_i, \text{ if } i \in \mathcal{V}_l, \\ \mathbf{P}_i^{(m-1)} \text{ if } i \in \mathcal{V}_r, \\ \mathbf{0}, \text{ otherwise.} \end{cases} \quad \hat{\mathbf{Y}}^{(k+1)} = \hat{\mathbf{A}}\hat{\mathbf{Y}}^{(k)}. \tag{13}$$

In the formulation, the reliable node set $\mathcal{V}_r$ is composed of nodes whose predicted probability belonging to the most likely class at $(m-1)$-th stage is greater than the threshold ϵ.

Reliable Label Distillation. To fully take advantage of the helpful information of the last stage, we also included a knowledge distillation module in our model. Again to guarantee the reliability of the knowledge distillation module, we only include the nodes in the reliable node set $\mathcal{V}_r$ at m-th stage ($m > 1$) and then define the weighted KL divergence as:

$$\mathcal{L}_{KD} = \sum_{i \in \mathcal{V}_r} \sum_j \alpha_i \mathbf{P}_{ij}^{(m-1)} \log \frac{\mathbf{P}_{ij}^{(m-1)}}{\mathbf{P}_{ij}^{(m)}}, \tag{14}$$

where α_i for reliable node i is its predicted probability belonging to the mostly likely class at $(m-1)$-th stage. We further incorporate α_i here to better guide the distillation process, assigning higher weights to more confident nodes.

The complete training loss for m-th stage ($m > 1$) is defined as:

$$\mathcal{L} = \mathcal{L}_{CE} + \gamma \mathcal{L}_{KD}, \tag{15}$$

where γ is a hyperparameter, balancing the importance of the knowledge distillation module.

3.3 Relation with current methods

GAMLP vs. GBP. Both GAMLP and GBP propose to weighted average the propagated features under different propagation steps and RF. However, GBP adopts a layer-wise propagation and ignores the inconsistent RF expansion speed for different nodes. As the optimal propagation steps and smoothing levels of different nodes are different, some nodes may face the over-smoothing or under-smoothing issue even propagated the same step. GAMLP considers the feature propagation in a more fine-grained node perspective. Compared with GBP, SGC, and S^2GC, the limitation of GAMLP is that all the propagated features are required in the model training, leading to high memory cost.

GAMLP vs. GAT. Each node in a GAT layer learns to weighted combine the embedding (or feature) of its neighborhoods with an attention mechanism, and the attention weights are measured by the

Table 1: Overview of datasets.

Dataset	#Nodes	#Features	#Edges	#Classes	#Train/Val/Test
ogbn-products	2,449,029	100	61,859,140	47	196K/49K/2,204K
ogbn-papers100M	111,059,956	128	1,615,685,872	172	1,207K/125K/214K

local information in a fixed RF – the node itself and its direct neighbors. Different from the attention mechanism in GAT, GAMLP considers more global information under different RF.

GAMLP vs. DAGNN. DAGNN can adaptively learn the combination weights via the gating mechanism and thus assign proper weights for different nodes. However, the gating mechanism in DAGNN is correlated with the parameterized node embedding rather than the training-free node feature used in GAMLP, leading to low scalability and efficiency.

GAMLP vs. JK-Net. Motivated by JK-Net, GAMLP with JK attention concatenate the propagated features under different propagation steps. However, the model prediction based on the concatenated feature is just used as a reference vector for the attention-based combination branch in GAMLP rather than the final results. Compared with JK-Net, GAMLP with JK attention is more effective in alleviating the over-smoothing and scalability issue that deep architecture introduces.

4 Experiments

In this section, we verify the effectiveness of GAMLP on seven graph datasets. We aim to answer the following two questions. **Q1:** Compared with current methods, can GAMLP achieve higher predictive accuracy? **Q2:** Why GAMLP is effective?

4.1 Experimental Setup

Datasets and baselines. We conduct the experiments on the ogbn-products and ogbn-papers100M datasets in [11]. The dataset statistics are shown in Table 1. For the comparison on the ogbn-products dataset, we choose the following baseline methods: GCN [15], GraphSAGE [10], SIGN [9], DeeperGCN [17], SAGN and SAGN+SLE [21], UniMP [20], and MLP+C&S [13]. For the comparison on the ogbn-papers100M dataset, we choose the following baseline methods: SGC [23], SIGN and SIGN-XL [9], SAGN and SAGN+SLE [21].

