Graph Transformer Networks: Learning Meta-path Graphs to Improve GNNs

Seongjun Yun, Minbyul Jeong, Sungdong Yoo, Seunghun Lee, Sean S. Yi, , Raehyun Kim
Jaewoo Kang, Hyunwoo J. Kim

Abstract—Graph Neural Networks (GNNs) have been widely applied to various fields due to their powerful representations of graph-structured data. Despite the success of GNNs, most existing GNNs are designed to learn node representations on the fixed and homogeneous graphs. The limitations especially become problematic when learning representations on a misspecified graph or a heterogeneous graph that consists of various types of nodes and edges. To address this limitations, we propose Graph Transformer Networks (GTNs) that are capable of generating new graph structures, which preclude noisy connections and include useful connections (e.g., meta-paths) for tasks, while learning effective node representations on the new graphs in an end-to-end fashion. We further propose enhanced version of GTNs, Fast Graph Transformer Networks (FastGTNs), that improve scalability of graph transformations. Compared to GTNs, FastGTNs are 230× faster and use 100× less memory while allowing the identical graph transformations as GTNs. In addition, we extend graph transformations to the semantic proximity of nodes allowing non-local operations beyond meta-paths. Extensive experiments on both homogeneous graphs and heterogeneous graphs show that GTNs and FastGTNs with non-local operations achieve the state-of-the-art performance for node classification tasks.

The code is available: https://github.com/seongjunyun/Graph_Transformer_Networks

Index Terms—Graph Neural Networks, Heterogeneous Graphs, Machine learning, Graphs and networks.

1 INTRODUCTION

Graph Neural Networks (GNNs) have become an increasingly popular tool to learn the representations of graph-structured data. They are widely used in a variety of tasks such as node classification [1], [2], [3], [4], [5], link prediction [6], [7], [8], [9], [10], graph classification [11], [12], [13], [14], and graph generation [15], [16], [17], [18], [19].

Despite their effectiveness to learn representations on graphs, most GNNs assume that the given graphs are fixed and homogeneous. Since the graph convolutions discussed above are determined by a fixed graph structure, a noisy graph with missing/spurious connections results in ineffective convolution with wrong neighbors on the graph. In addition, in some applications constructing a graph to operate GNNs is not trivial. For example, a citation network has multiple types of nodes (e.g., authors, papers, conferences) and edges defined by their relations (e.g., author-paper, paper-conference), which referred to as heterogeneous graphs. In heterogeneous graphs, the importance of each node type and edge type can vary depending on the task, and some node/edge types may even become completely useless. A naïve approach to deal with the heterogeneous graphs is to ignore the node/edge types and treat them as in a homogeneous graph (a standard graph with one type of nodes and edges). This, apparently, is suboptimal since models cannot exploit the type information. A more recent remedy is to manually design useful meta-paths, which are paths connected with heterogeneous edge types, and transform a heterogeneous graph into a homogeneous graph defined by the meta-paths. Then conventional GNNs can operate on the transformed homogeneous graphs [20], [21].

Despite improving on previous approaches, this is a two-stage approach and requires hand-crafted meta-paths for each problem. The accuracy of downstream analysis can be significantly affected by the choice of these meta-paths.

To address this limitation, we develop the Graph Transformer Networks (GTNs) that learn to transform an original graph into new graphs that preclude noisy connections and include useful multi-hop connections (e.g., meta-paths) for each task, and learn node representation on the new graphs in an end-to-end fashion. Specifically, the Graph Transformer layer, a core layer of GTN, learns a soft selection of adjacency matrices for edge types and multiply two selected adjacency matrices to generate useful meta-paths. Also, by leveraging an identity matrix, GTN can generate new graph structures based on arbitrary-length composite relations connected with softly chosen edge types in a heterogeneous graph.

Furthermore, we address the scalability issue of GTNs. To transform graphs, GTNs explicitly compute a new adjacency matrix of meta-paths by the matrix multiplications of huge adjacency matrices. This requires substantial computational costs and large memory making it infeasible to apply GTNs to a large graph. To address this issue, we propose an enhanced version of GTNs, Fast Graph Transformer Networks (FastGTNs), that implicitly transform the graphs without the multiplication of two adjacency matrices. Compared to GTNs, FastGTNs are 230× faster and use 100× less memory while allowing the identical graph transformations as GTNs.

Another issue of GTNs is its edge generation is limited to the nodes connected by a meta-path of the input graphs,
which do not take into account the semantic proximity of nodes. We further extend graph transformations to the semantic proximity of nodes allowing non-local operations beyond meta-paths.

The new graph structures from GTNs and FastGTNs lead to effective node representations resulting in state-of-the-art performance, without any predefined meta-paths from domain knowledge, on six benchmark classification on heterogeneous graphs. In addition, since GTNs and FastGTNs learn variable lengths of useful meta-paths (i.e., the neighborhood range of each node), GTNs and FastGTNs achieve state-of-the-art performance on six homogeneous graph datasets by adjusting neighborhood ranges for each dataset.

Our contributions are as follows:

(i) We propose a novel framework Graph Transformer Networks, to learn a new graph structure which involves identifying useful meta-paths and multi-hop connections for learning effective node representation on graphs.

(ii) We propose Fast Graph Transformer Networks (FastGTNs) that implicitly transform the graphs without the multiplication of adjacency matrices which requires excessive resources while allowing the identical transformations of GTNs.

(iii) We extend graph transformations to non-local operations incorporating the node features to utilize the semantic proximity of nodes beyond meta-paths.

(iv) We prove the effectiveness of node representation learnt by Graph Transformer Networks and FastGTNs with non-local operations resulting in the best performance against state-of-the-art GNN-baselines in six benchmark node classification on heterogeneous graphs and six benchmark node classification on homogeneous graphs. Also, our experiments demonstrate that on a large graph dataset FastGTNs show 230x faster inference time and 100x less memory usage than the GTNs.

2 RELATED WORKS

Recent years have witnessed significant development in deep learning architectures for graphs. [22] first proposed a convolution operation on graphs leveraging the Fourier transform and convolution kernels in a spectral domain. [2], [23] extended and improved spectral-based Graph Convolutional Networks (GCNs). On the other hand, spatial-based GNNs [8], [12], [24], [25], [26] have been proposed to improve the scalability of GNNs by performing graph convolution operations in the graph domain. [1] proposed the first-order approximation of the spectral filler using the Chebyshev polynomials. GCNs, the simplified spectral-based GNNs, can be viewed as spatial-based GNNs as well. Graph Attention Networks (GATs) [25] incorporate the attention mechanism into GNNs, which differentially aggregate the representations of neighbors on graphs using attention scores. GraphSAGE [24] expanded the operating range of GNNs for the inductive setting generating representations for unseen nodes by variable aggregation operations. Jumping Knowledge Networks (JKNets) [3] utilized flexible neighborhood ranges by adopting the skip-connection and Mixhop [26] leveraged a combination of powers of normalized adjacency matrices to aggregate features at various distances.

In contrast to the above works on homogeneous graphs, several studies [7], [20], [27] have attempted to extend GNN architectures to heterogeneous graphs that contain multiple types of nodes and edges. They are categorized into two approaches: GNNs with relation-specific parameters [7], [27] and GNNs with relation-based graph transformations [20]. Relational Graph Convolutional Networks (R-GCNs) [7] employed GCNs with relation-specific convolutions (or weight matrices) to deal with heterogeneous graphs. [27] proposed the Heterogeneous Graph Transformer (HGT) to parameterize the meta relation triplet of each edge type and used a structure that utilizes the self-attention of the transformer architecture [28] to learn specific patterns of different relationships. The second approaches, GNNs with relation-based graph transformations, generally utilize meta-paths. The Heterogeneous Graph Attention Network (HAN) [20] first transforms heterogeneous graphs into homogeneous graphs using manually selected meta-paths and applies an attention-based GNN on the graphs. However, the HAN has limitations that it is a multi-stage approach and requires the manual selection of meta-paths in each dataset. Also, performance can be significantly affected by the choice of meta-paths. Unlike this approach, our Graph Transformer Networks can operate on a heterogeneous graph and transform the graph for tasks while learning node representations on the transformed graphs in an end-to-end fashion.

3 METHOD

The goal of our framework, Graph Transformer Networks, is to generate new graph structures and learn node representations on the learned graphs simultaneously. Unlike most GNNs on graphs that assume the graph is given, GTNs seek for new graph structures using multiple candidate adjacency matrices to perform more effective graph convolutions and learn more powerful node representations. Learning new graph structures involves identifying useful meta-paths, which are paths connected with heterogeneous edges, and multi-hop connections. Before introducing our framework, we first briefly review the basic notions of heterogeneous graphs and GCN, and then introduce our Graph Transformer Networks.

