GRAPH ATTENTION MULTI-LAYER PERCEPTRON

Anonymous authors
Paper under double-blind review

ABSTRACT

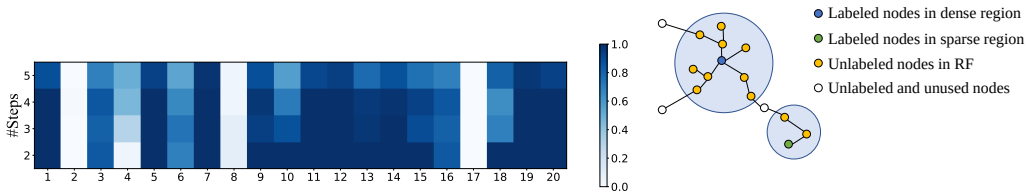
Recently, graph neural networks (GNNs) have achieved a stride of success in many graph-based applications. However, most GNNs suffer from a critical issue: representation learned is constructed based on a fixed k-hop neighborhood and insensitive to individual needs for each node, which greatly hampers the performance of GNNs. To satisfy the unique needs of each node, we propose a new architecture – Graph Attention Multi-Layer Perceptron (GAMLP). This architecture combines multi-scale knowledge and learns to capture the underlying correlations between different scales of knowledge with two novel attention mechanisms: Recursive attention and Jumping Knowledge (JK) attention. Instead of using node feature only, the knowledge within node labels is also exploited to reinforce the performance of GAMLP. Extensive experiments on 12 real-world datasets demonstrate that GAMLP achieves state-of-the-art performance while enjoying high scalability and efficiency.

1 INTRODUCTION

Graph Neural Networks (GNNs) generalize convolutional neural networks to graph-structured data and have achieved great success in a wide range of tasks, including node classification, link prediction, and recommendation. (Kipf & Welling, 2016; Hamilton et al., 2017; Bo et al., 2020; Cui et al., 2020; Fan et al., 2019). Through stacking K graph convolution layers, GNNs learn node representations by utilizing information from the K-hop neighborhood and thus enhance the performance by getting more unlabeled nodes involved in the training process. In such a GNN model, the nodes within the K-hop neighborhood of a specific node are called this node's Receptive Field (RF). As the size of RF grows exponentially to the number of GNN layers, the rapidly expanding RF incurs high computation and memory costs in a single machine. Besides, even in a distributed environment, GNN has to pull a great number of neighboring node features to compute the representation of each node, leading to high communication cost (Zheng et al., 2020).

Many recent advancements towards scalable GNNs are based on model simplification. For example, Simplified GCN (SGC) (Wu et al., 2019) decouples the feature propagation and the non-linear transformation process, and the former is executed during pre-processing. Unlike the sampling-based methods (Hamilton et al., 2017), which still need feature propagation during each training epoch, this time-consuming process in SGC is only executed once, and only the nodes of the training set are involved in the training process. As a result, SGC is computation and memory-efficient in a single machine and scalable in distributed settings since it does not require each machine to fetch neighboring node features during the model training process. Despite the high efficiency and scalability, SGC simply preserves a fixed RF for all the nodes by assigning them the same feature propagation depth. Such a fixed propagation mechanism in SGC disables its ability to exploit knowledge within neighborhoods of different sizes.

Lines of other simplified models have been proposed to learn better node representations exploiting multi-scale knowledge. SIGN (Frasca et al., 2020) proposes to concatenate all the propagated features without information loss, while S^2GC (Zhu & Koniusz, 2021) averages all these propagated features to generate the combined feature. Although multi-scale knowledge is considered, the importance and correlations between multiple scales are ignored. Being the first attempt to explore the correlations between different scales of knowledge, GBP (Chen et al., 2020b) adopts a heuristic constant decay factor for the weighted average for propagated features at different propagation steps. Motivated by Personalized PageRank, the large-scale features has a higher risk of over-smoothing, and they will contribute less to the combination in GBP.



(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.

Unfortunately, the coarse-grained, layer-wise combination prevents these methods from unleashing their full potential. As shown in Figure 1(a), different nodes require different propagation steps to achieve optimal predictive accuracy. Besides, assigning the same weight distribution to propagated features along with propagation depth to all the nodes may be unsuitable due to the inconsistent RF expansion speed shown in Figure 1(b). However, nodes in most existing GNNs are restricted to a fixed-hop neighborhood and insensitive to the actual demands of different nodes. This imperfection either makes that long-range dependencies cannot be fully leveraged due to limited hops/layers or loses local information by introducing many irrelevant nodes into the receptive fields for many nodes when increasing the number of propagation depth (Chen et al., 2020a; Li et al., 2018; Xu et al., 2018).

The above observations motivate us to explicitly learn the importance and correlation of multi-scale knowledge in a node-adaptive manner. To this end, we develop a new architecture – Graph Attention Multi-Layer Perceptron (GAMLP) – that could automatically exploit the knowledge over different neighborhoods at the granularity of nodes. GAMLP achieves this by introducing two novel attention mechanisms: *Recursive attention* and *Jumping Knowledge (JK) attention*. These two attention mechanisms can capture the complex correlations between propagated features at different propagation depths in a node-adaptive manner. Consequently, our architecture has the same benefits as the existing simplified and scalable GNN models while providing much better performance derived from its ability to utilizes a node-adaptive receptive field. Moreover, the proposed attention mechanisms can be applied to both node features and labels over neighborhoods with different sizes. By combining these two categories of information together, GAMLP could achieve the best of both worlds in terms of accuracy.