Implementations.

To alleviate the influence of randomness, we repeat each method ten times and report the mean performance and the standard deviations. The experiments are conducted on a machine with Intel(R) Xeon(R) Platinum 8255C CPU@2.50GHz, and a single Tesla V100 GPU with 32GB GPU memory. The operating system of the machine is Ubuntu 16.04. As for software versions, we use Python 3.6, Pytorch 1.7.1, and CUDA 10.1. The hyper-parameters in each baseline are set according to the original paper if available. Please refer to Appendix B for the detailed hyperparameter settings for our GAMLP+RLU.

4.2 Experimental Results.

End-to-end comparison. To answer **Q1**, Table 2, 3 show the test accuracy of our GAMLP+RLU and all the baseline methods. The experimental results show that GAMLP+RLU achieves state-of-the-art performance, exceeding the strongest baseline SAGN+SLE by 0.17% and 0.25% on ogbn-products and ogbn-papers100M datasets, respectively.

Interpretability. GAMLP can adaptively and effectively combine multi-scale propagated features for each node. To demonstrate this, Figure 4 shows the average attention weights of propagated features according to the number of steps and degrees of input nodes, where the maximum step is 6. In this experiment, we randomly select 20 nodes for each degree range (1-4, 5-8, 9-12) and plot the relative weight based on the maximum value. We get two observations from the heat map: 1) The 1-step and 2-step propagated features are always of great importance, which shows that GAMLP captures

Table 2: Test accuracy on ogbn-products dataset.

Methods	Validation Accuracy	Test Accuracy
GCN	92.00±0.03	75.64±0.21
GraphSAGE	92.24±0.07	78.50±0.14
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
MLP+C&S	91.47±0.09	84.18±0.07
SAGN+SLE	92.87±0.03	84.28±0.14
GAMLP+RLU	93.24±0.05	**84.59±0.10**

Table 3: Test accuracy on ogbn-papers100M dataset.

Methods	Validation 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
SAGN+SLE	71.31±0.10	68.00±0.15
GAMLP+RLU	71.59±0.05	**68.25±0.11**

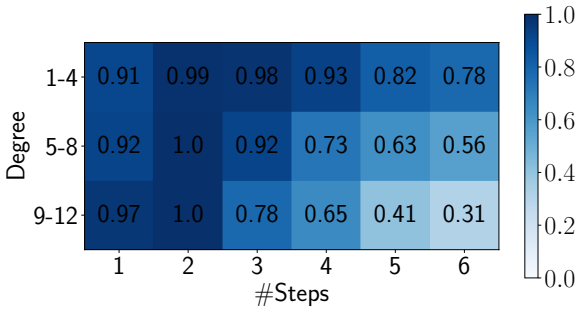


Figure 4: The average attention weights of propagated features of different steps on 60 randomly selected nodes from ogbn-products.

the local information as those widely 2-layer methods do; 2) The weights of propagated features with larger steps drop faster as the degree grows, which indicates that our attention mechanism could prevent high-degree nodes from including excessive irrelevant nodes which lead to over-smoothing. From the two observations, we conclude that GAMLP is able to identify the different RF demands of nodes and explicitly weight each propagated feature.

5 Conclusion

This paper presents Graph Attention Multilayer Perceptron (GAMLP), a scalable, efficient, and powerful graph learning method based on the reception field attention. Concretely, GAMLP defines three principled attention mechanisms, i.e., smoothing attention, recursive attention, and JK attention, and each node in GAMLP can leverage the propagated features over different size of RF in a node-specific way. Extensive experiments on three large ogbn graphs verify the effectiveness of the proposed method.

References

[1] D. Bo, X. Wang, C. Shi, M. Zhu, E. Lu, and P. Cui. Structural deep clustering network. In *Proceedings of The Web Conference 2020*, pages 1400–1410, 2020.

[2] J. Chen, T. Ma, and C. Xiao. Fastgcn: Fast learning with graph convolutional networks via importance sampling. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, 2018.

[3] J. Chen, J. Zhu, and L. Song. Stochastic training of graph convolutional networks with variance reduction. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pages 941–949, 2018.

[4] M. Chen, Z. Wei, B. Ding, Y. Li, Y. Yuan, X. Du, and J.-R. Wen. Scalable graph neural networks via bidirectional propagation. *arXiv preprint arXiv:2010.15421*, 2020.