3.1 Preliminaries

Heterogeneous Graph [29]. Let \( \mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{T}_e, \mathcal{T}_r) \) denote a directed graph where each node \( v \in \mathcal{V} \) and each edge \( e \in \mathcal{E} \) are associated with their type mapping functions \( \tau_e(v) : \mathcal{V} \rightarrow \mathcal{T}_e \) and \( \tau_e(e) : \mathcal{E} \rightarrow \mathcal{T}_r \), respectively. The heterogeneous graph \( \mathcal{G} \) can be represented by a set of adjacency matrices \( \{A_{t}^{l}\}_{l=1}^{\mathcal{T}_l} \) or a tensor (i.e., \( A_{t} \in \mathbb{R}^{\mathcal{V} \times \mathcal{V} \times \mathcal{T}_r} \)), where \( A_{t} \in \mathbb{R}^{\mathcal{V} \times \mathcal{V}} \) is an adjacency matrix of the \( t \)-th edge type and \( |\mathcal{V}| = N \). \( A_{t}[i,j] \) denotes the weight of the \( t \)-th type edge from node \( j \) to node \( i \). When a graph has a single type of nodes and edges, i.e., \( |\mathcal{T}_e| = 1 \) and \( |\mathcal{T}_r| = 1 \), it is called a homogeneous graph.

Meta-Path [20]. In heterogeneous graphs, a multi-hop connection is called a meta-path, which is a path connected with heterogeneous edge types, i.e., \( v_1 \xrightarrow{\tau_e(c_1)} v_2 \xrightarrow{\tau_e(c_2)} \).
Fig. 1. Graph Transformer Layer softly selects adjacency matrices (edge types) from the set of adjacency matrices \( A \) of a heterogeneous graph \( G \) and learns a new meta-path graph represented by \( A^{(k)} \) via the matrix multiplication of the output matrix of the previous \((k-1)\)-th GT Layer and the selected adjacency matrix \( F(A; \phi^{(k)}) \). The soft adjacency matrix selection is a weighted sum of candidate adjacency matrices obtained by \( 1 \times 1 \) convolution with non-negative weights from \( \text{softmax}(\alpha^{(k)}) \).

\[
\ldots \xrightarrow{\tau(e)\; v_t \rightarrow t_{t+1}} \text{softmax} 
\]

where \( \tau(e) \in T_e \) denotes the edge type of edge \( e \) on the meta-path.

**Graph Convolutional network (GCN).** In this work, a graph convolutional network (GCN) \([30]\) is used to learn useful representations for node classification in an end-to-end fashion. Let \( H^{(l)} \) be the feature representations of the \( l \)-th layer in GCNs, the forward propagation becomes

\[
H^{(l+1)} = \sigma\left( \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} H^{(l)} W^{(l)} \right),
\]

where \( \tilde{A} = A + I \in \mathbb{R}^{N \times N} \) is the adjacency matrix \( A \) of the graph \( G \) with added self-connections, \( \tilde{D} \) is the degree matrix of \( \tilde{A} \), i.e., \( \tilde{D}_{ii} = \sum_j \tilde{A}_{ij} \), and \( W^{(l)} \in \mathbb{R}^{d \times d} \) is a trainable weight matrix. One can easily observe that the convolution operation across the graph is determined by the given graph structure and it is not learnable except for the node-wise linear transform \( H^{(l)} W^{(l)} \). So the convolution layer can be interpreted as the composition of a fixed convolution followed by an activation function \( \sigma \) on the graph after a node-wise linear transformation. Since we learn graph structures, our framework benefits from the different convolutions, namely \( \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} \), obtained from learned multiple adjacency matrices. The architecture will be introduced later in this section. For a directed graph (i.e., asymmetric adjacency matrix), \( \tilde{A} \) in Eq. (1) can be normalized by the inverse of in-degree diagonal matrix \( D^{-1} \) as \( H^{(l+1)} = \sigma(\tilde{D}^{-1} \tilde{A} H^{(l)} W^{(l)}) \).

### 3.2 Learning Meta-Path Graphs

Previous works \([20], [21]\) require manually defined meta-paths and perform Graph Neural Networks on the meta-path graphs. Instead, our Graph Transformer Networks (GTNs) learn meta-path graphs for given data and tasks and operate graph convolution on the learned meta-path graphs. This gives a chance to find more useful meta-paths and lead to virtually various graph convolutions using multiple meta-path graphs.

The main idea of learning meta-path graphs is that a new adjacency matrix \( A_P \) of a useful meta-path \( P \) connected by a particular order of edge types (e.g., \( t_1, t_2 \ldots t_k \)) is obtained by the multiplications of adjacency matrices of the edge types as

\[
A_P = A_{t_k} \ldots A_{t_2} A_{t_1}.
\]

Based on this idea, each \( k \)-th Graph Transformer (GT) Layer in GTN learns to softly select adjacency matrices (edge types) by \( 1 \times 1 \) convolution with the weights from \( \text{softmax} \) function as

\[
F(A; \phi^{(k)}) = \text{conv}_{1 \times 1}(A; \text{softmax}(\phi^{(k)})) = \sum_{t=1}^{|T_e|} \alpha_t^{(k)} A_t,
\]

where \( \phi^{(k)} \in \mathbb{R}^{1 \times 1 \times |T_e|} \) is the parameter of \( 1 \times 1 \) convolution and \( \alpha_t^{(k)} = \text{softmax}(\phi^{(k)}) \) (i.e., the convex combination of adjacency matrices as \( \alpha_t^{(k)} A_t \)). Then the meta-path adjacency matrix is computed by matrix multiplication of an output and the output matrix of the previous \((k-1)\)-th GT Layer as \( A^{(k-1)} F(A; \phi^{(k)}) \). For numerical stability, the matrix is normalized by its degree matrix as

\[
A^{(k)} = (\tilde{D}^{(k)})^{-1} A^{(k-1)} F(A; \phi^{(k)}),
\]

where \( A^{(0)} = F(A; \phi^{(0)}) \) and \( \tilde{D}^{(k)} \) is a degree matrix of the output after the multiplication of two matrices \( A^{(k-1)} F(A; \phi^{(k)}) \).

Now, we need to check whether GTN can learn an arbitrary meta-path with respect to edge types and path length. The adjacency matrix of arbitrary length \( k+1 \) meta-paths can be calculated by

\[
A_P = \left( \sum_{t_0 \in T_e} \alpha_0^{(0)} A_{t_0} \right) \left( \sum_{t_1 \in T_e} \alpha_1^{(1)} A_{t_1} \right) \ldots \left( \sum_{t_k \in T_e} \alpha_k^{(k)} A_{t_k} \right)
\]

where \( A_P \) denotes the adjacency matrix of meta-paths, \( T_e \) denotes a set of edge types and \( \alpha_i^{(k)} \) is the weight for edge type \( t_k \) at the \( k \)-th GT layer. When \( \alpha \) is not one-hot vector,
We here introduce the architecture of Graph Transformer Networks. To consider multiple types of meta-paths simultaneously, the GTN generates multiple graph structures by setting the output channels of $A_t$ to $C$. Then the output matrix of $k$-th GT layer $A^k$ becomes the output tensor $A^k \in \mathbb{R}^{N \times N \times C}$ and the weight vector $\phi^k$ of the $k$-th GT Layer becomes the weight matrix $\Phi^k$. Eq. (5), can be represented in the form of tensor equations as

$$A^k = \left(\tilde{D}^{(k)}\right)^{-1}A^{(k-1)} * F(A^k; \Phi^k),$$

where $\tilde{D}^{(k)}$ is the degree tensor of the output after the multiplication of two tensors $A^{(k-1)} * F(A^k; \Phi^k)$. It is beneficial to learn different node representations via multiple different graph structures.

After the stack of $K$ GT Layers, multi-layer GNNs are applied to the each channel of the output tensor $A^{(K)}$ and the final node representations $Z$ can then be used for downstream tasks. For the node classification task, we applied dense layers followed by a softmax layer to the node representations. Then with ground truth labels of nodes, we can optimize the model weights by minimizing the cross-entropy via backpropagation and gradient descent.

### 3.4 Fast Graph Transformer Networks

In the previous sections, we demonstrated that GTNs can transform the original graphs into new meta-path graphs while learning representations on the meta-path graphs. However, GTNs have a scalability issue. GTNs explicitly compute a new adjacency matrix of meta-paths by the matrix multiplication of two adjacency matrices and store the new adjacency matrix at each layer. So a graph transformation in GTNs involves huge computational costs and large memory. This makes it infeasible to apply GTNs to a large graph. To address these issues, we develop the enhanced version of GTNs, FastGTNs, which implicitly transform the graph structures without storing the new adjacency matrices of.
where $\mathbf{A}$ matrices and normalizes the output adjacency matrix. So (9) $\mathbf{W} \in \mathbb{R}^{GTN}$ with node representations function is an identity function i.e., $f_{top}$ of the new graph structure and a channel aggregation channels is one, i.e., $\mathbf{C}$. matrices. For simplicity, we assume that the number of $\mathbf{GTNs}$. Our goal here is to derive a new architecture without detail. meta-paths. In this section, we describe our FastGTNs in detail.

