DEEP GRAPH TREE NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

We propose Graph Tree Networks (GTreeNets), a self-interpretive deep graph neural network architecture which originates from the tree representation of graphs. In the tree representation, each node forms its own tree where the node itself is the root node and all its neighbors up to k-hop are the subnodes. Under the tree representation, messages propagate upward from the leaf nodes to the root node naturally to update the root node's hidden features. This message passing scheme, which has better interpretability, is essentially different from that in the vanilla Graph Convolution Network (GCN), Graph Attention Network (GAT) and many of their derivatives. Two scalable graph learning models are proposed within this GTreeNet architecture - Graph Tree Convolution Network (GTCN) and Graph Tree Attention Network (GTAN), with experimentally demonstrated state-of-the-art performance on several benchmark datasets and the capability of going deep.

1 INTRODUCTION

Graph Neural Networks (GNNs), a class of neural networks for learning on graph structured data, have been successfully applied in many areas to solve real world problems, such as link predictions in social networks (Fan et al., 2019), pattern recognition (Ju et al., 2020; Shi & Rajkumar, 2020), product recommendation and personalized search in E-commerce (Zhu et al., 2019), fraud detection (Wang et al., 2019a; Dou et al., 2020), protein interface predictions (Fout et al., 2017), power estimation and tier design in the semiconductor industry (Zhang et al., 2020; Lu et al., 2020), traffic forecasting (Yu et al., 2017), and natural language processing (Yao et al., 2019; Vashishth et al., 2020; Wu et al., 2021). Among many different graph learning approaches, the class of spatial graph convolution based models, which adopts a message passing scheme to update node features, has gained particular attention due to its simplicity yet good performance. The most representative work among this class is the Graph Convolutional Network (GCN) (Kipf & Welling, 2017) which limits the ChebNet (Defferrard et al., 2016) up to the 1st order polynomial, and together with some further approximation leading to a direct neighbor aggregation. The success of the GCN model (Ying et al., 2018; Hu et al., 2020b) has led to the rapid development in spatial convolution based models, such as the Graph Attention Network (GAT) (Velickovic et al., 2018), GraphSage (Hamilton et al., 2017), APPNP (Klicpera et al., 2018), DAGNN (Liu et al., 2020), and etc.

Today the vanilla GCN and GAT are the two most popular baseline models. One GCN or GAT layer aggregates only the direct neighbor nodes. Although they work well in many test cases, their performance degrades as stacking multiple propagation layers to achieve larger receptive fields. Xu et al. (2018) and Li et al. (2018) attribute such degradation in performance to over-smoothing effect that nodes from different classes become indistinguishable, while Liu et al. (2020) attributes that to the intertwined propagation and transformation. A small neighborhood may not provide enough information especially when nodes are sparsely labeled (Klicpera et al., 2018; Liu et al., 2020). Many recent work has been devoted to extending the size of neighborhood utilized in graph learning. APPNP (Klicpera et al., 2018) and DAGNN (Liu et al., 2020) are two most recent deep graph models with state-of-the-art performance on several popular benchmark datasets. APPNP (Klicpera et al., 2018) is developed based on personalized PageRank to preserve local information which requires fine tuning of a teleport probability. DAGNN (Liu et al., 2020) aggregates neighbors from different hops in parallel (in one layer) to utilize information from a larger receptive field as compared to GCN. Although DAGNN has achieved state-of-the-art performance, it may only be applicable to adopt simple aggregation function as the one used in GCN. It could be computationally challenging to combine such multi-hop aggregation scheme with attention-based GNN models (such as GAT, and graph transformer (Dwivedi & Bresson, 2020; Hu et al., 2020c)); because this aggregation scheme requires computation of the attention weight for each pair of a node and its k-hop neighbor, which could be a too large number. In the aggregation of k-hop neighborhood, the number of attention weights is the total number of non-zero entries in the k$^{\text{th}}$ power of the graph adjacency matrix $\boldsymbol{A}$. Although the adjacency matrix is usually sparse, its k$^{\text{th}}$ power could be non-sparse especially for undirected graphs. For example, in the Cora dataset (Wang et al., 2019b) $\boldsymbol{A}^5$ has $\sim$2.2 million non-zero entries while $\boldsymbol{A}$ has only $\sim$10k non-zero entries. Furthermore, aggregating multi-hop neighborhood in one step neglects path information which is usually important for heterogeneous graphs with rich metapath information (Fu et al., 2020; Wang et al., 2019c).

We propose Graph Tree Networks (GTreeNets), a self-interpretive deep graph neural network architecture which originates from the tree representation of graphs. In the tree representation, each node forms its own tree where the node itself is the root node and all its neighbors up to k-hop are the subnodes. Under the tree representation, message propagates upward from the leaf nodes to the root node naturally to update the root node's hidden features. This message passing scheme, which has better interpretability, is essentially different from that in the vanilla Graph Convolution Network (GCN), Graph Attention Network (GAT) and many of their derivatives. Two scalable graph learning models are proposed within this GTreeNet architecture - Graph Tree Convolution Network (GTCN) and Graph Tree Attention Network (GTAN), with experimentally demonstrated state-of-the-art performance on several benchmark datasets and the capability of going deep by stacking multiple propagation layers. Our models do not require finely-tuned hyperparameter in the propagation scheme.

This work is structured as follows. Section 2 illustrates the architecture of our Graph Tree Networks (GTreeNets) and two derivative graph learning models - Graph Tree Convolution Network (GTCN) and Graph Tree Attention Network (GTAN), along with the analysis of model complexity. Section 3 presents the experimental results including three parts: Section 3.1 shows the performance test of our GTCN and GTAN models as compared to several popular and state-of-the-art GNN models on five benchmark datasets. Section 3.2 shows the performance test for our GTCN and GTAN models at different depths as compared to the vanilla GCN and GAT models. Section 3.3 discusses the effect of adding transformation in propagation layers in our GTCN and GTAN models. We conclude our work in Section 4 and discuss insights for future work.

2 Our Approach: Graph Tree Networks

We first introduce notations used throughout this paper. We follow the general convention to use bold uppercase and lowercase to represent matrices and vectors, respectively. The topology of a graph $G = (V, E)$ with nodes V and edges E can be fully described by its adjacency matrix $\boldsymbol{A}$ and degree matrix $\boldsymbol{D}$. $|V| = N$ and $|E| = M$ are the number of nodes and edges, respectively. $\mathcal{N}_u$ denotes the set of direct neighbors (1-hop neighbors) of node u. $\boldsymbol{X} \in \mathbb{R}^{N\times D}$ represents the feature map for all nodes, where each row $\boldsymbol{x}_u \in \mathbb{R}^D$ represents the feature vector of node u with dimension of D. $\boldsymbol{Y} \in \mathbb{R}^{N\times C}$ represents the class matrix for all nodes, where each row $\boldsymbol{y}_u$ represents the class vector of node u with C classes.