Our contributions are as follows: (1) *New perspective.* To the best of our knowledge, we are the first to explore both node-adaptive feature and label propagation schemes for scalable GNNs. (2) *Novel method.* We propose GAMLP, a scalable, efficient, and deep graph model. (3) *State-of-the-art performance.* Experimental results demonstrate that GAMLP achieves state-of-the-art performance on 12 benchmark datasets while maintains high scalability and efficiency. In particular, GAMLP outperforms the competitive baseline GraphSAINT (Zeng et al., 2020) in terms of accuracy by a margin of 0.42%, 3.02% and 0.44% on PPI, Flickr, and Reddit datasets under the inductive setting, while achieving up to $45\times$ training speedups in the large ogbn-products dataset. Remarkably, under the transductive setting in large OGB datasets, the accuracy of GAMLP exceeds the current state-of-the-art method by 1.03% and $\mathbf{1}.32\%$ on the ogbn-products and ogbn-papers100M datasets, respectively.

2 PRELIMINARIES

2.1 PROBLEM FORMULATION

We consider an undirected graph $\mathcal{G} = (\mathcal{V},\mathcal{E})$ with $|\mathcal{V}| = n$ nodes, $|\mathcal{E}| = m$ edges, and c different node classes. We denote by $\mathbf{A}$ the adjacency matrix of $\mathcal{G}$, weighted or not. Nodes can possibly have features vector of size f, stacked up in an $n \times f$ matrix $\mathbf{X}$. $\mathbf{D} = \text{diag}\,(d_1, d_2, \cdots, d_N) \in \mathbb{R}^{n\times n}$ 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 $\mathcal{V}_l$.

2.2 SCALABLE GNNS

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 (Hamilton et al., 2017) randomly samples a fixed-size set of neighbors for computation in each mini-batch. VR-GCN (Chen et al., 2018a) analyzes the variance reduction, and it reduces the size of samples with additional memory cost. For the layer-wise sampling, Fast-GCN (Chen et al., 2018b) samples a fixed number of nodes at each layer, and ASGCN (Huang et al., 2018) proposes the adaptive layer-wise sampling with better variance control. In the graph level, Cluster-GCN (Chiang et al., 2019) firstly clusters the nodes and then samples the nodes in the clusters, and GraphSAINT (Zeng et al., 2020) directly samples a subgraph for mini-batch training. Orthogonal to model simplification, sampling has already been widely used in many GNNs and GNN systems (Zheng et al., 2020; Zhu et al., 2019; Fey & Lenssen, 2019). However, these sampling-based GNNs are imperfect because they still face high communication costs, and the sampling quality highly influences the model performance.

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) (Wu et al., 2019), 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 $\widetilde{\mathbf{D}}^{-1/2}\widetilde{\mathbf{A}}\widetilde{\mathbf{D}}^{-1/2}$ (Klicpera et al., 2019), the transition probability matrix $\widetilde{\mathbf{A}}\widetilde{\mathbf{D}}^{-1}$ (Zeng et al., 2020), or the reverse transition probability matrix $\widetilde{\mathbf{D}}^{-1}\widetilde{\mathbf{A}}$ (Xu et al., 2018), 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 (Frasca et al., 2020) proposes to concatenate the propagated features at different propagation depth after simple linear transformation: $[\mathbf{X}^{(0)}\mathbf{W}_0, \mathbf{X}^{(1)}\mathbf{W}_1, ..., \mathbf{X}^{(K)}\mathbf{W}_K]$. S^2GC (Zhu & Koniusz, 2021) 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 (Chen et al., 2020b) 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 process from a node-wise perspective and each node in GAMLP has a personalized combination of different steps of the propagated features.

2.3 LABEL UTILIZATION ON GNNS.

Labels of training nodes are conventionally only used as supervision signals in loss functions in most graph learning methods. However, there also exist some graph learning methods that directly exploit the labels of training nodes. Among them, the label propagation algorithm (Zhu & Ghahramani, 2002) is the most well-known one. It simply regards the partially observed label matrix $\mathbf{Y} \in \mathbb{R}^{N \times C}$ as input features for nodes in the graph and propagates the input features through the graph structure, where C is the number of candidate classes. UniMP (Shi et al., 2020) proposes to map the partially observed label matrix $\mathbf{Y}$ to the dimension of the node feature matrix $\mathbf{X}$ and add these two matrices together as the new input feature. To fight against the label leakage problem, UniMP further randomly masks the training nodes during every training epoch.

Instead of using only the hard training labels, Correct & Smooth (Huang et al., 2020) first trains a simple model such as an MLP and gets this model's predicted soft labels for unlabeled nodes. Then, it propagates the learning errors on the labeled nodes to connected nodes and smooths the output in a

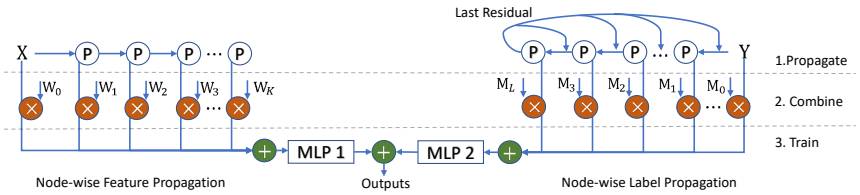


Figure 2: Overview of the proposed GAMLP, including (1) feature and label propagation, (2) combine the propagated features and labels with RF attention, and (3) MLP training. Note that both the feature and label propagation can be pre-processed.

Personalized PageRank manner like APPNP (Klicpera et al., 2019). Besides, SLE (Sun & Wu, 2021) decouples the label utilization procedure in UniMP, and executes the propagation in advance. Unlike UniMP, "label reuse" (Wang et al., 2021) concatenates the partially observed label matrix $\mathbf{Y}$ with the node feature matrix $\mathbf{X}$ to form the new input matrix. Concretely, it fills the missing elements in the partially observed label matrix $\mathbf{Y}$ with the soft label predicted by the model, and this newly generated $\mathbf{Y}'$ is again concatenated with $\mathbf{X}$ and then fed into the model to generate new predictions.