To derive FastGTNs, we first begin with the equation of GTNs. Our goal here is to derive a new architecture without the need for the explicit multiplications of large adjacency matrices. For simplicity, we assume that the number of channels is one, i.e., $C=1$, one GCN Layer is applied on top of the new graph structure and a channel aggregation function is an identity function i.e., $f_{agg}(x) = x$. Then the node representations $Z$ of the GTNs are given as follows:

$$Z = \sigma(\hat{D}^{-1}(A^{(k)} + I)XW),$$

where $A^{(k)} \in \mathbb{R}^{N \times N}$ is a new adjacency matrix from a GTN with $K$ GT layers, and $X \in \mathbb{R}^{N \times F}$ is input features, $W \in \mathbb{R}^{F \times d'}$ is a linear transformation in a GCN layer, $\hat{D}^{-1} \in \mathbb{R}^{N \times N}$ is an inverse degree matrix of $(A^{(k)} + I)$. We observe that a GT Layer multiplies two softly selected adjacency matrices and normalizes the output adjacency matrix. So (9) can be written as

$$Z = \sigma(\hat{D}^{-1}XW + \hat{D}^{-1}A^{(K)}XW) = \sigma(\hat{D}^{-1}XW + \hat{D}^{-1}\left( (\hat{D}^{(K)})^{-1}A^{(K-1)}(\alpha^{(k)} \cdot \mathbf{A}) \right)XW) = \sigma(\hat{D}^{-1}XW + \hat{D}^{-1}\left( (\hat{D}^{(K)})^{-1} \ldots (\hat{D}^{(1)})^{-1} (\alpha^{(0)} \cdot \mathbf{A})(\alpha^{(1)} \cdot \mathbf{A}) \ldots (\alpha^{(K)} \cdot \mathbf{A}) \right)XW).$$

The Equation (10) clearly shows the computational bottleneck in (10) is the multiplications of huge adjacency matrices, e.g., $(\alpha^{(0)} \cdot \mathbf{A})(\alpha^{(1)} \cdot \mathbf{A}) \ldots (\alpha^{(K)} \cdot \mathbf{A})$. To resolve this problem, we can rewrite (10) using the associative property of matrix multiplication as

$$Z = \sigma(\hat{D}^{-1}XW + \hat{D}^{-1}\left( (\hat{D}^{(K)})^{-1} \ldots (\hat{D}^{(1)})^{-1} (\alpha^{(0)} \cdot \mathbf{A})(\alpha^{(1)} \cdot \mathbf{A}) \ldots (\alpha^{(K)} \cdot \mathbf{A})XW) \right)).$$

Now, Equation (11) implies that at each layer, without the matrix multiplications of huge adjacency matrices, the identical features can be obtained by a sequence of feature transformations using a differently constructed adjacency matrix, e.g., $\alpha^{(k)} \cdot \mathbf{A}$. It efficiently reduces the computational cost from $O(N^3)$ to $O(N^2 F)$ and memory usage from $O(N^2)$ to $O(N F)$. However, note that since we do not compute the multiplication of two adjacency matrices anymore, we cannot compute degree matrices $\hat{D}^{-1} \left( (\hat{D}^{(K)})^{-1} \ldots (\hat{D}^{(1)})^{-1} \right)$.

To address this challenge, now we show that given a condition of input data and a proposition of normalized matrices, we can make all degree matrices in (11) into identity matrices. Then we can compute (11) without the matrix multiplications of huge adjacency matrices. We begin with a proposition and a condition, then derive an equation of our FastGTNs from (11) by using the Proposition 1.

**Proposition 1.** Given two normalized adjacency matrices $A$, $B \in \mathbb{R}^{N \times N}$, the followings are equivalent:

(i) $D_A^{-1}A \left( D_B^{-1}B \right) = \left( D_{AB}^{-1}AB \right)$

(ii) $D_{AB} = I$

![Fig. 3. Fast Graph Transformer Networks (FastGTNs) implicitly transform graph structures by a sequence of feature transformations using differently constructed adjacency matrices from Fast Graph Transformer Layers (FastGT Layers). Each k-th FastGT Layer first generates a non-local adjacency matrix $A^{(k)}_{nonlocal}$ using the hidden representations $Z^{(k-1)}$ from (k-1)-th FastGT layer and appends it to the set of adjacency matrices $\mathbf{A}$. To generate diverse graphs, k-th FastGT layer generates $C^{(k)}$ new adjacency matrices $\hat{A}^{(k,l)} \in \mathbb{R}^{N \times N}$ by applying a 1 x 1 convolution filter. Using the $C^{(k)}$ new adjacency matrices in $A^{(k)}$, a FastGT layer transforms the hidden presentations $Z^{(k-1)}$ into $Z^{(k)}$. After $K$ FastGT layers, the final representations are obtained by channel aggregation after a convex combination of the expanded input representations $Z^{(0)}$ and the output representations of the $K$-th FastGT layer, i.e., $Z^{(k+1)} = f_{agg}(\gamma Z^{(0)} + (1 - \gamma) Z^{(k)}), \gamma \in (0, 1).$]
We finally rewrite our FastGTNs for multi-channel and multi-

The proof of Proposition 1 is provided in A.2 in the sup-

function. Furthermore, to deal with huge graphs with about

adjacency matrices, \( K \)-layers generate \( K \)-adjacency matrices

(iii) \( D_{A+I}^{-1} = (D_A + I)^{-1} = \frac{1}{2} I \)

The proof of Proposition 1 is provided in A.2 in the sup-

adjacency matrices at each \( k \)-th layer i.e., \( \alpha^{(k)} \cdot \mathbb{A} \) is also

where \( D \) is row-

of adjacency matrices at each \( k \)-th layer i.e., \( \alpha^{(k)} \cdot \mathbb{A} \) is also

\[ Z = \sigma(\tilde{D}^{-1} X W + D^{-1}(\alpha^{(0)} \cdot \mathbb{A}(\alpha^{(1)} \cdot \mathbb{A} \ldots (\alpha^{(K)} \cdot \mathbb{A}XW))). \] (12)

By (iii) in Proposition 1, we can also know that \( \tilde{D}^{-1} =

(\mathbb{D}^{A(K)} + I)^{-1} = \frac{1}{2} I \), then \( Z \) can be represented as

\[ Z = \sigma(\frac{1}{2} X W + \frac{1}{2}(\alpha^{(0)} \cdot \mathbb{A}(\alpha^{(1)} \cdot \mathbb{A} \ldots (\alpha^{(K)} \cdot \mathbb{A}XW))). \] (13)

Since each layer constructs one convex combination of

adjacency matrices, \( K \)-layers generate \( K \)-adjacency matrices as

\[ Z = \sigma(\frac{1}{2} X W + \frac{1}{2}(\alpha^{(1)} \cdot \mathbb{A}(\alpha^{(2)} \cdot \mathbb{A} \ldots (\alpha^{(K)} \cdot \mathbb{A}XW))). \] (14)

Now this derivation means that our FastGTNs are not an

approximation of GTNs. Mathematically, they are exactly

identical. We’ll discuss more in Section 4.3 about the identity.

We reverse the order of layers from 1 to K and replace \( \frac{1}{2} \) with

a hyper-parameter \( \gamma \), then the output of a FastGTN can be

represented as

\[ Z = \sigma(\gamma X W + (1 - \gamma)(\alpha^{(K)} \cdot \mathbb{A} \ldots (\alpha^{(1)} \cdot \mathbb{A}XW))). \] (15)

We finally rewrite our FastGTNs for multi-channel and multi-

layer settings as

\[ Z^{l+1}_{agg} = f_{agg}\left( \sum_{c=1}^{C^{(l)}} \sigma(\gamma Z^{(l)} W^{(l)}_c + (1 - \gamma) Z^{(l,K)}_c) \right), \] (16)

where \( C^{(l)} \) denotes the number of channels, \( Z^{(l)} \) denotes the node representations from the \( l \)-th FastGTN layer,

\[ Z^{(l,K)}_c = (\alpha^{(l,K,c)} \cdot \mathbb{A} \ldots (\alpha^{(l,1,c)} \cdot \mathbb{A} Z^{(l)} W^{(l)}_c)), \] (17)

\( W^{(l)}_c \in \mathbb{R}^{d^{(l)} \times d^{(l+1)}} \) is a linear transformation in \( c \)-th channel of the \( l \)-th FastGTN layer, \( \mathbb{A} \) is a set of normalized adjacency matrices, \( \alpha^{(l,k,c)} \) is a convolution filter in the \( c \)-th channel of the \( k \)-th FastGT layer in the \( l \)-th FastGTN layer, \( Z^{(0)} \) is a feature matrix \( X \in \mathbb{R}^{N \times F} \) and \( f_{agg} \) is a channel aggregation function. Furthermore, to deal with huge graphs with about 30 million edges, we additionally propose a mini-batch training algorithm for GTNs and FastGTNs in A.1 in the supplement.