[5] W.-L. Chiang, X. Liu, S. Si, Y. Li, S. Bengio, and C.-J. Hsieh. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *SIGKDD*, pages 257–266, 2019.

[6] G. Cui, J. Zhou, C. Yang, and Z. Liu. Adaptive graph encoder for attributed graph embedding. In *SIGKDD*, pages 976–985, 2020.

[7] W. Fan, Y. Ma, Q. Li, Y. He, E. Zhao, J. Tang, and D. Yin. Graph neural networks for social recommendation. In *The World Wide Web Conference*, pages 417–426, 2019.

[8] M. Fey and J. E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR 2019 Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[9] F. Frasca, E. Rossi, D. Eynard, B. Chamberlain, M. Bronstein, and F. Monti. Sign: Scalable inception graph neural networks. *arXiv preprint arXiv:2004.11198*, 2020.

[10] W. L. Hamilton, R. Ying, and J. Leskovec. Inductive representation learning on large graphs. In *NeurIPS*, pages 1025–1035, 2017.

[11] W. Hu, M. Fey, H. Ren, M. Nakata, Y. Dong, and J. Leskovec. Ogb-lsc: A large-scale challenge for machine learning on graphs. *arXiv preprint arXiv:2103.09430*, 2021.

[12] Z. Hu, Y. Dong, K. Wang, and Y. Sun. Heterogeneous graph transformer. In *Proceedings of The Web Conference 2020*, pages 2704–2710, 2020.

[13] Q. Huang, H. He, A. Singh, S.-N. Lim, and A. R. Benson. Combining label propagation and simple models out-performs graph neural networks. *arXiv preprint arXiv:2010.13993*, 2020.

[14] W. Huang, T. Zhang, Y. Rong, and J. Huang. Adaptive sampling towards fast graph representation learning. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 4563–4572, 2018.

[15] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[16] J. Klicpera, A. Bojchevski, and S. Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019.

[17] G. Li, C. Xiong, A. Thabet, and B. Ghanem. Deepergcn: All you need to train deeper gcns. *arXiv preprint arXiv:2006.07739*, 2020.

[18] Q. Li, Z. Han, and X.-M. Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[19] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. Van Den Berg, I. Titov, and M. Welling. Modeling relational data with graph convolutional networks. In *European semantic web conference*, pages 593–607. Springer, 2018.

[20] Y. Shi, Z. Huang, W. Wang, H. Zhong, S. Feng, and Y. Sun. Masked label prediction: Unified message passing model for semi-supervised classification. *arXiv preprint arXiv:2009.03509*, 2020.

[21] C. Sun and G. Wu. Scalable and adaptive graph neural networks with self-label-enhanced training. *arXiv preprint arXiv:2104.09376*, 2021.

[22] T. Trouillon, C. R. Dance, J. Welbl, S. Riedel, É. Gaussier, and G. Bouchard. Knowledge graph completion via complex tensor factorization. *arXiv preprint arXiv:1702.06879*, 2017.

[23] F. Wu, T. Zhang, A. H. d. Souza Jr, C. Fifty, T. Yu, and K. Q. Weinberger. Simplifying graph convolutional networks. *arXiv preprint arXiv:1902.07153*, 2019.

[24] X. Wu, M. Jiang, and G. Liu. R-gsn: The relation-based graph similar network for heterogeneous graph. *arXiv preprint arXiv:2103.07877*, 2021.

[25] K. Xu, C. Li, Y. Tian, T. Sonobe, K.-i. Kawarabayashi, and S. Jegelka. Representation learning on graphs with jumping knowledge networks. In *ICML*, pages 5453–5462. PMLR, 2018.

[26] L. Yu, J. Shen, J. Li, and A. Lerer. Scalable graph neural networks for heterogeneous graphs. *arXiv preprint arXiv:2011.09679*, 2020.

[27] L. Yu, L. Sun, B. Du, C. Liu, W. Lv, and H. Xiong. Hybrid micro/macro level convolution for heterogeneous graph learning. *arXiv preprint arXiv:2012.14722*, 2020.

[28] L. Yu, L. Sun, B. Du, C. Liu, W. Lv, and H. Xiong. Heterogeneous graph representation learning with relation awareness. *arXiv preprint arXiv:2105.11122*, 2021.

[29] H. Zeng, H. Zhou, A. Srivastava, R. Kannan, and V. K. Prasanna. Graphsaint: Graph sampling based inductive learning method. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, 2020.