3 Graph Attention Multi-Layer Perceptron

3.1 Architecture Overview

As shown in Fig. 2, GAMLP decomposes the end-to-end GNN training into three parts: feature and label propagation, feature and label combination with RF attention, and the MLP training. As the feature and label 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 and labels under different receptive fields, thus boosting model performance.

3.2 Node-wise Feature and Label Propagation

Node-wise 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:

$$\mathbf{X}^{(k)} \leftarrow \hat{\mathbf{A}}\mathbf{X}^{(k-1)}, \ \forall k = 1, \ldots, K, \tag{2}$$

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

After K-step feature propagation shown in E.q. 2, 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:

$$\mathbf{H}_{\mathbf{X}} = \sum_{k=0}^{K} \mathbf{W}_k \mathbf{X}^{(k)}, \tag{3}$$

where $\mathbf{W}_k = Diag(\eta_k) \in \mathbb{R}^{n \times n}$ is the diagonal matrix derived from vector η_k, and $\eta_k \in \mathbb{R}^n$ is a vector derived from vector $\eta_k[i] = w_i(k), 1 \leq i \leq n$, and $w_i(k)$ measures the importance of the k-step propagated feature for node v_i.

Node-wise Label Propagation. We use a scalable and node-adaptive way to take advantage of the node labels of the training set. Concretely, the label embedding matrix $\mathbf{Y} \in \mathbb{R}^{n \times c}$ ($\mathbf{Y}^{(0)}$) is propagated with the normalized adjacency matrix $\hat{\mathbf{A}}$:

$$\mathbf{Y}^{(l)} \leftarrow \hat{\mathbf{A}}\mathbf{Y}^{(l-1)}, \ \forall l = 1, \ldots, L, \tag{4}$$

After L-step label propagation, we get a list of propagated labels under different propagation steps: $[\mathbf{Y}^{(0)}, \mathbf{Y}^{(1)}, \mathbf{Y}^{(2)}, ..., \mathbf{Y}^{(L)}]$. Generally, the propagated label $\mathbf{Y}^{(l)}$ is closer to the original label matrix $\mathbf{Y}^{(0)}$ with smaller propagation step l, and thus face a higher risk of data leakage problem if it is directly used as the model input. We propose last residual connection to solve this problem.

Definition 3.1 (**Last Residual Connection**). *Given the propagation step l, and a list of propagated labels: $[\mathbf{Y}^{(0)}, \mathbf{Y}^{(1)}, \mathbf{Y}^{(2)}, ..., \mathbf{Y}^{(L)}]$, we smooth each label $\mathbf{Y}^{(l)}$ with the smoothed label $\mathbf{Y}^{(L)}$:*

$$\hat{\mathbf{Y}}^{(l)} \leftarrow (1-\alpha_l)\mathbf{Y}^{(l)} + \alpha_l \mathbf{Y}^{(L)},\ l = 1, \ldots, L, \tag{5}$$

where $\alpha_l = \cos\left(\frac{\pi l}{2L}\right)$ controls the proportion of $\mathbf{Y}^{(L)}$ in the l-step propagated label.

Similar to the node-wise feature propagation introduced in Sec. 3.2, we propose to average these propagated labels in a weighted manner:

$$\mathbf{H}_{\mathbf{Y}} = \sum_{l=0}^{L} \hat{\mathbf{W}}_l \hat{\mathbf{Y}}^{(l)}. \tag{6}$$

3.3 Node-adaptive Attention Mechanisms

To satisfy different RF requirements for each node, we introduce two RF attention mechanisms to get $w_i(k)$. Note that these attention mechanisms can be used in both the feature and label propagation, and we introduce them from a feature perspective here. To apply them for node-wise label propagation, we only need to replace the feature $\mathbf{X}_i$ in Eq. 7 and Eq. 8 with the label $\mathbf{Y}_i$.

Definition 3.2 (**Recursive Attention**). *At each propagation step l, suppose $s \in \mathbb{R}^d$ is a learnable parameter vector, we recursively measure the feature information gain compared with the previous combined feature as:*

$$\widetilde{\mathbf{X}}_i^{(l)} = \mathbf{X}_i^{(l)} \parallel \sum_{k=0}^{l-1} w_i(k)\mathbf{X}_i^{(k)}, \quad \widetilde{w}_i(l) = \delta(\widetilde{\mathbf{X}}_i^{(l)} \cdot s), \quad w_i(l) = e^{\widetilde{w}_i(l)} / \sum_{k=0}^{K} e^{\widetilde{w}_i(k)}. \tag{7}$$

As $\widetilde{\mathbf{X}}_i^{(l-1)} \in \mathbb{R}^d$ 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. A larger $w_i(l)$ indicates 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.

Jumping Knowledge Network (JK-Net) (Xu et al., 2018) 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. Concretely, GAMLP with JK attention includes two branches: the concatenated JK branch and the attention-based combination branch.

Definition 3.3 (**JK Attention**). *Given the MLP prediction of the JK branch as $\mathbf{E}_i = MLP(\mathbf{X}_i^{(1)} \parallel \mathbf{X}_i^{(2)} \parallel ... \parallel \mathbf{X}_i^{(K)}) \in \mathbb{R}^{Kf}$, the combination weight is defined as:*

$$\widetilde{\mathbf{X}}_i^{(l)} = \mathbf{X}_i^{(l)} \parallel \mathbf{E}_i, \quad \widetilde{w}_i(l) = \delta(\widetilde{\mathbf{X}}_i^{(l)} \cdot s), \quad w_i(l) = e^{\widetilde{w}_i(l)} / \sum_{k=0}^{K} e^{\widetilde{w}_i(k)}. \tag{8}$$

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.