3.5 Non-Local Operations.

One limitation of the GTNs is that its transformation is limited to compositions of existing relations. Specifically, K

GT layers can generate edges only up to (K+1)-hop relations.

It cannot generate remote relations based on semantic proximity between nodes. To address this limitation, we extend

graph transformations to non-local operations incorporating the node features to utilize the semantic proximity of nodes

beyond meta-paths. However, as mentioned in the previous section, since GTN itself requires large computation cost, we extend non-local operations only to FastGTNs. Specifically, at each \( k \)-th FastGT layer in each \( l \)-th FastGTN layer, we construct a non-local adjacency matrix \( A_{non\_local}^{(l,k)} \in \mathbb{R}^{N \times N} \) based on hidden representations \( Z^{(l,k-1)} \) from the previous FastGT layer and append the non-local adjacency matrix to the candidate set of adjacency matrices \( \mathbb{A} \) to utilize the non-local relations for graph transformations. To construct \( A_{non\_local}^{(l,k)} \) we first calculate a graph affinity matrix at each \( k \)-th FastGT Layer \( M^{(l,k)} \) based on the similarity between node features. We take an average of multi-channel hidden representations from each \( (k-1) \)-th FastGT Layer and project the averaged representations into a latent space by a non-linear transformation \( g_{th} \). Then we compute the affinity matrix \( M^{(l,k)} \) using the similarity in the latent space as

\[ M^{(l,k)} = (g_{th}(H^{(l,k-1)}) g_{th}(H^{(l,k-1)})^T), \] (18)

\[ H^{(l,k-1)} = \frac{1}{C^{(l)}} \sum_{c=1}^{C^{(l)}} Z^{(l,k-1)}_c, \] (19)

where \( Z^{(l,k-1)} \) denotes hidden representations at a \( (k-1) \)-th FastGT Layer. We use the trick of a decoder in GAE [6] as the similarity function to get the affinity matrix. The affinity matrix can be seen as a weighted adjacency matrix of the fully connected graph. If we include the dense affinity matrix as an adjacency matrix for graph transformation, it causes huge computation cost and may rather propagate irrelevant information between nodes. Therefore, we sparsify the affinity matrix by extracting only \( n \) largest weights for each node \( i \) and construct non-local adjacency matrix \( A_{non\_local}^{(l,k)} \) as

\[ A_{non\_local}^{(l,k)}[i,j] = \begin{cases} M^{(l,k)}_{ij}, & \text{if } j \in \arg \text{top} \ k(M^{(l,k)}[i,\cdot], n) \\ 0, & \text{otherwise} \end{cases} \] (20)

We row-wise normalize the non-local adjacency matrix by applying the softmax function to edge weights of each row. Then the final non-local adjacency matrix at each \( k \)-th FastGT layer is represented as

\[ A_{non\_local}^{(l,k)} = \text{softmax}(\text{topk}(M^{(l,k)}[i,\cdot], n)). \] (21)

To use the normalized non-local adjacency matrix for transformations, we add a non-local parameter to 1x1 convolution filters of each FastGT Layer and then append the matrix \( A_{non\_local}^{(l,k)} \) to the set of adjacency matrix of \( k \)-th FastGT Layer i.e., \( \mathbb{A} \).
3.6 Relations to Other GNN Architectures

FastGTNs enhance the scalability of GTNs by implicitly transforming graph structures. Moreover, the FastGTNs become a flexible/general model that subsumes other graph neural networks. In this section, we discuss relationships between our FastGTNs and other GNN architectures. Interestingly, if input graphs are normalized i.e., \( D = I \), several popular graph neural networks such as GCN [1] and MixHop [26] can be special cases of our FastGTNs. In addition, RGCN [7] can be subsumed by our FastGTNs with minor modifications. We first discuss the graph convolution network (GCN). The GCN computes the output node representations from the \( l \)-th GCN layer as

\[
Z^{(l+1)} = \sigma \left( \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} Z^{(l)} W^{(l)} \right) \quad (22)
\]

where \( \tilde{A} = A + I \in \mathbb{R}^{N \times N} \) and \( \tilde{D} \) is the degree matrix of \( \tilde{A} \). If the number of FastGT layers in our FastGTNs is one i.e., \( K = 1 \), the number of channels is one, i.e., \( C = 1 \) and \( \gamma \) equals \( \frac{1}{2} \), the output node representations are as

\[
Z^{(l+1)} = \sigma \left( \frac{1}{2} Z^{(l)} W^{(l)} + \frac{1}{2} A Z^{(l)} W^{(l)} \right). \quad (23)
\]

Then if the first FastGT layer only selects the adjacency matrix, i.e., \( \alpha^{(l,1)} \cdot \hat{A} = 1 \cdot A + 0 \cdot I \), the output representations are exactly same as the output of the GCNs.

Mixhop [26] is an extended GNN architecture which can capture long-range dependencies by mixing powers of the adjacency matrix as

\[
Z^{(l+1)} = \left\| \sigma \left( \hat{A}^{j} Z^{(l)} W^{(l)} \right) \right\|_{p} \quad (25)
\]

where \( P \) is a set of integer adjacency powers, \( \hat{A} \) is a symmetrically normalized adjacency matrix with self-connections, i.e., \( \hat{A} = \tilde{D}^{-\frac{1}{2}} (A + I) \tilde{D}^{-\frac{1}{2}} \) and \( \hat{A}^{j} \) denotes the adjacency matrix \( \hat{A} \) multiplied by itself \( j \) times. Since the degree matrix of \( A + I \) equals \( 2I \), we can rewrite equation (25) as

\[
Z^{(l+1)} = \left\| \sigma \left( \left( \alpha \cdot \hat{A} \right)^{j} Z^{(l)} W^{(l)} \right) \right\|_{p} \quad (26)
\]

where \( \alpha \cdot \hat{A} = \frac{1}{2} A + \frac{1}{2} I \). Then if \( \gamma \) equals 0, the number of channels equals the size of \( P \) i.e., \( C = |P| \), number of FastGT layers in each channel equals \( j \) i.e., \( K = j \), all FastGT layers choose the adjacency matrix and the identity matrix in the same ratio, i.e., \( \alpha^{(l,k)} \cdot \hat{A} = \frac{1}{2} A + \frac{1}{2} I \) and a channel aggregation function is an identity function i.e., \( f_{agg}(x) = x \), the output is same as the output of the Mixhop, i.e., the Mixhop can be a special case of our FastGTNs.

Lastly, we discuss RGCN [7] which extends the GCN to heterogeneous graphs by utilizing relation-specific parameters. Specifically, the output representations from the \( l \)-th RGCN are as

\[
Z^{(l+1)} = \sigma \left( \sum_{i=1}^{\tau} D_{i}^{-1} A_{i} Z^{(l)} \left( \sum_{i=1}^{B} a_{(l)}^{i} V^{(l)}_{i} \right) \right) \quad (27)
\]

\[
= \sigma \left( \sum_{i=1}^{B} a_{(l)}^{i} \cdot \hat{A} Z^{(l)} V^{(l)} \right), \quad (28)
\]

where \( A_{i} \) denotes an adjacency matrix for relation type \( t \), \( V^{(l)}_{i} \) denotes a basis parameter of \( l \)-th RGCN layer and \( a_{(l)}^{i} \) denotes a coefficient for relation type \( t \). The derivation from (27) to (28) is provided in A.3 in the supplement.

If the number of FastGT layers is one i.e., \( K = 1 \), the number of channels equals number of basis matrices i.e., \( C = B \), \( \gamma \) equals 0 and a channel aggregation function \( f_{agg} \) is summation, the output node representations are given as

\[
Z^{(l+1)} = \sigma \left( \sum_{c}^{B} a_{(l,1,c)} \cdot \hat{A} Z^{(l)} W^{(l)} \right). \quad (29)
\]

Then (28) and (29) are exactly same except for that (28) apply different linear combinations in each layer.

4 EXPERIMENTS

In this section, we evaluate our proposed methods on both homogeneous and heterogeneous graph datasets. The experiments aim to address the following research questions:

- **Q1.** How effective are the GTNs and FastGTNs with non-local operations compared to state-of-the-art GNNs on both homogeneous and heterogeneous graphs in node classification?
- **Q2.** Can the FastGTNs efficiently perform the identical graph transformation compared to the GTNs?
- **Q3.** Can GTNs adaptively produce a variable length of meta-paths depending on datasets?
- **Q4.** How can we interpret the importance of each meta-path from the adjacency matrix generated by GTNs?