2.1 Tree Representation and Network Structure

Tree is a straightforward representation for the graph topology, where each node and its neighborhood form one tree with the node itself being the root node and its neighbors being the subnodes. Figure 1 illustrates the tree representation for node 1 with up to 3-hop neighbors in the sample graph. We use a directed graph just for illustration, the tree representation also works for undirected graphs.

In Figure 1, $\boldsymbol{h}_u^k \in \mathbb{R}^F$ is the vector of node u's output hidden features at k-hop away from the root node 1. The final output of the root node 1 is denoted as $\boldsymbol{h}_1^0$. Messages propagate upward from subnodes to the root node hop by hop in the tree. Nodes may occur multiple times at the same level in the tree (like node 5 occurring twice at 2-hop as shown in Figure 1), each of them is from a different k-hop path that may pass different messages to the root node. We assume a node u at any k-hop preserves its initial information prior to receiving information from its child nodes. With this

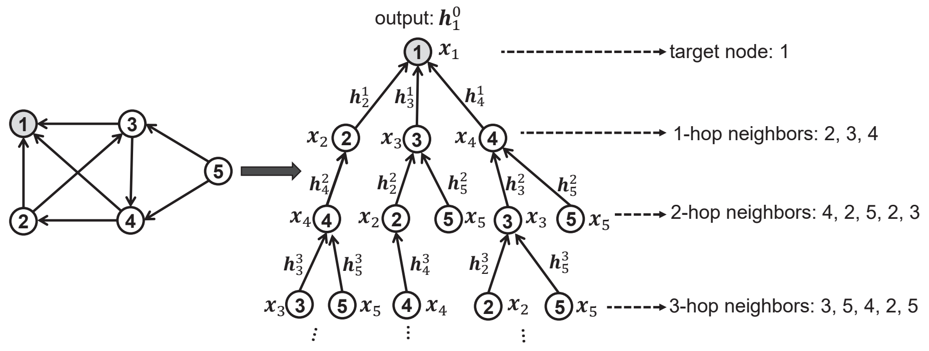

output: h_1^0
target node: 1
1-hop neighbors: 2, 3, 4
2-hop neighbors: 4, 2, 5, 2, 3
3-hop neighbors: 3, 5, 4, 2, 5

Figure 1: Sample of tree representation for node 1 (showing up to 3-hop neighborhood). The input features of a node u is denoted as $\boldsymbol{x}_u$, and the hidden features of the node u at the k^{th} hop of the root node is denoted as $\boldsymbol{h}_u^k$.

assumption, a node's features are updated by aggregating its child nodes and its own initial features in the tree representation.

This tree representation of graphs leads to a new graph learning architecture - Graph Tree Networks (GTreeNets) as shown in Appendix A.1, with the general message passing rule formulated as Equation 1:

$$\boldsymbol{h}_u^k = f_k\left(\text{aggregate}\left(\phi\left(\boldsymbol{x}_u\right), \left\{\boldsymbol{h}_v^{k+1}, \forall v \in \mathcal{N}_u\right\}\right)\right) \tag{1}$$

where $k = L-1, \dots, 0$. L is the tree depth (number of propagation layers). $\boldsymbol{x}_u$ is the vector of input features of node u. $\boldsymbol{z}_u = \phi(\boldsymbol{x}_u)$ is the transformation of input features of node u, i.e. the initial hidden features for the propagation layer. $\boldsymbol{h}_u^k$ is the vector of hidden features of node u at k-hop, and $\boldsymbol{h}_u^L = \boldsymbol{z}_u$. $f_k(\cdot)$ is a hop-wise (layer-wise) transformation function such as MLP.

Combination of different aggregation function and transformation function in Equation 1 derives various graph learning models under the GTreeNet architecture. We propose two graph tree models in Section 2.2 and 2.3: Graph Tree Convolution Network (GTCN) and Graph Tree Attention Network (GTAN).

2.1.1 Relation to Propagation Scheme in GCN-like models

The general message passing rule in GCN-like models such as GCN (Kipf & Welling, 2017), GAT (Velickovic et al., 2018), and GraphSage (Hamilton et al., 2017) is

$$\boldsymbol{h}_u^n = f_{n-1}\left(\text{aggregate}\left(\boldsymbol{h}_u^{n-1}, \left\{\boldsymbol{h}_v^{n-1}, \forall v \in \mathcal{N}_u\right\}\right)\right) \tag{2}$$

Comparing to Equation 1, it is notable that the message passing scheme in our proposed GTreeNet is essentially different from the one adopted by the GCN-like models. In GTreeNet, we update a node's hidden features by aggregating its neighbor's hidden features and its own initial features. While in GCN-like models, a node's hidden features are updated by aggregating its neighbor's hidden features and its own hidden features from the last propagation layer. Our propagation scheme preserves the local information when stacking multiple propagation layers while the propagation scheme in GCN-like models does not. Note that we use n to denote the propagation layer number in Equation 2 to avoid ambiguities.

2.2 Graph Tree Convolution Network

The propagation rule of our proposed Graph Tree Convolution Network is

$$\boldsymbol{h}_u^k = \sum\nolimits_{v \in \mathcal{N}_u} \hat{\boldsymbol{A}}_{uv} \boldsymbol{h}_v^{k+1} + \hat{\boldsymbol{A}}_{uu} \boldsymbol{z}_u \tag{3}$$

where $\hat{\boldsymbol{A}} = \tilde{\boldsymbol{D}}^{-\frac{1}{2}} \tilde{\boldsymbol{A}} \tilde{\boldsymbol{D}}^{-\frac{1}{2}}$ is the symmetric normalized adjacency matrix[1]. $\tilde{\boldsymbol{A}} = (\boldsymbol{A} + \boldsymbol{I})$ is the adjacency matrix with added self-loops. $\tilde{\boldsymbol{D}}$ is the degree matrix with self-loops where $\tilde{\boldsymbol{D}}_{uu} = \sum_v \tilde{\boldsymbol{A}}_{uv}$. $\boldsymbol{z}_u = \text{MLP}(\boldsymbol{x}_u)$.

[1]We find that using $\hat{\boldsymbol{A}} = \tilde{\boldsymbol{D}}^{-1} \tilde{\boldsymbol{A}}$ yields similar performance.

As compared to the general propagation rule in GTreeNet shown in Equation 1, our GTCN model uses an MLP as the transformation function $\phi(\cdot)$, an identity function as $f_k(\cdot)$, and a weighted sum function as the aggregation function with weights being the elements of the normalized adjacency matrix.