3.4 Model Training

Both the combined feature $\mathbf{H}_{\mathbf{X}}$ and combined label $\mathbf{H}_{\mathbf{Y}}$ are transformed with MLP, and then be added to get the final output embedding:

$$\widetilde{\mathbf{H}} = \text{MLP}(\mathbf{H}_{\mathbf{X}}) + \beta \text{MLP}(\mathbf{H}_{\mathbf{Y}}), \tag{9}$$

where β is a hyper-parameter that measures the importance of the combined label. For example, some graphs have 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 graph features.

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

$$\mathcal{L}_{CE} = -\sum_{i \in \mathcal{V}_l} \sum_{j} \mathbf{Y}_{ij} \log(\text{softmax}(\widetilde{\mathbf{H}})_{ij}), \tag{10}$$

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

3.5 Properties of GAMLP

High Efficiency and Scalability. Compared with the previous GNNs (e.g., GCN and GraphSAGE), our proposed GAMLP only need to do the feature and label propagation only once. Suppose P and Q are the number of layers in MLP trained with feature and labels, and k is the sampled nodes, the time complexity of GAMLP is $\mathcal{O}(Pnf^2 + Qnc^2)$, which is smaller than the complexity of GraphSAGE (i.e., $\mathcal{O}(k^K nf^2)$). Besides, it also cost less memory than the sampling-based GNNs, and thus can scale to a larger graph in a single machine. Notably, like other simplified GNNs (i.e., SGC and SIGN), GAMLP can pre-compute the propagated features and labels only once. It doesn't need to pull the intermediate representation of other nodes during the MLP training. Therefore, it can also be well adapted to the distributed environment. Further details can be found in Appendix A.3.

Deep propagation. With our recursive and JK attention, GAMLP can support large propagation depth without the over-smoothing issue since each node can get the node personalized combination weights for different propagated features and labels according to its demand. Such characteristic is essential for sparse graph, i.e., sparse labels, edges, and features. For example, a graph with a low label rate or edge rate can increase the propagation depth to spread the label supervision over the full graph. Each node can utilize the high-order graph structure information with deep propagation and then boost the node classification performance. Further details is in Appendix B.2.

3.6 Relation with current methods

GAMLP vs. GBP. Both GAMLP and GBP propose to combine the propagated features under different propagation steps. However, GBP adopts a layer-wise propagation scheme and ignores the inconsistent receptive field 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 issue, even propagating the same step. GAMLP considers the feature and label propagation in a more fine-grained node perspective.

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 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. 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.

GAMLP vs. SAGN. SAGN also proposes to do node-specific propagation in GNN. Concretely, SAGN learns the node-specific attention weights with the original node feature. Unlike SAGN, GAMLP adopts two attention mechanisms to learn the interactions between the propagated features over different sizes of receptive fields. Besides, node-wise label propagation is also employed in GAMLP for better utilization of node labels.

4 Experiments

In this section, we verify the effectiveness of GAMLP on 12 real-world graph datasets under both the transductive and inductive settings. We aim to answer the following four questions. **Q1:** Can GAMLP outperform the state-of-the-art GNN methods? **Q2:** If so, where does the performance gain of GAMLP come from? **Q3:** How about the efficiency of GAMLP compared with current GNN methods? **Q4:** How does GAMLP perform when applied to highly sparse graphs (i.e., given few

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
UNIMP	82.6±0.4	72.5±0.9	80.1±0.5	83.9±0.8	92.0±1.1	92.4±0.3	93.5±0.8
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

edges and low label rate)? More experimental results about the heterogeneous graph, propagation depth and interpretability can be found in Appendix B.

4.1 Experimental Setup

Datasets. We evaluate the predictive accuracy of GAMLP under both transductive and inductive settings. For transductive settings, we conduct experiments on nine transductive datasets: three citation network datasets (Cora, Citeseer, PubMed) (Sen et al., 2008), two user-item datasets (Amazon Computer, Amazon Photo), two co-author datasets (Coauthor CS, Coauthor Physics) (Shchur et al., 2018), and two OGB datasets (ogbn-products, ogbn-papers100M) (Hu et al., 2021). For inductive settings, we perform the comparison experiments on three inductive datasets: PPI, Flickr, and Reddit (Zeng et al., 2019). The statistics about these datasets can be found in Table 10 in Appendix C.1.

Baselines. Under the transductive setting, we compare GAMLP with the following representative baseline methods: GCN (Kipf & Welling, 2016), GAT (Veličković et al., 2017), JK-Net (Xu et al., 2018), ResGCN (Li et al., 2019), APPNP (Klicpera et al., 2018), AP-GCN (Spinelli et al., 2020), SGC (Wu et al., 2019), SIGN (Frasca et al., 2020), S^2GC (Zhu & Koniusz, 2021), and GBP (Chen et al., 2020b). For the comparison in the OGB datasets, we choose the top-performing methods from the OGB leaderboard along with their accuracy results. Under the inductive setting, we choose following representative methods: SGC (Wu et al., 2019), GraphSAGE (Hamilton et al., 2017), Cluster-GCN (Chiang et al., 2019), and GraphSAINT (Zeng et al., 2019).

In addition, two variants of GAMLP are tested in the evaluation: GAMLP(JK) and GAMLP(R). "JK" and "R" stand for adopting "JK attention" and "Recursive attention" for the node-adaptive attention mechanism, respectively.