4.1 Experimental Settings

**Datasets.** We evaluate our method on twelve benchmark datasets for node classification including six homogeneous graph datasets and six heterogeneous graph datasets for node classification. Detailed statistics regarding each dataset can be found in Table 1. The datasets for each type (i.e., homogeneous or heterogeneous) are as follows:

- **Heterogeneous Graph Datasets**
  - DBLP and ACM are both citation networks. They differ in the sense that DBLP has three types of nodes (paper (P), author (A), conference (C)) and four types of edges (PA, AP, PC, CP). ACM is similar but has has subject (S) as a node type instead of conference (C), with edge types differing accordingly. DBLP and ACM are node classification datasets with author research area and paper category as labels, respectively.
  - IMDB is a movie network dataset. It contains three types of nodes (movies (M), actors (A), directors (D)) and uses the genres of movies as labels.
  - CS, ML, NN each refer to domain-specific subgraphs from the Open Academic Graph (OAG) [31]. OAG is a large citation network with ten types of nodes (paper (P), author (A), field (L0, L1, L2, L3, L4, L5), venue (V), institute (I)) and ten types of edges (PA, PL0 − PL5, PV, AI, PP). The task is to predict the venue that each paper is published at.

- **Homogeneous Graph Datasets**
walk based baselines as well as state-of-the-art GNN based methods. Since homogeneous graph neural networks (e.g., GCN, GAT, JK-Net, MixHop and GCNII) cannot differentially handle the different types of nodes and edges, we apply them after converting the heterogeneous graphs into homogeneous graphs. In both dataset, each node represents a user of the online community and the edges correspond to whether or not users are following each other [33].

- **AIR-USA** is a dataset made up of graphs representing airport traffic within the US. Each node represents an airport and edges between nodes indicate the existence of commercial flights between the two airports [22].
- **BLOGCATALOG** and **FLICKR** are both social network datasets, with the former being a blogging platform and the latter being an image and video sharing platform. In both dataset, each node represents a user of the online community and the edges correspond to whether or not users are following each other [33].
- **CORA** and **CITESEER** are citation network datasets. Both datasets are comprised of nodes which represent papers published in various fields and edges which represent citation links [30].
- **PPI** refers to the protein-protein interaction network dataset. The network’s nodes represent a protein structure that contains features corresponding to different gene sets. Edges refer to the relation between such proteins [24].

**Baselines.** To evaluate the effectiveness of representations learnt by the Graph Transformer Networks in node classification, we compare GTNs with conventional random walk based baselines as well as state-of-the-art GNN based methods. Since homogeneous graph neural networks (e.g., GCN, GAT, JK-Net, MixHop and GCNII) cannot differentially handle the different types of nodes and edges, we apply them after converting the heterogeneous graphs into homogeneous graphs.

- **MLP** is a simple baseline model that uses only node features for prediction.
- **Node2Vec** [34] is a random walk based network embedding method which was originally designed for embedding homogeneous graphs. In heterogeneous graphs, we ignore the heterogeneity of nodes and edges and run DeepWalk on the entire heterogeneous graph.
- **GCN** [1] utilizes a first-order approximation of the spectral graph filter to aggregate features from neighbors.
- **GAT** [25] leverages an attention mechanism to learn the relative weights between the neighborhood nodes.
- **JK-Net** [3] leverages a variable range of neighborhoods by connecting the last layer of the network with all preceding hidden layers.
- **MixHop** [26] mixes powers of the adjacency matrices and applies a GCN to capture long-range dependencies.
- **GCNII** [35] improves GCN with initial residual connection and identity mapping to prevent over-smoothing.
- **RGCN** [7] employs GCNs with relation-specific weight matrices to deal with heterogeneous graphs.
- **HAN** [20] uses manually selected meta-paths to transform a heterogeneous graph into a homogeneous graph and then applies GNNs on the homogeneous graph.
- **HGT** [27] parameterizes the meta relation triplet of each edge type and uses a structure that utilizes the self-attention of the transformer architecture [28] to learn specific patterns of different relationships.

**Implementation details.** All the models in this paper are implemented using PyTorch and PyTorch Geometric [36] and the experiments are conducted on a single GPU (Quadro RTX 8000). For Node2Vec, GCN, GAT, JK-Net, GCNII, and RGCN, we used the implementations in PyTorch Geometric. We reimplemented HAN and HGT referencing the code from the authors of the papers [20, 27]. We set the dimensionality of hidden representations to 64 throughout the neural networks and apply the Adam optimizer for all models. For each model and each dataset, we perform a hyper-parameter search within the following ranges: the learning rate is from 1e-3 to 1e-6, the dropout rate is from 0.1 to 0.8 and the epoch is from 50 to 200. Based on the accuracy on validation sets, the best models are selected and the models are used for evaluation. From ten independent runs, the mean and standard deviation of micro-F1 scores on test datasets are computed. In our FastGTNs, as discussed in Proposition 1, we perform the row-wise normalization of the adjacency matrices in $A$. To avoid the division-by-zero, we add a small positive number to the diagonal elements of the adjacency matrices, i.e., $D^{-1}(A + \epsilon I)$.

### 4.2 Results on Node Classification

We evaluated the effectiveness of our GTNs and FastGTNs with non-local operations in six heterogeneous graph datasets and six homogeneous graph datasets. By analysing the result of our experiment, we will answer the research question Q1. Effectiveness of Graph Transformer Networks on heterogeneous graph datasets. Table 2 and 3 show the classification results on six heterogeneous graph datasets. In large-scale graph datasets (e.g., CS, ML, NN, DBLP, BLOGCATALOG, and FLICKR), we trained GNN-based methods...
in homogeneous graphs. We additionally compared our methods with three well-known GNN models MixHop [26], JK-Net [3] and GCNII [35]. We can observe that our GTN and FastGTN consistently outperform all GNN baselines in homogeneous graph datasets, especially on BLOGCATELOG and FLICKR datasets. Interestingly, in BLOGCATELOG and FLICKR datasets, the Multi-Layer Perceptron (MLP) model using the only node features achieved better performance than all GNN baseline models. It implies that noisy input graphs rather hinder learning of most GNNs whereas GTNs and FastGTNs successfully suppress the noisy edges by weighing an attention score of an identity matrix and learn a high accuracy classifier.

**Identity matrix in A to learn variable-length meta-paths.** As mentioned in Section 3.2, the identity matrix is included in the candidate adjacency matrices A. To verify the effect of identity matrix, we trained and evaluated another model named GTN$_{-I}$ as an ablation study. The GTN$_{-I}$ has exactly the same model structure as GTN but its candidate adjacency matrix A doesn’t include an identity matrix. In general, the GTN$_{-I}$ usually performs worse than the GTN. In heterogeneous graph datasets, it is worth to note that the difference is greater in IMDB than DBLP. One explanation is that the length of meta-paths GTN$_{-I}$ produced is not effective in IMDB. As we stacked 3 layers of GTL, GTN$_{-I}$ always produce 4-length meta-paths. However shorter meta-paths (e.g. MDM) are preferable in IMDB. Also, in homogeneous graph datasets, the differences in BLOGCATELOG and FLICKR datasets, the Multi-Layer Perceptron (MLP) model using the only node features achieved better performance than all GNN baseline models. It implies that noisy input graphs rather hinder learning of most GNNs whereas GTNs and FastGTNs successfully suppress the noisy edges by weighing an attention score of an identity matrix and learn a high accuracy classifier.