[30] W. Zhang, Z. Sheng, Y. Jiang, Y. Xia, J. Gao, Z. Yang, and B. Cui. Evaluating deep graph neural networks. *arXiv preprint arXiv:2108.00955*, 2021.

[31] D. Zheng, C. Ma, M. Wang, J. Zhou, Q. Su, X. Song, Q. Gan, Z. Zhang, and G. Karypis. Distdgl: Distributed graph neural network training for billion-scale graphs. In *10th IEEE/ACM Workshop on Irregular Applications: Architectures and Algorithms, IA3 2020, Atlanta, GA, USA, November 11, 2020*, pages 36–44. IEEE, 2020.

[32] H. Zhu and P. Koniusz. Simple spectral graph convolution. In *International Conference on Learning Representations*, 2021.

[33] R. Zhu, K. Zhao, H. Yang, W. Lin, C. Zhou, B. Ai, Y. Li, and J. Zhou. Aligraph: A comprehensive graph neural network platform. *Proc. VLDB Endow.*, 12(12):2094–2105, Aug. 2019.

Table 4: Test accuracy on ogbn-mag dataset.

Methods	Validation Accuracy	Test Accuracy
R-GCN	40.84±0.41	39.77±0.46
SIGN	40.68±0.10	40.46±0.12
HGT	49.84±0.47	49.27±0.61
R-GSN	51.82±0.41	50.32±0.37
HGConv	53.00±0.18	50.45±0.17
R-HGNN	53.61±0.22	52.04±0.26
NARS	53.72±0.09	52.40±0.16
NARS-SAGN+SLE	55.91±0.17	54.40±0.15
NARS-GAMLP+RLU	57.02±0.41	**55.90±0.27**

A Experiments on ogbn-mag

A.1 Compared Baselines

Ogbn-mag dataset is a heterogeneous graph consists of 1,939,743 nodes and 21,111,007 edges of different types. For comparison, we choose eight baseline methods from the OGB ogbn-mag leaderboard: R-GCN [19], SIGN [9], HGT [12], R-GSN [24], HGConv [27], R-HGNN [28], NARS [26], and NARS-SAGN+SLE [21].

A.2 Adapt GAMLP to Heterogeneous Graphs

In its original design, GAMLP does not support training on heterogeneous graphs. Here we imitate the model design of NARS to adapt GAMLP to heterogeneous graphs.

First, we sample subgraphs from the original heterogeneous graphs according to relation types and regard the subgraph as a homogeneous graph although it may have different kinds of nodes and edges. Then, on each subgraph, the propagated features of different steps are generated. The propagated features of the same propagation step across different subgraphs are aggregated using 1-d convolution. After that, aggregated features of different steps are fed into our GAMLP to get the final results. This variant of our GAMLP is called NARS-GAMLP as it mimics the design of NARS.

As ogbn-mag dataset only contains node features for "paper" nodes, we here adopt the ComplEx algorithm [22] to generate features for other nodes.

A.3 Experiment Results

We report the test accuracy of our proposed GAMLP and GAMLP+RLU on ogbn-mag dataset in Table 4. It can be seen that our NARS-GAMLP achieves the state-of-the-art performance and outperforms the current state-of-the-art model NARS-SAGN+SLE by a significant margin of 1.50%.

Table 5: Detailed hyperparameter setting on OGB datasets.

Datasets	attention type	hidden size	num layer in JK	num layer	activation
ogb-products	Recursive	512	/	4	leaky relu, a=0.2
ogb-papers100M	JK	1024	4	6	sigmoid
ogb-mag	JK	512	4	4	leaky relu, a=0.2

Table 6: Detailed hyperparameter setting on OGB datasets.

Datasets	hops	hops for label	input dropout	attention dropout	dropout
ogb-products	5	9	0.2	0.5	0.5
ogb-papers100M	6	9	0	0	0.5
ogb-mag	5	3	0.1	0	0.5

Table 7: Detailed hyperparameter setting on OGB datasets.

Datasets	gamma	threshold	temperature	batch size	stages
ogb-products	0.1	0.85	1	50000	400, 300, 300, 300
ogb-papers100M	1	0	0.001	5000	100, 150, 150, 150
ogb-mag	10	0.4	1	10000	250, 200, 200, 200

B Detailed Hyperparameters

We provide the detailed hyperparameter setting on GAMLP+RLU in Table 5, 6 and 7 to help reproduce the results.