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
GBP	92.82±0.10	80.48±0.05
SIGN	92.99±0.04	80.52±0.16
DeeperGCN	92.38±0.09	80.98±0.20
UniMP	93.08±0.17	82.56±0.31
SAGN	93.09±0.04	81.20±0.07
SAGN+0-SLE	93.27±0.04	83.29±0.18
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
SAGN+0-SLE	71.06±0.08	67.55±0.15
GAMLP(JK)	71.92±0.04	**68.07±0.10**
GAMLP(R)	71.21±0.03	67.46±0.02

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

4.2 End-to-end Comparison

Transductive Performance. To answer **Q1**, we report the transductive performance of GAMLP in Tables 1, 2, and 3. We observe that both variants of GAMLP outperform all the baseline methods on almost all the datasets. For example, on the small Citeseer dataset, GAMLP(JK) outperforms the state-of-the-art method S^2GC by a large margin of 1.6%; on the medium-sized dataset Amazon Computers, the predictive accuracy of GAMLP (JK) exceeds the one of the state-of-the-art method GBP by 1.0%; on the two large OGB datasets, GAMLP takes the lead by 1.03% and 1.32% on ogbn-products and ogbn-papers100M, respectively. Furthermore, the experimental results illustrate that the contest between the two variants of GAMLP is not a one-horse race, which suggests that these two different attention mechanisms both have their irreplaceable sense in some ways.

Inductive Performance. We also evaluate GAMLP under the inductive setting. The experiment results in Table 4 show that GAMLP consistently outperforms all the baseline methods. The leading advantage of GAMLP(JK) over SOTA inductive method – GraphSAINT is more than 3.0% on the widely-used dataset – Filckr. The impressive performance of GAMLP under the inductive setting illustrates that GAMLP is alpowerful in predicting the properties of unseen nodes.

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

4.3 Ablation Study

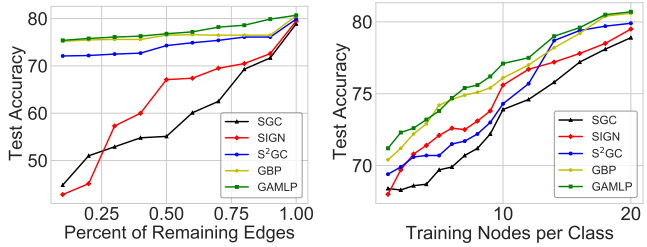
To answer **Q2**, we focus on two modules in GAMLP: (1) label utilization; (2) attention mechanism in the node-wise propagation. For the second one, we evaluate the effects of different choices for reference vectors in the JK attention.

Label Utilization. In this part, we evaluate whether adding last residual connection and making use of training labels really help or not. 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. The experimental results in Table 5 show that utilizing labels brings huge performance gain to GAMLP: from 81.43% to 83.59%. The performance drop from removing the last residual connections ("-plain_label" in Table 5) is significant since directly adopting the raw training labels leads to the overfitting issue. 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. It further demonstrates the effectiveness of our proposed last residual connections.

Reference Vector in Attention Mechanism. In this part, we study the role of the reference vector (originally set as the concatenated features from different propagation steps) in our proposed JK attention. 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 predictive accuracy of each variant on the PubMed dataset is reported in Table 6. The experimental results show that our original choice of the reference

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



(a) Edge Sparsity (b) Label Sparsity

Figure 3: Test accuracy on PubMed dataset under different levels of label and edge sparsity.

vector is the best among itself and its three variants. 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 receptive fields with different sizes.

4.4 Efficiency Comparison

To answer **Q3**, we evaluate the efficiency of each method on the ogbn-products dataset. We compare the efficiency of GAMLP with sampling-based GraphSAINT and Cluster-GCN, graph-wise-propagation-based SGC, and layer-wise-propagation-based SIGN. Table 7 illustrates the relative training time of each compared method along with its predictive accuracy. The training time of SGC is set to 1 as reference. We observe that (1) sampling-based methods (e.g., GraphSAINT) consume much more time than graph/layer-wise-propagation based methods (e.g., SGC, SIGN) due to the high computation cost introduced by the sampling process; (2) the two variants of GAMLP achieve the best predictive accuracy while requiring comparable training time with SGC.

4.5 Experiments on Sparse Graphs

To answer **Q4**, we conduct experiments to evaluate the predictive accuracy of GAMLP when faced with edge and label sparsity problems, where the number of edges and training labels are highly scarce. We randomly remove a fixed percentage of edges from the original graph to simulate the edge sparsity problem. The removed edges are exactly the same for all the compared methods. Besides, we enumerate the number of training nodes per class from 1 to 20 to evaluate the performance of GAMLP given different levels of label sparsity. The experimental results in Figure 3 show that GAMLP consistently outperforms all the baselines when faced with different levels of edge and label sparsity. This experiment further demonstrates the effectiveness of our proposed node-wise propagation scheme. The node-wise propagation enables GAMLP to better capture long-range dependencies, which is crucial when applying GNN methods to highly sparse graphs.

5 Conclusion

We presented Graph Attention Multilayer Perceptron (GAMLP), a scalable, efficient, and deep graph model based on receptive field attention. GAMLP introduced two new attention mechanisms: recursive attention and JK attention, which enables to learn the representations over RF with different sizes in a node-adaptive manner. Extensive experiments on 12 graph datasets verified the effectiveness of the proposed method. GAMLP moves forward the performance boundary of scalable GNNs, especially on large-scale graphs. This initial attempt also motivates several interesting future directions: (1) exploring other attention mechanisms and (2) studying the mechanisms on heterogeneous graphs.