### 4.3 Exactness and Efficiency of FastGTNs

In this section, we show the exactness and efficiency of FastGTN compared to GTN. First, As discussed in [11] in Section 3.3, our FastGTN without non-local operations is an exact version of GTN. It means that the model parameters from GTN are compatible with FastGTN and the graph transformations of GTN can be identically performed by FastGTN. To show the exactness of FastGTN, we first train a GTN on the IMDB dataset and copy the model parameters of the GTN $(\{\Phi^{(k)}_l\}_{k=1}^{K})$ and the GNN $(\{W^{(i)}_l\}_{l=1}^{L})$ to the

| Model | DBLP | ACM | IMDB |
|-------|------|-----|------|
| MLP   | 79.18±0.015 | 86.19±0.003 | 49.51±0.019 |
| Node2Vec | 86.10±0.001 | 76.27±0.002 | 47.32±0.005 |
| GCN   | 85.63±0.003 | 91.70±0.003 | 60.41±0.009 |
| GAT   | 94.68±0.002 | 91.99±0.003 | 59.64±0.016 |
| RGCN  | 91.16±0.002 | 91.93±0.004 | 59.87±0.008 |
| HAN   | 92.17±0.005 | 91.10±0.004 | 59.80±0.013 |
| HGT   | 94.21±0.005 | 91.14±0.005 | 60.98±0.002 |
| GTN$_{-I}$ | 94.74±0.005 | 85.36±0.021 | 59.27±0.021 |
| GTN   | 94.47±0.003 | 91.96±0.005 | 61.02±0.018 |
| FastGTN | 94.85±0.003 | 92.51±0.005 | 64.63±0.008 |

**Table 3**

Node classification (micro F1-score) on homogeneous graph datasets.

and GTN in the mini-batch setting with graph sampling algorithm [24], [27]. We observe that our proposed methods, GTN and FastGTN, consistently outperform all network embedding methods and graph neural network methods in six heterogeneous graph datasets. GNN-based methods perform better than random walk-based network embedding methods. Interestingly, though the HAN is a modified GAT for a heterogeneous graph, the GAT usually performs better than the HAN. This result shows that using the pre-defined meta-paths as the HAN may cause adverse effects on performance. In contrast, Our GTN and FastGTN achieved the best performance compared to all other baselines on all the datasets. It demonstrates that the GTN can learn a new graph structure which consists of useful meta-paths for learning more effective node representation. Also, the performance gap between GTNs and FastGTNs on IMDB (60.02% vs. 64.64%) shows that in FastGTNs the graph transformations based on the semantic similarity (i.e., non-local operations) are effective. We additionally provide an ablation study of non-local operations in A.6 in the supplement.

**Effectiveness of Graph Transformer Networks on homogeneous graphs.** In homogeneous graphs, although a number of edge types is only one, as we add an identity matrix to the candidate adjacency matrix, our GTNs can find the effective neighborhood range for each dataset. Table 3 shows the performance of GTNs, FastGTNs and other baselines.
we prove that predictions (confidence scores) by special cases proving the efficiency of graph transformations. For more popular graph neural networks such as GCN and MixHop, the supplement). Note that the GTN parameters \( \{ \Phi^{(k)} \}_{k=1}^{K} \) should be reversely copied as in (15). As our FastGTNs also generalize of other pre-defined meta-paths between target nodes (a type of nodes with labels for node classifications). Also, new relevant meta-paths between all types of nodes are discovered by GTNs.

![Comparison with predefined meta-paths and top-ranked meta-paths by GTNs. Our model found important meta-paths that are consistent with pre-defined meta-paths between target nodes (a type of nodes with labels for node classifications). Also, new relevant meta-paths between all types of nodes are discovered by GTNs.](image)

**Table 4**

| Dataset | Predefined Meta-path | Top 3 (between target nodes) | Top 3 (all) |
|---------|----------------------|-----------------------------|-------------|
| DBLP    | ACPA, APA            | ACPA, APAPA, APA            | CPCPA, ACPA, CP |
| ACM     | PAP, PSP             | PAP, PSP                    | APAP, APA, SPAP |
| IMDB    | MAM, MDM             | MDM, MAM, MDMMDM            | DM, AM, MDM |

**Table 5**

Comparison with predefined meta-paths and top-ranked meta-paths by GTNs. Our model found important meta-paths that are consistent with pre-defined meta-paths between target nodes (a type of nodes with labels for node classifications). Also, new relevant meta-paths between all types of nodes are discovered by GTNs.

detailed efficiency comparisons with GTN, we measured the inference time and memory consumption of the two methods. Figure 5 shows that FastGTN is significantly more efficient than GTN in both the inference time and memory consumption. The performance gain is larger on larger graphs. In particular, on the PPI dataset, our experiments show that our FastGTN is \( 230 \times \) faster and \( 100 \times \) more memory-efficient than the GTN. Again, this speed-up and memory efficiency are achieved without any accuracy loss. During training, a similar performance gain is observed (see A.4 in the supplement).

### 4.4 Interpretation of Graph Transformer Networks

We examine the transformation learnt by GTNs to discuss the question interpretability Q4. We first describe how to calculate the importance of each meta-path from our GT layers. For the simplicity, we assume the number of output channels is one. Then, the new adjacency matrix from the \( l \)th GT layer can be written as

\[
A^{(k)} = \left( D^{(k-1)} \right)^{-1} \cdots \left( D^{(1)} \right)^{-1} \left( \alpha_0^{(0)} \cdot A_0 \right) \left( \alpha_1^{(1)} \cdot A_1 \right) \cdots \left( \alpha_k^{(k)} \cdot A_k \right) = \left( D^{(k-1)} \right)^{-1} \cdots \left( D^{(1)} \right)^{-1} \sum_{i_0, i_1, \ldots, i_k \in T} \alpha_0^{i_0} \alpha_1^{i_1} \cdots \alpha_k^{i_k} A_{i_0} A_{i_1} \cdots A_{i_k},
\]

(30)
where $\mathcal{T}_c$ denotes a set of edge types and $\alpha^{(l)}_{t_i}$ is an attention score for edge type $t_i$ at the $l$th GT layer. So, $A^{(l)}$ can be viewed as a weighted sum of all meta-paths including 1-length (original edges) to $l$-length meta-paths. The contribution of a meta-path $t_i, t_{i-1}, \ldots, t_0$ is obtained by $\prod_{i=0}^{l} \alpha^{(i)}_{t_i}$.

Now we can interpret new graph structures learnt by GTNs. The weight $\prod_{i=0}^{l} \alpha^{(i)}_{t_i}$ for a meta-path $(t_0, t_1, \ldots, t_l)$ is an attention score and it provides the importance of the meta-path in the prediction task. In Table 5, we summarized predefined meta-paths, that are widely used in literature, and the meta-paths with high attention scores learnt by GTNs.

As shown in Table 5, between target nodes, that have class labels to predict, the predefined meta-paths by domain knowledge are consistently top-ranked by GTNs as well. This shows that GTNs are capable of learning the importance of meta-paths for tasks. More interestingly, GTNs discovered important meta-paths that are not in the predefined meta-path set. For example, in the DBLP dataset GTN ranks CPCPA as most importance meta-paths, which is not included in the predefined meta-path set. It makes sense that author’s research area (label to predict) is relevant to the venues where the author publishes. We believe that the interpretability of GTNs provides useful insight in node classification by the attention scores on meta-paths.

Fig. 6 shows the attention scores of adjacency matrices (edge type) from each Graph Transformer Layer. Compared to the result of DBLP, identity matrices have higher attention scores in IMDB. As discussed in Section 3.3, a GTN is capable of learning shorter meta-paths than the number of GT layers, which they are more effective as in IMDB. By assigning higher attention scores to the identity matrix, the GTN tries to
We proposed Graph Transformer Networks for learning node representations on both homogeneous graphs and heterogeneous graphs. Our approach transforms graphs into multiple new graphs defined by meta-paths with arbitrary edge types and arbitrary length up to one less than the number of Graph Transformer layers while it learns node representation via convolution on the learnt meta-path graphs. Also, we proposed the enhanced version of GTNs, Fast Graph Transformer Networks, which are 230× faster and use 100× less memory while allowing the identical graph transformations as GTNs. The learnt graph structures from GTNs and FastGTNs lead to more effective node representation resulting in state-of-the-art performance, without any predefined meta-paths from domain knowledge, on all twelve benchmark node classification on both homogeneous and heterogeneous graphs. Since our Graph Transformer layers can be combined with existing GNNs, we believe that our framework opens up new ways for GNNs to optimize graph structures by themselves to operate convolution depending on data and tasks without any manual efforts. As several heterogeneous graph datasets have been studied for other network analysis tasks, such as link prediction [37], [38] and graph classification [39], [40], applying our GTNs to the other tasks can be interesting future directions.

5 Conclusion

We proposed Graph Transformer Networks for learning node representations on both homogeneous graphs and heterogeneous graphs. Our approach transforms graphs into multiple new graphs defined by meta-paths with arbitrary edge types and arbitrary length up to one less than the number of Graph Transformer layers while it learns node representation via convolution on the learnt meta-path graphs. Also, we proposed the enhanced version of GTNs, Fast Graph Transformer Networks, which are 230× faster and use 100× less memory while allowing the identical graph transformations as GTNs. The learnt graph structures from GTNs and FastGTNs lead to more effective node representation resulting in state-of-the-art performance, without any predefined meta-paths from domain knowledge, on all twelve benchmark node classification on both homogeneous and heterogeneous graphs. Since our Graph Transformer layers can be combined with existing GNNs, we believe that our framework opens up new ways for GNNs to optimize graph structures by themselves to operate convolution depending on data and tasks without any manual efforts. As several heterogeneous graph datasets have been studied for other network analysis tasks, such as link prediction [37], [38] and graph classification [39], [40], applying our GTNs to the other tasks can be interesting future directions.

References

[1] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *International Conference on Learning Representations*, 2017.

[2] Qimai Li, Zhichao Han, and Xiao ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning, 2018.