The mathematical description of our GTCN model is then formulated as

$$\begin{aligned}
&\boldsymbol{Z} = \text{MLP}\left(\boldsymbol{X}\right) \\
&\boldsymbol{H}^L = \boldsymbol{Z} \\
&\boldsymbol{H}^k = \boldsymbol{A}_1\boldsymbol{H}^{k+1} + \boldsymbol{A}_2\boldsymbol{Z}, \quad k = L-1, L-2, \ldots, 0 \\
&\boldsymbol{Y}_{\text{out}} = \text{softmax}\left(\boldsymbol{H}^0\boldsymbol{W}\right)
\end{aligned} \tag{4}$$

where $\boldsymbol{Z} \in \mathbb{R}^{N\times F}$ is the initial hidden feature map obtained by applying an MLP to the input feature map $\boldsymbol{X}$. $\boldsymbol{H}^k \in \mathbb{R}^{N\times F}$ is the hidden feature map at the k^{th} hop. $\boldsymbol{A}_1$ and $\boldsymbol{A}_2$ are the non-diagonal and diagonal component of the normalized adjacency matrix, respectively. $\boldsymbol{W} \in \mathbb{R}^{F\times C}$ is the trainable weight matrix.

Note that our model separates the transformation from propagation with no trainable parameters required in the propagation layers. By aggregating a node's neighbor information with its own initial information, we always preserve the local information of a node. These two characteristics make our model capable of going deep without comprising performance as demonstrated in Section 3.2.

2.2.1 Relation to GCN and APPNP

GCN. The propagation rule in GCN is

$$\boldsymbol{h}_u^n = \text{ReLU}\left(\left(\sum\nolimits_{v\in\mathcal{N}_u} \hat{\boldsymbol{A}}_{uv}\boldsymbol{h}_v^{n-1} + \hat{\boldsymbol{A}}_{uu}\boldsymbol{h}_u^{n-1}\right)\boldsymbol{W}^{n-1}\right) \tag{5}$$

which uses weighted sum to aggregate the hidden features of node u and its neighbors from the last propagation layer, and an MLP with ReLU activation function as $f_{n-1}(\cdot)$ to transform the feature representation.

Except for the different information used in aggregation as discussed in Section 2.1.1, another major difference between our GTCN model and the GCN model is that a transformation function is applied after each propagation layer in GCN which causes intertwined propagation and transformation, and introduces more trainable parameters. These differences make the GCN model subject to compromised performance when stacking multiple propagation layers.

APPNP. The propagation rule in APPNP is

$$\boldsymbol{h}_u^n = (1-\alpha)\left(\sum\nolimits_{v\in\mathcal{N}_u} \hat{\boldsymbol{A}}_{uv}\boldsymbol{h}_v^{n-1} + \hat{\boldsymbol{A}}_{uu}\boldsymbol{h}_u^{n-1}\right) + \alpha\boldsymbol{z}_u \tag{6}$$

which uses weighted sum to aggregate the hidden features of node u and its neighbors from the last propagation layer, as well as its initial hidden feature. Note that the APPNP also adopts an identity function as $f_{n-1}(\cdot)$, i.e. no transformation is applied after propagation.

APPNP uses the hidden features of node u from the last propagation layer ($\boldsymbol{h}_u^{n-1}$) in aggregation to update its hidden feature while our GTCN model does not. In addition, APPNP introduces a teleport probability α to adjust the influence from the neighborhood which requires fine tuning for different graphs.

Note that the APPNP model can be derived from our GTreeNet by adding self-loops in the graph. Therefore, node u is also a child node of itself when building the tree representation. In this case, the general propagation rule in the resulting tree representation becomes:

$$\boldsymbol{h}_u^k = f_k\left(\text{aggregate}\left(\phi\left(\boldsymbol{x}_u\right), \boldsymbol{h}_u^{k+1}, \left\{\boldsymbol{h}_v^{k+1}, \forall v \in \mathcal{N}_u\right\}\right)\right) \tag{7}$$

2.2.2 Relation to Child-Sum Tree-LSTM

The Child-Sum Tree-LSTM (Tai et al., 2015) can also be derived from our GTreeNet, which uses RNN aggregation function to aggregate a node's initial features and its neighbors with no further transformation.

2.3 Graph Tree Attention Network

The propagation rule of our proposed Graph Tree Attention Network is

$$\boldsymbol{h}_u^k = \mathrm{ELU}\left(\sum\nolimits_{v\in\mathcal{N}_u} \alpha_{u,v}^k \boldsymbol{h}_v^{k+1} + \alpha_{u,u}^k \boldsymbol{z}_u\right) \tag{8}$$

where $\alpha_{u,v}^k$ and $\alpha_{u,u}^k$ are trainable attention weights calculated as

$$\begin{aligned} \alpha_{u,v}^k &= \mathrm{softmax}\left(e_{u,v}^k\right) \\ e_{u,v}^k &= \begin{cases} \mathrm{LeakyReLU}\left(\boldsymbol{w}^T\left[\boldsymbol{z}_u \parallel \boldsymbol{h}_v^{k+1}\right]\right) & \text{if } u \neq v \\ \mathrm{LeakyReLU}\left(\boldsymbol{w}^T\left[\boldsymbol{z}_u \parallel \boldsymbol{z}_u\right]\right) & \text{if } u = v \end{cases} \end{aligned} \tag{9}$$

As compared to the general propagation rule in GTreeNet shown in Equation 1, our GTAN model uses an MLP as the transformation function $\phi(\cdot)$, an ELU function as $f_k(\cdot)$, and an attention based weighted sum function as the aggregation function.

The mathematical description of our GTAN model is then formulated as

$$\begin{aligned} &\boldsymbol{Z} = \mathrm{MLP}(\boldsymbol{X}) \\ &\boldsymbol{H}^L = \boldsymbol{Z} \\ &\boldsymbol{h}_u^k = \mathrm{ELU}\left(\sum\nolimits_{v\in\mathcal{N}_u} \alpha_{u,v}^k \boldsymbol{h}_v^{k+1} + \alpha_{u,u}^k \boldsymbol{z}_u\right), \quad k = L-1, L-2, \ldots, 0 \\ &\boldsymbol{Y}_{\text{out}} = \mathrm{softmax}\left(\boldsymbol{H}^0 \boldsymbol{W}_0\right) \end{aligned} \tag{10}$$

where $\alpha_{u,v}^k$ and $\alpha_{u,u}^k$ are calculated as in Equation 9.

2.3.1 Relation to GAT

The propagation rule in GAT is

$$\boldsymbol{h}_u^n = \mathrm{ELU}\left(\left(\sum\nolimits_{v\in\mathcal{N}_u} \alpha_{u,v}^n \boldsymbol{h}_v^{n-1} + \alpha_{u,u}^n \boldsymbol{h}_u^{n-1}\right) \boldsymbol{W}^{n-1}\right) \tag{11}$$

which uses attention based weighted sum to aggregate the hidden features of node u and its neighbors from the last propagation layer, and an MLP with ELU activation function as $f_{n-1}(\cdot)$ to transform the feature representation.

