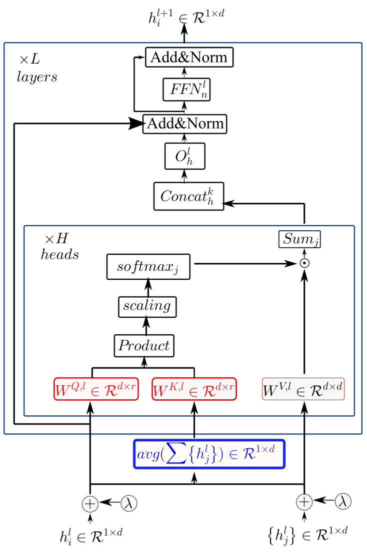
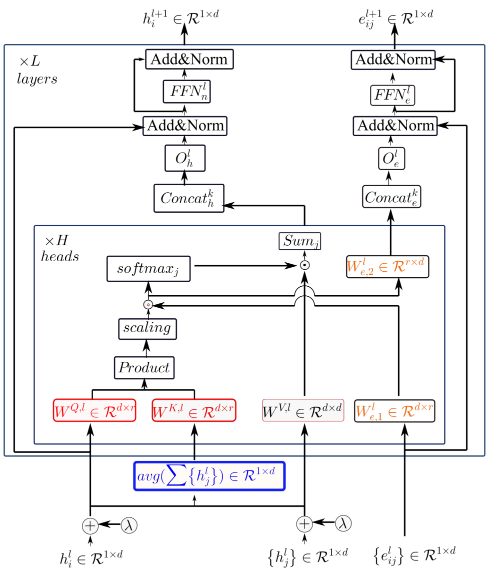
Graphical Abstract

Low-Rank and Global-Representation-Key-based Attention for Graph Transformers

Lingping Kong, Varun Ojha, Ruobin Gao, Ponnuthurai Nagaratnam Suganthan, Václav Snášel


$h_i^{l+1} \in \mathcal{R}^{1\times d}$
$\times L$ layers
Add&Norm
FFN_n^l
Add&Norm
O_h^l
$Concat_h^k$
$\times H$ heads
Sum_j
$softmax_j$
$scaling$
$Product$
$W^{Q,l} \in \mathcal{R}^{d\times r}$
$W^{K,l} \in \mathcal{R}^{d\times r}$
$W^{V,l} \in \mathcal{R}^{d\times d}$
$avg(\sum\{h_j^l\}) \in \mathcal{R}^{1\times d}$
λ
$h_i^l \in \mathcal{R}^{1\times d}$
$\{h_j^l\} \in \mathcal{R}^{1\times d}$


$h_i^{l+1} \in \mathcal{R}^{1\times d}$
$e_{ij}^{l+1} \in \mathcal{R}^{1\times d}$
$\times L$ layers
Add&Norm
FFN_n^l
FFN_e^l
Add&Norm
O_h^l
O_e^l
$Concat_h^k$
$Concat_e^k$
$\times H$ heads
Sum_j
$softmax_j$
$W_{e,2}^l \in \mathcal{R}^{r\times d}$
$scaling$
$Product$
$W^{Q,l} \in \mathcal{R}^{d\times r}$
$W^{K,l} \in \mathcal{R}^{d\times r}$
$W^{V,l} \in \mathcal{R}^{d\times d}$
$W_{e,1}^l \in \mathcal{R}^{d\times r}$
$avg(\sum\{h_j^l\}) \in \mathcal{R}^{1\times d}$
λ
$h_i^l \in \mathcal{R}^{1\times d}$
$\{h_j^l\} \in \mathcal{R}^{1\times d}$
$\{e_{ij}^l\} \in \mathcal{R}^{1\times d}$

Low-Rank and Global-Representation-Key-based Attention for Graph Transformers

Lingping Kong[a,*], Varun Ojha[b], Ruobin Gao[c], Ponnuthurai Nagaratnam Suganthan[d] and Václav Snášel[a]