[3] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Kenichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. *International Conference on Machine Learning*, 2018.

[4] Ziqi Liu, Chaochao Chen, Longfei Li, Jun Zhou, Xiaolong Li, Le Song, and Yuan Qi. Geniepath: Graph neural networks with adaptive receptive paths. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2019.

[5] 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*, 2020.

[6] Thomas N Kipf and Max Welling. Variational graph auto-encoders. *NIPS Workshop on Bayesian Deep Learning*, 2016.

[7] 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*, Springer, 2018.

[8] Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. In *Advances in Neural Information Processing Systems*, 2018.

[9] Komal K Teru, Etienne Denis, and William L Hamilton. Inductive relation prediction by subgraph reasoning. In *Proceedings of the International Conference on Machine Learning*, 2020.

[10] Shikhar Vashishth, Soumya Sanyal, Vikram Nitin, and Partha Talukdar. Composition-based multi-relational graph convolutional networks. In *International Conference on Learning Representations*, 2020.
Seongjun Yun received the bachelor’s degree from Computer Science and Engineering Department, Korea University, South Korea, in 2018. He is working toward the doctoral degree at Korea University. His current research interest includes graph neural networks, recommendation systems, and machine learning. He was a recipient of the Naver PhD Fellowship award in 2020.

Minbyul Jeong received the BS degree in Computer Science and Engineering Department from Korea University, Seoul, Republic of Korea. His current research interests include natural language processing for real world problems and graph neural network.

Sungdong Yoo received the BS degree in Computer Science and Engineering Department, Korea University, in 2019. He is working toward the Master degree with Korea Univ. His research interest includes graph neural network and reinforcement learning. He was a research intern with LG CNS, Seoul, in 2019.

Seunghun Lee received his B.S. degree from the Korea University in 2020, and he is currently working toward the MS degree in the Department of Computer Science and Engineering. His research interests are in neural networks, especially in graph neural networks and semi-supervised learning.

Sean S. Yi received his bachelor’s degree from Korea University in 2019, and is currently pursuing a master’s degree from the same institution under the supervision of Professor Jaewoo Kang. His research interests largely lie in natural language processing, graphical models, and the interplay between the two. Specifically, he is interested in how to utilize graphical models in order to effectively model representations of human language and extract information.

Seunghun Lee received his B.S. degree in Computer Science from Korea University in 1994, M.S. in Computer Science from the University of Colorado at Boulder in 1996, and Ph.D. in Computer Science from the University of Wisconsin–Madison in 2003. From 1996 to 1997, he was a member of the technical staff at AT&T Labs Research, Florham Park, NJ. From 2000 to 2001, he was CTO and Co-Founder of WISEngine Inc., Santa Clara, CA, and Seoul, Korea. From 2003 to 2006, he was an assistant professor in the computer science department at North Carolina State University. Since 2006, he has been a professor of Computer Science at Korea University. He is also head of the department of the interdisciplinary graduate program in Bioinformatics at Korea University. He is jointly appointed professor in the department of Medicine at Korea University.

Hyunwoo J. Kim is an assistant professor in the Department of Computer Science and Engineering at Korea University. Prior to the position, he worked at Amazon Lab126 in Sunnyvale California. He earned his Ph.D. in computer science in 2017 from the University of Wisconsin-Madison and his Ph.D. minor is statistics. His research interests include geometric deep learning for graphs and manifolds, manifold statistics, machine learning, computer vision, and medical imaging.

Jaewoo Kang Jaewoo Kang (Member, IEEE) received B.S. degree in Computer Science from Korea University in 1994, M.S. in Computer Science from the University of Colorado at Boulder in 1996, and Ph.D. in Computer Science from the University of Wisconsin–Madison in 2003. From 1996 to 1997, he was a member of the technical staff at AT&T Labs Research, Florham Park, NJ. From 2000 to 2001, he was CTO and Co-Founder of WISEngine Inc., Santa Clara, CA, and Seoul, Korea. From 2003 to 2006, he was an assistant professor in the computer science department at North Carolina State University. Since 2006, he has been a professor of Computer Science at Korea University. He is also head of the department of the interdisciplinary graduate program in Bioinformatics at Korea University. He is jointly appointed professor in the department of Medicine at Korea University.
APPENDIX A

A.1 Mini-batch training for GTNs

Since GTNs transform the entire input graph at once, when the size of an input graph is too large, GTNs requires an excessive amount of memory and incurs high computational cost. To alleviate this scalability issue, we present a mini-batch training algorithm for GTNs. Algorithm 1 describes our mini-batch training algorithm. Specifically, we first combine a set of input adjacency matrices \( \{A_t\}_{t=1}^{T_a} \) into one adjacency matrix \( A \) for extracting subgraphs. We then select target nodes for each mini-batch training iteration. For each iteration, based on the target nodes of each mini-batch, we extract a subgraph and use its adjacency matrix \( A' \) from the original graph’s adjacency matrix \( A \) by using two types of graph sampling algorithms: a neighborhood-based graph sampling algorithm [24] and a layer-wise graph sampling algorithm [27]. After sampling the subgraph, we separate the sampled subgraph’s adjacency matrix \( A' \) into a set of adjacency matrices corresponding to each edge type such as \( \{A'_{t}\}_{t=1}^{T_a} \). The sampled adjacency matrix set \( \{A'_{t}\}_{t=1}^{T_a} \) is fed into GTNs for mini-batch training. This mini-batch training algorithm enables GTNs to handle large-scale graph datasets with up to 30 million edges in an efficient manner.

Algorithm 1: Mini-batch training algorithm for Graph Transformer Networks

**Input:** set of adjacency matrices \( A \); feature matrix \( X \); training set \( V_{\text{train}} \), and \( Y_{\text{train}} \); Graph Transformer Networks \( f_B \); number of layers \( L \).

**Output:** set of adjacency matrices \( A' \)

1. Combine a set of input adjacency matrices into one adjacency matrix \( A \leftarrow \bigcup_{t=1}^{T_a} A_t \);
2. while do
   3. Sample a mini-batch of \( m \) target nodes \( \{v_i\}_{i=1}^{m} \) from the training set \( V_{\text{train}} \) with corresponding targets \( Y_B \);
   4. \( V_{B} \leftarrow \{v_i\}_{i=1}^{m} \);
   5. \( E_B \leftarrow \emptyset \);
   6. for \( l = 1, 2, \ldots, L + 1 \) do
      7. Sample nodes \( V^{(l)} \) and edges \( E^{(l)} \) based on \( V_B \) by using graph sampling algorithm [24, 27];
      8. \( V_B \leftarrow V_B \cup V^{(l)} \);
      9. \( E_B \leftarrow E_B \cup E^{(l)} \);
   10. Reconstruct an adjacency matrix \( A' \) based on sampled nodes \( V_B \) and edges \( E_B \);
   11. Divide the adjacency matrix \( A' \) into a set of adjacency matrices corresponding to each edge type \( \{A'_{t}\}_{t=1}^{T_a} \);
   12. Compute prediction \( Y' \) from \( f_B \), \( X \), and \( A' \) by using Eq. (8) and the node classifier;
   13. Calculate cross-entropy loss from \( Y' \) and \( Y_B \);
   14. Update weights of \( f_B \);
3. end do

A.2 Proofs of the Proposition 1

**Proposition 1.** Given two normalized adjacency matrices \( A \), \( B \in \mathbb{R}^{N \times N} \), the followings are equivalent:

i) \( (D^{-1}_A A^{-1} B) = (D^{-1}_{AB} AB) \)

ii) \( D^{-1}_{AB} = I \)

iii) \( D^{-1}_{A+1} = (D_A + I)^{-1} = \frac{1}{2} I \)

Since the adjacency matrices \( A \), \( B \) are normalized, i.e., \( \sum_j A_{ij} = 1 \), the degree matrices \( D_A, D_B \) are equal to the identity matrix \( I \) and the inverse degree matrices \( D_A^{-1}, D_B^{-1} \) are also equal to the identity matrix \( I \) respectively. Thus, (i) in Proposition 1 can be re-written as \( AB = D^{-1}_{AB} AB \) and to satisfy the (i), we need to prove that \( D_{AB} \) is an identity matrix. Since \( D_{AB} \) is the degree matrix of the multiplication of two matrices \( A \) and \( B \), each \( i \)-th diagonal element of \( D_{AB} \) can be represented as \( D_{AB}[i, i] = \sum_j (AB)_{ij} = \sum_j \sum_k A_{ik} B_{kj} \). Then, we can derive that \( D_{AB} \) is equal to the identity matrix, i.e., \( D_{AB}[i, i] = 1 \) as follows:

\[
D_{AB}[i, i] = \sum_j (AB)_{ij} = \sum_j \sum_k A_{ik} B_{kj} = \sum_k A_{ik} \sum_j B_{kj} = \sum_k A_{ik} = 1
\]

Therefore, degree matrix \( D_{AB} \) is equal to an identity matrix \( I \), which satisfies (i), (ii) and (iii) in Proposition 1.