Except for the different information used in aggregation as discussed in Section 2.1.1, another major difference between our GTAN model and the GAT model is that a transformation function with trainable parameters is applied after each propagation layer in GAT which introduces more trainable parameters. This might make the learning difficult and jeopardize the model performance.

2.4 Model Complexity

It is straightforward to see from Equation 4 and 10 that the time complexity of our GTCN and GTAN models is $O(L|E|F)$ which is in the same order as the vanilla GCN and GAT. Here $|E|$ is the number of edges, F is the dimension of hidden features, and L is the model depth.

3 Experiments

In this section, we conduct our experiments in three parts. The first one in Section 3.1 is to demonstrate the performance of our GTCN and GTAN models. We choose a depth of 10 for both models to aggregate information from up to 10-hop neighbors in the graph, and compare their performances against two current state-of-the-art deep graph models: APPNP and DAGNN with the same depth. We also include the most popular GCN and GAT models, as well as the Child-Sum Tree-LSTM as our baseline models. The second one in Section 3.2 is to demonstrate the deep capability of our models by comparing our model performance with that of the vanilla GCN and GAT at different model depths. The Child-Sum Tree-LSTM is also included as a comparison. The third one in Section 3.3 is to discuss the effect of adding an MLP transformation function $f_k(\cdot)$ with trainable parameters in our GTCN and GTAN models. The codes to reproduce our experiments are publicly available.

Table 1: Statistics of datasets.

	Cora	**Citeseer**	**PubMed**	**Coauthor-CS**	**ogbn-arxiv**
Nodes	2708	3327	19717	18333	169343
Edges	10556	9228	88651	81894	1166243
Features/Node	1433	3703	500	6805	128
Classes	7	6	3	15	40
Label Rate	5.2%	3.6%	0.3%	1.6%	53.7%
Training	20/class	20/class	20/class	20/class	90941
Validation	500	500	500	450	29799
Test	1000	1000	1000	17583	48603

Datasets. We work on four popular benchmark datasets (Cora, Citeseer, PubMed and MS Coauthor-CS) from DGL library (Wang et al., 2019b) and one OGB dataset ogbn-arxiv (Hu et al., 2020a) to demonstrate the performance of our proposed models. The four datasets from the DGL library are undirected graphs. Fixed data splits are provided for the Cora, Citeseer and PubMed datasets. For the Coauthor-CS dataset, we split the data in the following way. We randomly selected 20 training samples per class and 30 validation samples per class, and all remaining samples are used as the test samples. We repeat the above process three times yielding three splits for the Coauthor-CS dataset to avoid any bias. The ogbn-arxiv dataset is a homogeneous, unweighted, directed graph representing the citation between all Computer Science arXiv papers indexed by MAG. Each paper has 128-dimensional features obtained by averaging the embeddings of words. The task is to predict the subject areas (40 total classes) of the paper.

The statistics of the benchmark datasets are summarized in Table 1. Note that the adjacency matrix for the Citeseer and PubMed datasets provided in DGL (Wang et al., 2019b) includes several self-loops. We remove these self-loops to obtain the adjacency matrix in our experiments.

3.1 Performance of Graph Tree Convolution and Attention Networks

Baseline models. We compare our GTCN and GTAN models to five GNN models: the vanilla GCN (Kipf & Welling, 2017), GAT (Velickovic et al., 2018), Child-Sum Tree-LSTM (Tai et al., 2015) and the current state-of-the-art APPNP (Klicpera et al., 2018) and DAGNN (Liu et al., 2020) models (we are also interested in comparing with the AdaGCN model (Sun et al., 2021) but the source code for the model has not been publicly available yet).

Model hyperparameters. For fair comparisons, all models are set up in almost the same settings as in their original papers. All implementations use Adam optimizer with two hyperparameters: learning rate and weight decay.

The detailed experiment setup is described in Appendix A.2.

Results of accurancy. We take 30 runs of each model on each DGL dataset, and 10 runs of each model on the obgn-arxiv dataset[2]. The test results of classification accuracy are represented by mean $\pm$ std. Test results on the four DGL datasets are summarized in Table 2. The Macro-F1 scores of each model on the four DGL dataset are summarized in Table 6 (See Appendix A.3). Test results on the obgn-arxiv dataset are summarized in Table 3.

Our GTCN model outperforms the current state-of-the-art baseline models APPNP and DAGNN on the Cora, Citeseer, and Coauthor-CS datasets. Our GTAN model achieves state-of-the-art performance on the Coauthor-CS dataset and the ogbn-arxiv dataset.

The ogbn-arxiv dataset is much larger with much more complex neighborhood structures as compared to the four DGL datasets. This may explain why our GTAN model outperforms GTCN model. GTAN has trainable pair-wise attention weights in each propagation layers, while GTCN does not have trainable weights in all propagation layers, i.e. GTCN employs fixed weights in the message

[2]This is set by OGB.

Table 2: Summary of average classification accuracy ± one standard deviation (in percent) after filtering out the top and bottom 10% data on DGL datasets

Method	Cora	Citeseer	PubMed	Coauthor-CS (1)	Coauthor-CS (2)	Coauthor-CS (3)
GCN (hop = 2)	81.5 ± 0.2	71.6 ± 0.3	79.0 ± 0.3	90.7 ± 0.2	90.2 ± 0.2	89.7 ± 0.2
GAT (hop = 2)	83.0 ± 0.5	71.1 ± 0.9	77.6 ± 0.4	90.4 ± 0.3	90.3 ± 0.5	89.5 ± 0.4
Tree-LSTM (hop = 2)	81.9 ± 0.7	68.6 ± 1.0	76.6 ± 0.6	91.0 ± 0.2	90.8 ± 0.2	90.8 ± 0.2
APPNP (hop = 10)	83.6 ± 0.5	71.6 ± 0.5	79.6 ± 0.2	91.8 ± 0.4	91.8 ± 0.3	91.4 ± 0.3
DAGNN (hop = 10)	84.0 ± 0.5	72.6 ± 0.5	**79.6 ± 0.4**	90.4 ± 0.3	90.3 ± 0.4	89.6 ± 0.6
GTAN (hop = 10, ours)	83.4 ± 1.0	71.4 ± 0.4	79.4 ± 0.2	92.2 ± 0.3	92.4 ± 0.2	92.0 ± 0.3
GTCN (hop = 10, ours)	**84.5 ± 0.6**	**72.9 ± 0.5**	79.2 ± 0.3	**92.7 ± 0.1**	**92.5 ± 0.1**	**92.4 ± 0.2**

Table 3: Summary of average classification accuracy ± one standard deviation (in percent) on ogbn-arxiv dataset.