6 REPRODUCIBILITY STATEMENT

The source code of GAMLP can be found in Anonymous Github (`https://anonymous.4open.science/r/ICLR-GAMLP`). To ensure reproducibility, we have provided the overview of datasets and baselines in Section 4.1 and Table 10 in Appendix C.1. The detailed hyperparameter settings for our GAMLP can be found in Appendix C.2. Our experimental environment is presented in Appendix C.1, and please refer to "README.md" in the Github repository for more details.

REFERENCES

Deyu Bo, Xiao Wang, Chuan Shi, Meiqi Zhu, Emiao Lu, and Peng Cui. Structural deep clustering network. In *Proceedings of The Web Conference 2020*, pp. 1400–1410, 2020.

Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, and Xu Sun. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 3438–3445, 2020a.

Jianfei Chen, Jun Zhu, and Le 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*, pp. 941–949, 2018a.

Jie Chen, Tengfei Ma, and Cao 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*, 2018b.

Ming Chen, Zhewei Wei, Bolin Ding, Yaliang Li, Ye Yuan, Xiaoyong Du, and Ji-Rong Wen. Scalable graph neural networks via bidirectional propagation. *arXiv preprint arXiv:2010.15421*, 2020b.

Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *SIGKDD*, pp. 257–266, 2019.

Ganqu Cui, Jie Zhou, Cheng Yang, and Zhiyuan Liu. Adaptive graph encoder for attributed graph embedding. In *SIGKDD*, pp. 976–985, 2020.

Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. Graph neural networks for social recommendation. In *The World Wide Web Conference*, pp. 417–426, 2019.

Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR 2019 Workshop on Representation Learning on Graphs and Manifolds*, 2019. URL `https://arxiv.org/abs/1903.02428`.

Fabrizio Frasca, Emanuele Rossi, Davide Eynard, Ben Chamberlain, Michael Bronstein, and Federico Monti. Sign: Scalable inception graph neural networks. *arXiv preprint arXiv:2004.11198*, 2020.

William L Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NeurIPS*, pp. 1025–1035, 2017.

Weihua Hu, Matthias Fey, Hongyu Ren, Maho Nakata, Yuxiao Dong, and Jure Leskovec. Ogb-lsc: A large-scale challenge for machine learning on graphs. *arXiv preprint arXiv:2103.09430*, 2021.

Ziniu Hu, Yuxiao Dong, Kuansan Wang, and Yizhou Sun. Heterogeneous graph transformer. In *Proceedings of The Web Conference 2020*, pp. 2704–2710, 2020.

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 preprint arXiv:2010.13993*, 2020.

Wen-bing Huang, Tong Zhang, Yu Rong, and Junzhou 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*, pp. 4563–4572, 2018.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. *arXiv preprint arXiv:1810.05997*, 2018.

Johannes Klicpera, Aleksandar Bojchevski, and Stephan 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.

Guohao Li, Matthias Muller, Ali Thabet, and Bernard Ghanem. Deepgcns: Can gcns go as deep as cnns? In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 9267–9276, 2019.

Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *European semantic web conference*, pp. 593–607. Springer, 2018.

Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Gallagher, and Tina Eliassi-Rad. Collective classification in network data. *AI Mag.*, 29(3):93–106, 2008.

Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*, 2018.

Yunsheng Shi, Zhengjie Huang, Wenjin Wang, Hui Zhong, Shikun Feng, and Yu Sun. Masked label prediction: Unified message passing model for semi-supervised classification. *arXiv preprint arXiv:2009.03509*, 2020.

Indro Spinelli, Simone Scardapane, and Aurelio Uncini. Adaptive propagation graph convolutional network. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

Chuxiong Sun and Guoshi Wu. Scalable and adaptive graph neural networks with self-label-enhanced training. *arXiv preprint arXiv:2104.09376*, 2021.

Théo Trouillon, Christopher R Dance, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. Knowledge graph completion via complex tensor factorization. *arXiv preprint arXiv:1702.06879*, 2017.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

Yangkun Wang, Jiarui Jin, Weinan Zhang, Yong Yu, Zheng Zhang, and David Wipf. Bag of tricks for node classification with graph neural networks. *arXiv preprint arXiv:2103.13355*, 2021.

Felix Wu, Tianyi Zhang, Amauri Holanda de Souza Jr, Christopher Fifty, Tao Yu, and Kilian Q Weinberger. Simplifying graph convolutional networks. *arXiv preprint arXiv:1902.07153*, 2019.

Xinliang Wu, Mengying Jiang, and Guizhong Liu. R-gsn: The relation-based graph similar network for heterogeneous graph. *arXiv preprint arXiv:2103.07877*, 2021.

Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In *ICML*, pp. 5453–5462. PMLR, 2018.

Le Yu, Leilei Sun, Bowen Du, Chuanren Liu, Weifeng Lv, and Hui Xiong. Hybrid micro/macro level convolution for heterogeneous graph learning. *arXiv preprint arXiv:2012.14722*, 2020a.

Le Yu, Leilei Sun, Bowen Du, Chuanren Liu, Weifeng Lv, and Hui Xiong. Heterogeneous graph representation learning with relation awareness. *arXiv preprint arXiv:2105.11122*, 2021.

Lingfan Yu, Jiajun Shen, Jinyang Li, and Adam Lerer. Scalable graph neural networks for heterogeneous graphs. *arXiv preprint arXiv:2011.09679*, 2020b.

Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. Graphsaint: Graph sampling based inductive learning method. In *ICLR*, 2019.

Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor 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.

Da Zheng, Chao Ma, Minjie Wang, Jinjing Zhou, Qidong Su, Xiang Song, Quan Gan, Zheng Zhang, and George 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*, pp. 36–44. IEEE, 2020.

Hao Zhu and Piotr Koniusz. Simple spectral graph convolution. In *International Conference on Learning Representations*, 2021.