A.3 Relation to RGCN

If input graphs are normalized, RGCN [7] can be subsumed by our FastGTNs with minor modifications. The RGCN [7] extends the GCN to heterogeneous graphs by utilizing relation-specific parameters. Specifically, the output representations from the \( l \)-th RGCN are as

\[
Z^{(l+1)} = \sigma \left( \sum_{t=1}^{T_c} D^{-1}_t A_t Z^{(l)} W_t^{(l)} \right), \tag{31}
\]

where \( A_t \) denotes an adjacency matrix of \( t \)-th edge type \( W_t^{(l)} \) denotes the relation specific parameters of the model. The RGCN also address overparameterization by proposing basis decomposition of \( W_t^{(l)} \) as \( W_t^{(l)} = \sum_{b} a_{tb}^{(l)} Y_b^{(l)} \), consequently the equation of the RGCN is re-written as

\[
Z^{(l+1)} = \sigma \left( \sum_{t=1}^{T_c} D^{-1}_t A_t Z^{(l)} \left( \sum_{i=1}^{B} a_{ti}^{(l)} Y_i^{(l)} \right) \right), \tag{32}
\]
Fig. 7. Comparisons of training time (up) and memory usage (down) between GTNs (blue) and FastGTNs (orange) on both homogeneous and heterogeneous graph datasets (x-axis). For fair comparison, FastGTNs were measured without the non-local operations. FastGTNs significantly speed up the graph transformations 150 \times (PPI) and reduce the memory usage 60 \times (PPI). Then to compare with our FastGTNs, we derive the equation similar to our FastGTNs as follows:

\[ Z^{(l+1)} = \sigma \left( \sum_{t=1}^{B} A_t Z^{(l)} \left( \sum_{i=1}^{B} a_{ti}^{(l)} V_i^{(l)} \right) \right) \]  

(33)

\[ = \sigma \left( \sum_{t=1}^{B} \sum_{i=1}^{B} A_t Z^{(l)} a_{ti}^{(l)} V_i^{(l)} \right) \]  

(34)

\[ = \sigma \left( \sum_{i=1}^{B} \sum_{t=1}^{B} A_t Z^{(l)} a_{ti}^{(l)} V_i^{(l)} \right) \]  

(35)

\[ = \sigma \left( \sum_{i=1}^{B} \sum_{t=1}^{B} A_t Z^{(l)} a_{ti}^{(l)} V_i^{(l)} \right) \]  

(36)

\[ = \sigma \left( \sum_{i=1}^{B} Z^{(l)} \sum_{t=1}^{B} a_{ti}^{(l)} V_i^{(l)} \right) \]  

(37)

\[ = \sigma \left( \sum_{i=1}^{B} a_{ti}^{(l)} A_t Z^{(l)} V_i^{(l)} \right) \]  

(38)

\[ = \sigma \left( \sum_{i=1}^{B} a_{ti}^{(l)} \cdot A Z^{(l)} V_i^{(l)} \right) \]  

(39)

A.4 Training Efficiency

In this section, we compared our FastGTNs with GTNs in terms of the training efficiency. As shown in Figure 7, we measured the training time and the memory consumption of the two methods. Figure 7 shows that FastGTNs are significantly more efficient than GTNs in terms of both the training time and memory. In particular, the comparison on the large-scale dataset, PPI, shows our FastGTNs are 150 \times faster and 60 \times more efficient than the GTNs.

A.5 Generalization of GCN and MixHop

As discussed in Section 3.6, our FastGTNs subsume two popular graph neural networks, GCN and MixHop. For the GCN, if the number of FastGT layers in FastGTNs is one i.e., \( K = 1 \), the number of channels is one, i.e., \( C = 1 \), \( \gamma \) equals \( \frac{1}{2} \) and the first FastGT layer only selects the adjacency matrix, i.e., \( \alpha^{(1)} \cdot A = 1 \cdot A + 0 \cdot I \), the GCN can be a special case.
of FastGTNs. For the MixHop, if $\gamma$ equals 0, the number of channels equals the size of $P$, i.e., $C = |P|$. number of FastGT layers in each channel equals $j$ and all FastGT layers choose the adjacency matrix and the identity matrix in the same ratio, i.e., $\alpha(0) \triangleq \frac{1}{2} A + \frac{1}{2} I$, the output is same as the output of the Mixhop, i.e., the Mixhop can be a special case of FastGTNs. To show that special cases of FastGTNs can be identifiedly performed by the GCN and the MixHop, we first train a GCN and a MixHop on the CORA dataset and copy the model parameters of GCN$\{W^{(j)} \} _{j=1}^L$ and MixHop$\{W^{(j)} \} _{j=1}^L$ to the corresponding model parameters in our FastGTN. Figure 8 and Figure 8 prove that the predictions (confidence scores) by the FastGTN and other graph neural networks, GCN and MixHop, are identical. All 50 randomly drawn data points from a test set are on the Identity line (i.e., $y = x$).

### A.6 Ablation Study for non-local Operations

We evaluate the effectiveness of non-local operations in FastGTNs. Table 5 shows the performance gap of FastGTNs with/without non-local operations. Also, the attention scores on the non-local adjacency matrix at each layer of FastGTNs are reported to show whether the non-local operations are adaptively applied. First, non-local operations improve the performance in most datasets (7 out of 9) by $0.39 \sim 4.4$ in terms of the micro-F1 score. In addition, Table 5 shows that if the non-local operations are useful then FastGTNs have relatively higher attention scores on non-local operations. In contrast, in BLOGCATALOG, FLICKR datasets, FastGTNs where non-local operations are not useful, the FastGTNs (without non-local operations) properly reduced the attention scores on non-local operations to $0.004 \sim 0.012$. The results show that FastGTNs can adaptively exploit non-local operations depending on datasets.

| Dataset | w/o non-local | w/ non-local | Attention scores on non-local op. |
|---------|--------------|--------------|-----------------------------------|
|         | L1           | L2           |                                   |
| BLOGC   | 88.96        | 87.97 (↓ 0.99) | 0.012 | 0.005 |
| FLICKR  | 75.01        | 73.64 (↓ 1.37) | 0.004 | 0.005 |
| AIR-USA | 56.60        | 57.73 (↑ 1.13) | 0.153 | 0.158 |
| CITESEER| 68.32        | 69.14 (↑ 0.82) | 0.099 | 0.095 |
| CORA    | 79.17        | 80.29 (↑ 1.12) | 0.146 | 0.152 |
| PPI     | 40.95        | 42.40 (↑ 1.45) | 0.243 | 0.223 |
| DBLP    | 94.46        | 94.85 (↑ 0.39) | 0.048 | 0.049 |
| ACM     | 91.79        | 92.51 (↑ 0.72) | 0.026 | 0.026 |
| IMDB    | 60.23        | 64.63 (↑ 4.4)  | 0.066 | 0.069 |

TABLE 5

Ablation study for non-local operations on both homogeneous and heterogeneous graph datasets. Non-local operations improve the performance of FastGTNs on all datasets except for on BLOGCATALOG and FLICKR datasets. The attention scores on non-local operations show that our FastGTNs adaptively leverage non-local operations by adjusting the attention scores on non-local adjacency matrices.

### A.7 Interpretation of GTNs in Homogeneous Graphs

Fig. 10 shows ratios of each power of adjacency matrix in the output matrix $A^{(K)}$ from GTNs, respectively in the AIR-USA (left) and BLOGCATALOG (right) datasets. Based on the Eq. (30), we calculate the ratio of each hop in the final adjacency matrix after GTNs. Specifically, if the number of GT layers equals two, then the ratio of the identity matrix $I$ is obtained by $(\alpha(0) * \alpha(1) + \alpha(1) * \alpha(2) + \alpha(2) * \alpha(3))$. As we discussed in Section 4.2, since graph structures in the BLOGCATALOG dataset are noisy enough for a simple MLP to outperform other GNN baselines, GTNs learn to assign higher attention scores to the identity matrix, which effectively minimizes the range of neighborhoods. In contrast, in the AIR-USA dataset, each GT layer assigns relatively higher attention scores to the adjacency matrix, which expands the range of neighborhoods. GTNs can adaptively learn effective range of neighborhoods depending on the dataset.

Fig. 10. We visualized the ratios corresponding to each hop in the output matrix $A^{(K)}$ from GTNs based on the attention scores of each GT Layer in AIR-USA (left) and BLOGCATALOG (right) dataset. In (a), we use two GT layers in GTNs, thus output matrix from GTNs can involve up to three hop adjacency matrix. In (b), we use one GT layer, thus output matrix from GTNs can involve up to two hop adjacency matrix.