Method	ogbn-arxiv
GCN (hop = 3)	71.7 ± 0.2
GAT (hop = 3)	71.8 ± 0.1
Tree-LSTM (hop = 3)	71.4 ± 0.3
APPNP (hop = 5)	71.5 ± 0.1
DAGNN (hop = 16)	72.1 ± 0.3
GTAN (hop = 4, ours)	**72.7 ± 0.2**
GTCN (hop = 5, ours)	72.3 ± 0.2

aggregation. Therefore GTAN could capture more complex neighborhood structure than GTCN, and is more favored in such case.

Training time per epoch. The average training time per epoch on the DGL datasets are summarized in Table 4. Obviously, our GTCN model is scalable to large graph and computational efficient even with deep layers. Attention based models require computation for attention weights between each pair of a node and its neighbor nodes in each propagation layer, therefore are less efficient for dense graphs.

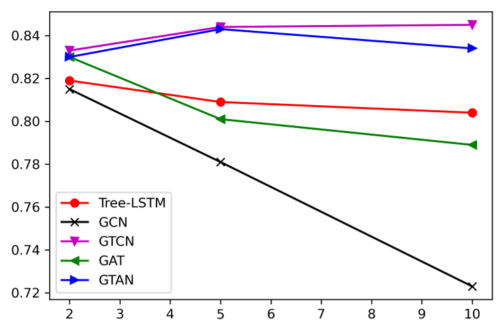
3.2 Test of model depth capability

In this section, we demonstrate our models' capability of going deep by testing our models on the four DGL datasets with different numbers of propagation layers. We compare our GTCN model with the vanilla GCN, and our GTAN model with the vanilla GAT. We also include the Child-Sum Tree-LSTM model in the comparison as it is a derivative model within our GTreeNet architecture. All models are tested with a depth of 2, 5 and 10, respectively. The classification accuracy is obtained with the same process as described in Section 3.1.

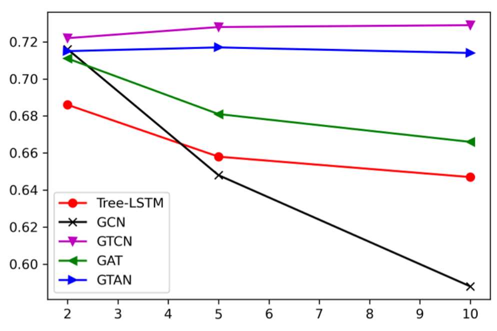
Model hyperparameters. For all models under test, we use 64 hidden units for every intermediate hidden layer. For the vanilla GCN, the dropout is 0.5, learning rate is 0.01 and weight decay is 5e-4 for all datasets and all model depths. For the vanilla GAT, we use one attention head for all experiments. At depth 2, the hyperparameters are the same as used in Section 3.1. At depth 5, the hyperparameters are mostly the same as used at depth 2 except that we set 0 for the attention dropout for all datasets. At depth 10, the hyperparameters are fine-tuned with the layer dropout and the attention dropout of (0.2, 0) for all datasets. The learning rate and weight decay are 0.01 and

Table 4: Average training time per epoch (ms/epoch) on DGL datasets.

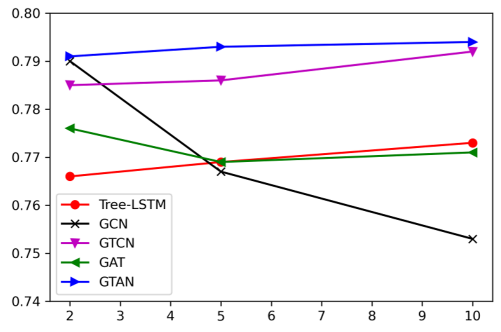
Method	**Cora**	**Citeseer**	**PubMed**	**Coauthor-CS**
GCN (hop = 2)	3.5	3.3	3.6	7.4
GAT (hop = 2)	4.7	4.6	5.0	12.0
GCN (hop = 10)	14.3	13.9	19.1	25.9
GAT (hop = 10)	17.6	17.8	21.8	32.7
APPNP (hop = 10)	3.7	3.5	3.8	12.0
DAGNN (hop = 10)	4.7	4.6	5.1	13.2
GTAN (hop = 10, ours)	17.7	18.1	19.4	34.4
GTCN (hop = 10, ours)	3.7	3.6	8.1	18.6



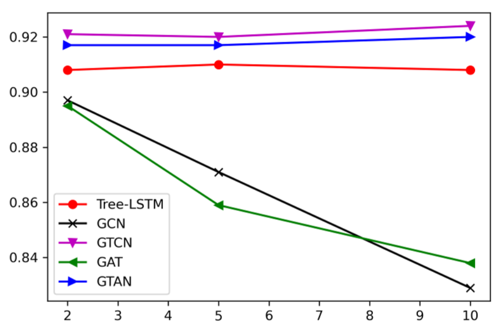
(a) Cora



(b) Citeseer



(c) Pubmed



(d) Coauthor-CS

Figure 2: Model performance in accuracy at different depths, tested on four datasets: Cora (a), Citeseer (b), Pubmed (c), and Coauthor-CS (d).

5e-3 respectively for all datasets. For the Child-Sum Tree-LSTM, the settings are consistent for all three depths and four datasets, with the input dropout and the dropout after the LSTM cell being (0.8, 0.6), learning rate being 0.01 and weight decay being 5e-4. For our GTCN model, the settings are the same as used in Section 3.1. For our GTAN model, the settings for depth 2 and 5 are mostly the same as described in Section 3.1, except that the attention dropout is set to 0.6 for the Cora and Citeseer datasets.

Results. The test results of all five models at different depths are illustrated in Figure 2. More details are summarized in Table 7, 8, 9, and 10 respectively in Appendix A.4. The results show that the performance of Graph Tree Network based models (GTCN, GTAN and Child-Sum Tree-LSTM) does not compromise in general when going deep, while the performance of the vanilla GCN and GAT models degrades significantly when going deep which is known as the "over-smoothing" issue. This comparison demonstrates the superiority of the message passing scheme in our GTreeNet.

3.3 Effect of adding Transformation after Propagation

We find that adding an MLP network as f_k in Equation 1 will degrade the model performance. To demonstrate this experimentally, we construct a variant of our GTCN model by adding an MLP with the ReLU activation function after each propagation layer, and call it a GTCN2 model. The propagation rule in GTCN2 model is now $\boldsymbol{H}^k = \text{ReLU}\left(\left(\boldsymbol{A}_1\boldsymbol{H}^{k+1} + \boldsymbol{A}_2\boldsymbol{Z}\right)\boldsymbol{W}\right)$. We compare the performance of the GTCN and GTCN2 models on the Cora dataset at different depths (2, 5, and 10). Same tests are also performed on our GTAN model, and all results are summarized in Table 5.