Rong Zhu, Kun Zhao, Hongxia Yang, Wei Lin, Chang Zhou, Baole Ai, Yong Li, and Jingren Zhou. Aligraph: A comprehensive graph neural network platform. *Proc. VLDB Endow.*, 12(12): 2094–2105, August 2019. ISSN 2150-8097. doi: 10.14778/3352063.3352127. URL `https://doi.org/10.14778/3352063.3352127`.

Xiaojin Zhu and Zoubin Ghahramani. Learning from labeled and unlabeled data with label propagation. 2002.

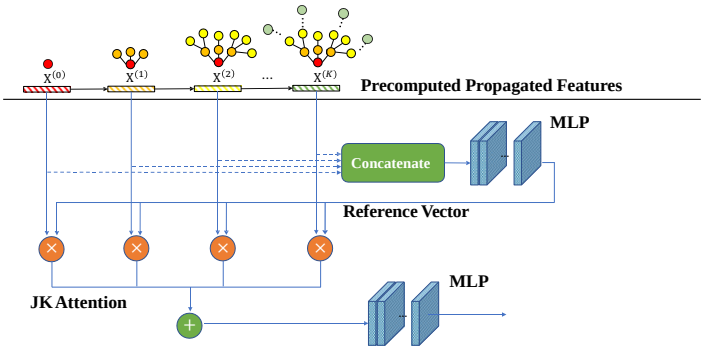


Figure 4: The architecture of GAMLP with JK Attention.

A More details about GAMLP

A.1 An example of JK attention

Fig. 4 provides a more zoomed-in look of JK attention, one of the two node-adaptive attention mechanisms we proposed. The propagated features are concatenated and then fed into an MLP to map the concatenated feature to the hidden dimension of the model. The mapped feature is then set as the reference vector of the following attention mechanism, where a linear layer is adopted to calculate the combination weight for propagated features at different propagation steps. The propagated features are then multiplied with the corresponding combination weight, and the summed results are fed into another MLP to generate final predictions.

A.2 Comparison between the label usage in UniMP and GAMLP

(1) The label usage in UniMP is coupled with the training process, making it hard to scale to large graphs. While GAMLP decouples the label usage from the training process, the label propagation process can be executed as preprocessing.

(2) The label propagation steps in UniMP are restricted to the same number of model layers. And if the number of model layers becomes large, UniMP will encounter the efficiency and scalability issues even on relatively small graphs. While the label propagation steps in GAMLP can be quite large since the label propagation is performed as preprocessing.

(3) Both UniMP and GAMLP propose approaches to fight against the label leakage issue. However, the random masking in UniMP has to be executed in each training epoch, while the last residual connection (composed of simple matrix addition) in GAMLP just needs to be executed once during preprocessing. Thus, UniMP consumes more resources than GAMLP to fight the label leakage issue.

A.3 Complexity analysis

Table 8 provides a detailed asymptotic complexity comparison between GAMLP and representative scalable GNN methods. During preprocessing, the time cost of clustering in Cluster-GCN is $\mathcal{O}(m)$ and the time complexity of most linear models is $\mathcal{O}(Kmf)$. Besides, GAMLP has an extra time cost $\mathcal{O}(Lmc)$ for the propagation of training labels. GBP takes advantage of Monte-Carlo method and conducts this process approximately with a bound of $\mathcal{O}(Knf + K\frac{\sqrt{m \lg n}}{\varepsilon})$, where ε is a error threshold. Compared with sampling-based GNNs, graph/layer/node-wise-propagation-based models usually have smaller training and inference time complexity. Memory complexity is a crucial factor in large-scale graph learning as it fundamentally determines whether it is possible to adopt the method. Compared with SIGN, both GBP and GAMLP do not need to store smoothed features at different propagation steps, and the memory complexity can be reduced from $\mathcal{O}(bLf)$ to $\mathcal{O}(bf)$.

Table 8: Algorithm analysis for existing scalable GNNs. n, m, c, and f are the number of nodes, edges, classes, and feature dimensions, respectively. b is the batch size, and k refers to the number of sampled nodes. K and L corresponds to the number of times we aggregate features and labels respectively. Besides, P and Q are the number of layers in MLP classifiers trained with features and labels respectively.

Type	Method	Pre-processing	Training	Memory
Sampling	GraphSAGE	-	$\mathcal{O}(k^K nf^2)$	$\mathcal{O}(bk^K f + Kf^2)$
	FastGCN	-	$\mathcal{O}(kKnf^2)$	$\mathcal{O}(bkKf + Kf^2)$
	Cluster-GCN	$\mathcal{O}(m)$	$\mathcal{O}(Pmf + Pnf^2)$	$\mathcal{O}(bKf + Kf^2)$
Graph-wise propagation	SGC	$\mathcal{O}(Kmf)$	$\mathcal{O}(nf^2)$	$\mathcal{O}(bf + f^2)$
Layer-wise propagation	SIGN	$\mathcal{O}(Kmf)$	$\mathcal{O}(Pnf^2)$	$\mathcal{O}(bLf + Pf^2)$
	S^2GC	$\mathcal{O}(Kmf)$	$\mathcal{O}(nf^2)$	$\mathcal{O}(bf + f^2)$
	GBP	$\mathcal{O}(Knf + K\frac{\sqrt{m \lg n}}{\varepsilon})$	$\mathcal{O}(Pnf^2)$	$\mathcal{O}(bf + Pf^2)$
Node-wise propagation	GAMLP	$\mathcal{O}(Kmf + Lmc)$	$\mathcal{O}(Pnf^2 + Qnc^2)$	$\mathcal{O}(bf + Pf^2 + Qc^2)$

Table 9: 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-GAMLP	55.52±0.08	**54.01±0.21**

B Additional experiments

B.1 Experiments on ogbn-mag

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 (Schlichtkrull et al., 2018), SIGN (Frasca et al., 2020), HGT (Hu et al., 2020), R-GSN (Wu et al., 2021), HGConv (Yu et al., 2020a), R-HGNN (Yu et al., 2021), and NARS (Yu et al., 2020b).

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 (Trouillon et al., 2017) to generate features for other nodes.

Experiment Results. We report the validation and test accuracy of our proposed GAMLP on the ogbn-mag dataset in Table 9. It can be seen from the results that NARS-GAMLP achieves great performance on the heterogeneous graph ogbn-mag, outperforming the performance of the strongest single model baseline NARS by a large margin of 1.61%.

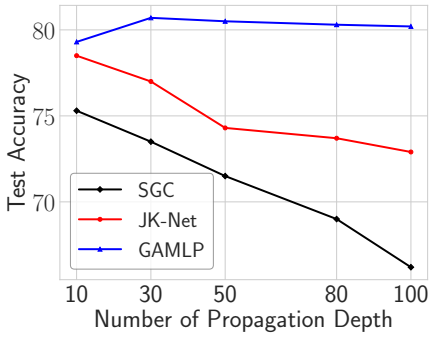


Figure 5: Test accuracy when the propagation depth increases from 10 to 100.

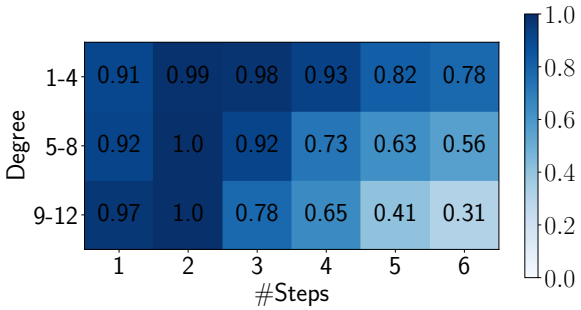


Figure 6: The average attention weights of propagated features of different steps on 60 randomly selected nodes from ogbn-products.

B.2 Deep propagation is possible

Equipped with the learnable node-wise propagation scheme, our GAMLP can still maintain high predictive accuracy even when the propagation depth is over 50. Here, we evaluate the predictive accuracy of our proposed GAMLP(JK) at propagation depth 10, 30, 50, 80, 100 on the PubMed dataset. The performance of JK-Net and SGC are also reported as baselines. The experimental results in Fig. 5 show that even at propagation depth equals 100, the predictive accuracy of our GAMLP(JK) still exceeds 80.0%, higher than the predictive accuracy of most baselines in Table 1. At the same time, the predictive accuracy of SGC and JK-Net both drops rapidly when propagation depth increases from 10 to 100.

B.3 Interpretability of the attention mechanism

GAMLP can adaptively and effectively combine multi-scale propagated features for each node. To demonstrate this, Fig. 6 shows the average attention weights of propagated features of GAMLP(JK) 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 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, indicating that our attention mechanism could prevent high-degree nodes from including excessive irrelevant nodes, leading to over-smoothing. From the two observations, we conclude that GAMLP can identify the different RF demands of nodes and explicitly weight each propagated feature.

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

Table 11: Ablation study of choices for α_l on the ogbn-products dataset.

Choices	Test Accuracy
Fixed weight	82.56±0.43
Linear-decreasing weight	82.72±0.93
Cosine function	83.59±0.05

B.4 Choices for α_l in the Last Residual Connection

Our first choice for the α_l in the last residual connection module is $\alpha_l = \frac{L-l}{L}$. However, we find that GAMLP still encounters the over-fitting issue on some datasets. Thus, we instead choose $\alpha_l = \cos(\frac{\pi l}{2L})$ to give more penalties to labels at large propagation steps. We provide the performance comparison on the ogbn-products dataset in Table 11. Three weighting schemes for the last residual connection module are tested: "Cosine function" stands for $\alpha_l = \cos(\frac{\pi l}{2L})$, the one in GAMLP; "Linear-decreasing weight" stands for $\alpha_l = \frac{L-l}{L}$; and "Fixed weight" stands for $\alpha_l = 0.7$. Table 11 shows that the weighting scheme GAMLP adopts, $\alpha_l = \cos(\frac{\pi l}{2L})$, outperforms the other two options.

C Detailed experiment setup

C.1 Experiment Environment

We provide detailed information about the datasets we adopted during the experiment in Table 10. To alleviate the influence of randomness, we repeat each method ten times and report the mean performance and the standard deviations. For the largest ogbn-papers100M dataset, we run each method five times instead. 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 C.2 for the detailed hyperparameter settings for our GAMLP.

C.2 Detailed Hyperparameters

We provide the detailed hyperparameter setting on GAMLP in Table 12, 13 and 14 to help reproduce the results. To reproduce the experimental results of GAMLP, just follow the same hyperparameter setting yet only run the first stage.

Table 12: 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	1280	4	6	sigmoid
ogb-mag	JK	512	4	4	leaky relu, a=0.2

Table 13: Detailed hyperparameter setting on OGB datasets.

Datasets	hops	hops for label	input dropout	attention dropout	dropout
ogb-products	5	10	0.2	0.5	0.5
ogb-papers100M	12	10	0	0.5	0.5
ogb-mag	5	3	0.1	0	0.5

Table 14: Detailed hyperparameter setting on OGB datasets.

Datasets	beta	patience	lr	batch size	epochs
ogb-products	1	300	0.001	50000	400
ogb-papers100M	1	60	0.0001	5000	400
ogb-mag	1	100	0.001	10000	400