Adding an MLP network after each propagation layer introduces additional trainable parameters, which may cause worse fitting especially when training samples are limited. The results also show that the model performance degrades as stacking more propagation layers after adding an MLP network after each propagation layer, which are consistent with the findings by Liu et al. (2020).

Table 5: Model performance in accuracy with and without nonlinear layers, tested on the Cora dataset.

Method (Cora)	**Depth = 2**	**Depth = 5**	**Depth = 10**
GTCN	83.3 ± 0.3	84.4 ± 0.6	84.5 ± 0.5
GTCN2	80.7 ± 0.5	80.6 ± 0.6	80.0 ± 0.8
GTAN	83.0 ± 0.4	83.7 ± 0.7	83.3 ± 1.0
GTAN2	80.8 ± 0.6	79.1 ± 0.6	78.6 ± 1.0

4 Conclusion

In this paper, we propose an innovative self-interpretive graph learning architecture: deep Graph Tree Network (GTreeNet), which is naturally derived from the tree representation of graphs. The message passing scheme in the GTreeNet has better interpretability than that in the vanilla GCN and GAT models. Two models within the GTreeNet architecture are proposed - Graph Tree Convolution Network (GTCN) and Graph Tree Attention Network (GTAN), with demonstrated state-of-the-art performance on several benchmark datasets and the capability of going deep by stacking multiple propagation layers. The major advantage of the message passing scheme in the GTreeNet is that models adopting this message passing scheme can go deep by stacking multiple propagation layers instead of calculating multi-hop information directly, where only information from the direct neighbors is used in each layer. This advantage may allow us to extend our GTCN and GTAN models to heterogeneous graphs. Dealing with metapaths in heterogeneous graphs is one of the biggest hurdles confronted by most of the current heterogeneous models like MAGNN (Fu et al., 2020) and HAN (Wang et al., 2019c), as the number of metapaths increases dramatically when aggregating multi-hop neighbors. The message passing scheme in our models helps avoid dealing with metapaths explicity as information on metapaths is self-contained by stacking multiple propagation layers. This opens future research opportunities. The general propagation rule we formulate in the GTreeNet architecture may also open new research opportunities by exploring combination of different aggregation schemes and transformation functions.

References

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in neural information processing systems*, 29:3844–3852, 2016.

Yingtong Dou, Zhiwei Liu, Li Sun, Yutong Deng, Hao Peng, and Philip S Yu. Enhancing graph neural network-based fraud detectors against camouflaged fraudsters. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pp. 315–324, 2020.

Vijay Prakash Dwivedi and Xavier Bresson. A generalization of transformer networks to graphs. *arXiv preprint arXiv:2012.09699*, 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.

Alex Fout, Jonathon Byrd, Basir Shariat, and Asa Ben-Hur. Protein interface prediction using graph convolutional networks. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL `https://proceedings.neurips.cc/paper/2017/file/f507783927f2ec2737ba40afbd17efb5-Paper.pdf`.

Xinyu Fu, Jiani Zhang, Ziqiao Meng, and Irwin King. Magnn: Metapath aggregated graph neural network for heterogeneous graph embedding. In *Proceedings of The Web Conference 2020*, pp. 2331–2341, 2020.

William L Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pp. 1025–1035, 2017.

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *arXiv preprint arXiv:2005.00687*, 2020a.

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *arXiv preprint arXiv:2005.00687*, 2020b.

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

Xiangyang Ju, Steven Farrell, Paolo Calafiura, Daniel Murnane, Lindsey Gray, Thomas Klijnsma, Kevin Pedro, Giuseppe Cerati, Jim Kowalkowski, Gabriel Perdue, et al. Graph neural networks for particle reconstruction in high energy physics detectors. *arXiv preprint arXiv:2003.11603*, 2020.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL `https://openreview.net/forum?id=SJU4ayYgl`.

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

Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *Thirty-Second AAAI conference on artificial intelligence*, 2018.

Meng Liu, Hongyang Gao, and Shuiwang Ji. Towards deeper graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 338–348, 2020.

Yi-Chen Lu, Sai Surya Kiran Pentapati, Lingjun Zhu, Kambiz Samadi, and Sung Kyu Lim. Tp-gnn: a graph neural network framework for tier partitioning in monolithic 3d ics. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6. IEEE, 2020.

Weijing Shi and Raj Rajkumar. Point-gnn: Graph neural network for 3d object detection in a point cloud. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 1711–1719, 2020.

Ke Sun, Zhanxing Zhu, and Zhouchen Lin. Ada{gcn}: Adaboosting graph convolutional networks into deep models. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=QkRbdiiEjM`.

Kai Sheng Tai, Richard Socher, and Christopher D Manning. Improved semantic representations from tree-structured long short-term memory networks. *arXiv preprint arXiv:1503.00075*, 2015.

Shikhar Vashishth, Naganand Yadati, and Partha Talukdar. Graph-based deep learning in natural language processing. In *Proceedings of the 7th ACM IKDD CoDS and 25th COMAD*, pp. 371–372. 2020.

Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL `https://openreview.net/forum?id=rJXMpikCZ`.

Daixin Wang, Jianbin Lin, Peng Cui, Quanhui Jia, Zhen Wang, Yanming Fang, Quan Yu, Jun Zhou, Shuang Yang, and Yuan Qi. A semi-supervised graph attentive network for financial fraud detection. In *2019 IEEE International Conference on Data Mining (ICDM)*, pp. 598–607. IEEE, 2019a.

Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, Chao Ma, Lingfan Yu, Yu Gai, Tianjun Xiao, Tong He, George Karypis, Jinyang Li, and Zheng Zhang. Deep graph library: A graph-centric, highly-performant package for graph neural networks. *arXiv preprint arXiv:1909.01315*, 2019b.

Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S Yu. Heterogeneous graph attention network. In *The World Wide Web Conference*, pp. 2022–2032, 2019c.

Lingfei Wu, Yu Chen, Heng Ji, and Yunyao Li. Deep learning on graphs for natural language processing. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: Tutorials*, pp. 11–14, 2021.

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

Liang Yao, Chengsheng Mao, and Yuan Luo. Graph convolutional networks for text classification. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pp. 7370–7377, 2019.

Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 974–983, 2018.

Bing Yu, Haoteng Yin, and Zhanxing Zhu. Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. *arXiv preprint arXiv:1709.04875*, 2017.

Yanqing Zhang, Haoxing Ren, and Brucek Khailany. Grannite: Graph neural network inference for transferable power estimation. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6. IEEE, 2020.

Rong Zhu, Kun Zhao, Hongxia Yang, Wei Lin, Chang Zhou, Baole Ai, Yong Li, and Jingren Zhou. Aligraph: a comprehensive graph neural network platform. *arXiv preprint arXiv:1902.08730*, 2019.

A Appendix

A.1 GTreeNet Illustration

Figure 3 is an illustration of our Graph Tree Network (GTreeNet).

A.2 Experimemt Setup

The maximum training epochs is set to 1000 for all experiments.

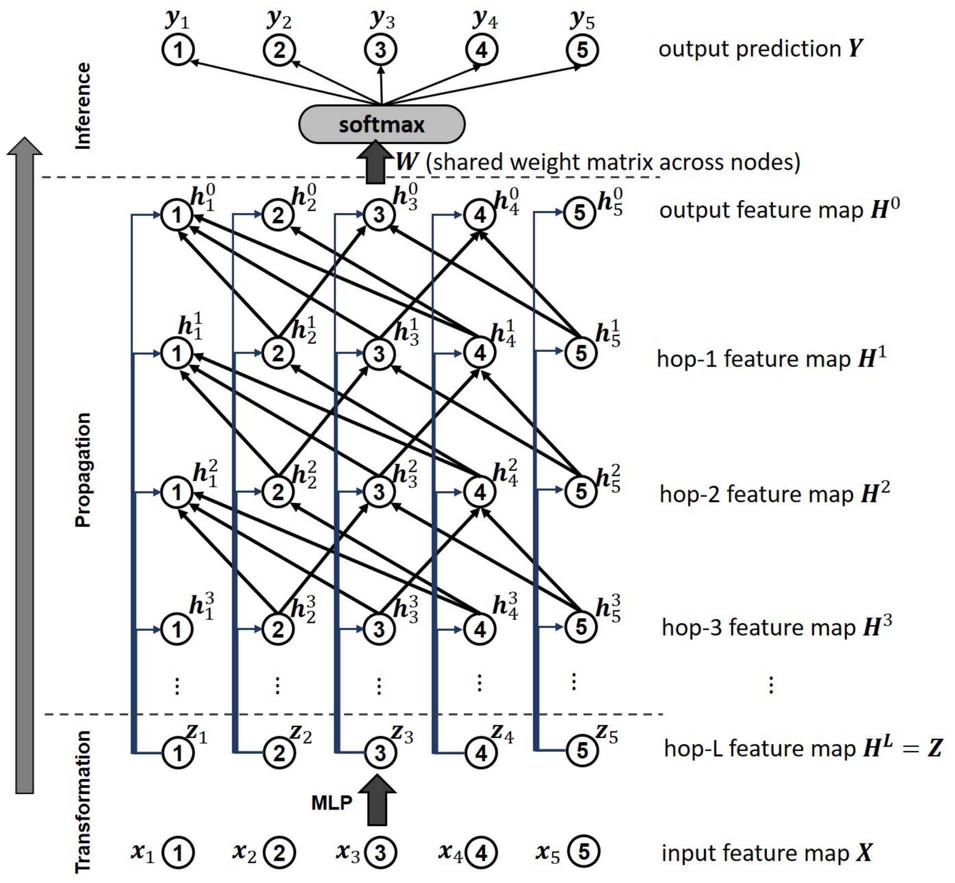

Inference
Propagation
Transformation
y_1 y_2 y_3 y_4 y_5
output prediction $\boldsymbol{Y}$
softmax
$\boldsymbol{W}$ (shared weight matrix across nodes)
output feature map $\boldsymbol{H}^0$
hop-1 feature map $\boldsymbol{H}^1$
hop-2 feature map $\boldsymbol{H}^2$
hop-3 feature map $\boldsymbol{H}^3$
hop-L feature map $\boldsymbol{H}^L = \boldsymbol{Z}$
MLP
input feature map $\boldsymbol{X}$

Figure 3: Illustration of GTreeNet on the sample graph in Figure 1.

A.2.1 Test on DGL datasets

For the vanilla GCN, we use two layers with 64 hidden units and the dropout is 0.5 for all datasets. The learning rate and weight decay are 0.01 and 5e-4 respectively for all datasets.

For the vanilla GAT, we also use two layers with 64 hidden units, and other hyperparameters are fine-tuned to obtain the best performance. One attention head is used for both layers[3]. The layer dropout and attention dropout are 0.8 and 0.8 respectively for the Cora and Citeseer datasets, 0.8 and 0.2 respectively for the PubMed and Coauthor-CS datasets. The learning rate and weight decay are 0.01 and 5e-4 respectively for all datasets.

For the 2-layer Child-Sum Tree-LSTM model, the input dropout and the dropout after the LSTM cell are 0.8 and 0.6 respectively. The learning rate and weight decay are 0.01 and 5e-4 respectively for all datasets.

For the APPNP, DAGNN, and our GTCN and GTAN models, we use a maximum hop of 10, the same as that used by the original paper of the APPNP model (Klicpera et al., 2018). 64 hidden units are used for these four deep models. For the APPNP, the dropout is 0.5 for the first MLP layer and the edge dropout is 0.5 for the propagation layer, the teleport probability α is 0.1 for the Cora, Citeseer and Pubmed datasets, and 0.2 for the Coauthor-CS dataset.[4] For the DAGNN, the dropout is 0.8 and learning rate is 0.01 for all datasets. The weight decay is 2e-2 for the Citeseer dataset, and 5e-3 for the Cora, PubMed and Coauthor-CS datasets. For our GTCN, we have two dropouts: one

[3]The original paper (Velickovic et al., 2018) uses 8 heads for the first layer, but we find that 1 head already achieves matching results.

[4]We have validated with greedy search that the α value used in the original paper is approximately optimal.

Table 6: Summary of average Macro-F1 score $\pm$ one standard deviation (in percent) after filtering out the top and bottom 10% data.

Method	**Cora**	**Citeseer**	**PubMed**	**Coauthor-CS (1)**	**Coauthor-CS (2)**	**Coauthor-CS (3)**
GCN (hop = 2)	80.5 $\pm$ 0.4	68.8 $\pm$ 0.3	78.8 $\pm$ 0.3	87.7 $\pm$ 0.3	87.5 $\pm$ 0.4	86.4 $\pm$ 0.3
GAT (hop = 2)	81.6 $\pm$ 0.5	67.0 $\pm$ 0.7	77.1 $\pm$ 0.3	87.8 $\pm$ 0.4	87.8 $\pm$ 0.4	86.9 $\pm$ 0.4
Tree-LSTM (hop = 2)	80.7 $\pm$ 0.6	64.2 $\pm$ 0.9	76.2 $\pm$ 0.6	88.8 $\pm$ 0.2	88.5 $\pm$ 0.2	88.1 $\pm$ 0.2
APPNP (hop = 10)	81.8 $\pm$ 0.4	68.3 $\pm$ 0.3	78.9 $\pm$ 0.2	89.1 $\pm$ 0.5	89.6 $\pm$ 0.3	88.8 $\pm$ 0.5
DAGNN (hop = 10)	82.5 $\pm$ 0.4	68.9 $\pm$ 0.3	78.9 $\pm$ 0.3	87.5 $\pm$ 0.7	87.4 $\pm$ 0.5	86.9 $\pm$ 0.5
GTAN (hop = 10, ours)	81.7 $\pm$ 0.6	68.0 $\pm$ 0.4	78.6 $\pm$ 0.3	90.2 $\pm$ 0.4	**90.6 $\pm$ 0.3**	89.8 $\pm$ 0.3
GTCN (hop = 10, ours)	**82.8 $\pm$ 0.7**	**69.1 $\pm$ 0.6**	**78.9 $\pm$ 0.3**	**90.6 $\pm$ 0.2**	90.5 $\pm$ 0.2	**90.2 $\pm$ 0.2**

Table 7: Model performance in accuracy with different depths, tested on the Cora dataset

Method (Cora)	**Depth = 2**	**Depth = 5**	**Depth = 10**
GCN	81.5 $\pm$ 0.2	78.1 $\pm$ 0.6	72.3 $\pm$ 1.4
GAT	83.0 $\pm$ 0.5	80.1 $\pm$ 0.6	78.9 $\pm$ 0.9
Tree-LSTM	81.9 $\pm$ 0.7	80.9 $\pm$ 0.7	80.4 $\pm$ 0.7
GTAN (ours)	83.0 $\pm$ 0.4	84.3 $\pm$ 0.5	83.4 $\pm$ 1.0
GTCN (ours)	83.3 $\pm$ 0.3	84.4 $\pm$ 0.6	84.5 $\pm$ 0.5

is for the initial one-layer MLP, and the other is for the propagation layer. Corresponding dropout values are set to (0.6, 0.6) for the Cora dataset, (0.8, 0.6) for the Citeseer dataset, (0.8, 0.5) for the Pubmed dataset and (0.6, 0.2) for the Coauthor-CS dataset. The learning rate is 0.01 for the Cora, Citeseer and Coauthor-CS datasets, and 0.02 for the PubMed dataset. The weight decay is 5e-4 for the Cora, Citeseer and PubMed datasets, and 5e-3 for the Coauthor-CS dataset. For our GTAN, we have the same two dropouts as for the GTCN, which are set to (0.6, 0) for the Cora, Citeseer and PubMed datasets, and (0.2, 0.2) for the Coauthor-CS dataset. The learning rate is set to 0.01 for all datasets. The weight decay is set to 5e-4 for the Cora, Citeseer and PubMed datasets, and 5e-3 for the Coauthor-CS dataset.

The early-stopping patience number is set to 100 for the DAGNN model (the same number is used in the original paper (Liu et al., 2020)), 300 for our GTAN model, and 200 for all other models.

A.2.2 Test on OGB dataset

For the vanilla GCN, we use three layers with 256 hidden units and the dropout is 0.5. The learning rate and weight decay are 0.01 and 0, respectively.

For the vanilla GAT, we use three layers with 128 hidden units. One attention head is used for all layers. The input dropout and attention dropout are 0.2 and 0, respectively. The learning rate and weight decay are 0.01 and 0, respectively.

For Child-Sum Tree-LSTM model, we use three layers with 256 hidden units. The input dropout and the dropout after the LSTM cell are 0.2 and 0, respectively. The learning rate and weight decay are 0.01 and 0, respectively.

For the APPNP model, we use 5 layers with 256 hidden units. The input dropout and the dropout after each propagation layer are 0.2 and 0, respectively. The learning rate and weight decay are 0.01 and 0, respectively.

Table 8: Model performance in accuracy with different depths, tested on the Citeseer dataset

Method (Citeseer)	**Depth = 2**	**Depth = 5**	**Depth = 10**
GCN	71.6 ± 0.3	64.8 ± 1.0	58.8 ± 1.9
GAT	71.1 ± 0.9	68.1 ± 0.9	66.6 ± 1.0
Tree-LSTM	68.6 ± 1.0	65.8 ± 1.1	64.7 ± 1.2
GTAN (ours)	71.5 ± 0.6	71.7 ± 0.6	71.4 ± 0.4
GTCN (ours)	72.2 ± 0.7	72.8 ± 0.5	72.9 ± 0.5

Table 9: Model performance in accuracy with different depths, tested on the PubMed dataset

Method (PubMed)	**Depth = 2**	**Depth = 5**	**Depth = 10**
GCN	79.0 ± 0.3	76.7 ± 0.6	75.3 ± 0.9
GAT	77.6 ± 0.4	76.9 ± 0.5	77.1 ± 0.7
Tree-LSTM	76.6 ± 0.6	76.9 ± 0.4	77.3 ± 0.5
GTAN (ours)	79.1 ± 0.4	79.3 ± 0.4	79.4 ± 0.2
GTCN (ours)	78.5 ± 0.5	78.6 ± 0.5	79.2 ± 0.3

Table 10: Model performance in accuracy with different depths, tested on the Coauthor-CS(3) dataset

Method (Coauthor-CS)	**Depth = 2**	**Depth = 5**	**Depth = 10**
GCN	89.7 ± 0.2	87.1 ± 0.7	82.9 ± 0.8
GAT	89.5 ± 0.4	85.9 ± 1.6	83.8 ± 0.6
Tree-LSTM	90.8 ± 0.2	91.0 ± 0.1	90.8 ± 0.2
GTAN (ours)	91.7 ± 0.2	91.7 ± 0.2	92.0 ± 0.3
GTCN (ours)	92.1 ± 0.2	92.0 ± 0.2	92.4 ± 0.2

For the DAGNN model, we use 16-hop with 256 hidden units. The dropout is 0.2. The learning rate and weight decay are 0.005 and 0, respectively.

For our GTAN model, we use 4 layers with 128 hidden units. The input dropout and the dropout after each propagation layer are 0.2 and 0, respectively. The learning rate and weight decay are 0.01 and 5e-5, respectively.

For our GTCN model, we use 5 layers with 256 hidden units. The input dropout and the dropout after each propagation layer are 0.2 and 0.2, respectively. The learning rate and weight decay are 0.01 and 5e-5, respectively.

A.3 TEST RESULTS ON MODEL PERFORMANCE IN MACRO-F1 SCORE

The test results of model performance in Macro-F1 score on the four DGL datasets are summarized in Table 6.

A.4 TEST RESULTS ON MODEL PERFORMANCES WITH DIFFERENT DEPTHS

The test results of model accuracy on the four DGL datasets with different number of propagation layers are summarized in Table 7, 8, 9, and 10, respectively.