[a]*Department of Computer Science, VSB-Technical University of Ostrava, Ostrava, The Czech Republic*
[b]*Department of Computer Science, University of Reading, Reading, United Kingdom*
[c]*School of Civil and Environmental Engineering, Nanyang Technological University, Singapore, Singapore*
[d]*School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore, Singapore*

ARTICLE INFO

Keywords:
Graph transformers
Graph representation
Low-rank attention
Global representation vector

ABSTRACT

Transformer architectures have been applied to Graph-specific data (e.g., social media networks, shopper lists, molecules, etc.). It has shown outstanding performance on node-level, edge-level, and graph-level prediction tasks. Moreover, researchers have proved that the attention matrix in Transformers has low-rank properties. The finding of low-rank properties inspires this work to propose a virtual Global Representation (GR) for attention, replacing pairwise node feature attention. First, as in classic Transformers with *Query*, *Key*, and *Value* derived from the node features, in the proposed model, the *Key* features are replaced with the GR, which learns the attention scores between nodes and the GR. Besides, two types of GR in batch and debatch modes are designed and experimented with within this work. Second, we provide mathematical representations showing the efficient way of node feature and edge feature updating in low-rank property with GR. Finally, the proposed model outperforms the generalized Graph Transformers baseline model and achieves 2.4% higher accuracy on the standard CLUSTER dataset.

1. Introduction

Transformers architecture has shown outstanding performance in sequential and graph data, such as text, speech, molecule, and paper citation data. As a result, model variants built upon Transformers have become prevalent in many domains, such as natural language processing (NLP) and computer vision, image segmentation, etc. Moreover, there are many attempts to devise efficient Transformers and leverage Transformers into the graph-specific field, aiming to improve graph representation performance compared with non-Transformer-based methods for solving graph-specific problems like molecule solubility and node classification. This paper follows the critical observation that self-attention is low-rank (Wang et al., 2020). At the same time, we also learn that the *GR can preserve the sparsity of graphs* and their low-rank property of attention.

Our work utilizes a graph neural network (GNN), where $G = (\mathcal{V}, E)$ is a graph, $\mathcal{V}$ is the set of vertices, and E is the corresponding edges between nodes. In this paper, we have undirected edges between nodes. Let a real vector represent a node, namely node feature as h_i. The task of GNN is to learn the node feature or graph feature, which can differentiate a node or graph by its vector. The algorithm of a GNN follows iterative updates on the node features (sometimes on the edge features) by aggregating vectors (features) of nodes from neighbors. In our work, we assign initial node (edge) features to the graph data and train the GNN model by updating node (edge) features and then classify the node or graph-level features by its output node or aggregated node features. Further, we predict the testing data for node classification or graph regression tasks.

During the GNN model training, we refer to the multi-head attention techniques from Transformers. Transformers have achieved tremendous success and have become the best performing model in NLP. One of the key components in Transformers is the multi-head self-attention mechanism (MH-SA), which we will refer to and modify. MH-SA divides a node feature vector into multiple sub-vectors (heads) and projects each sub-vector into *Query*, *Key*, and *Value* node

*Corresponding author
lingping_kong@yahoo.com (L. Kong); v.k.ojha@reading.ac.uk (.V. Ojha); gaor0009@e.ntu.edu.sg (.R. Gao); p.n.suganthan@gmail.com (.P.N. Suganthan); vaclav.snasel@vsb.cz (. Václav Snášel)
ORCID(s): 0000-0002-6825-1469 (L. Kong); 0000-0002-9256-1192 (.V. Ojha); 0000-0003-0781-1482 (.R. Gao); 0000-0003-0901-5105 (.P.N. Suganthan); 0000-0002-9600-8319 (. Václav Snášel)

features by a linear mapping. For example, suppose a node feature dimension is $h_{1\times d}$, and the number of the head is k. Then, the *Query*, *Key*, and *Value* vector get $h_{1\times \frac{d}{k}}$ dimension in the classic Transformers for each head.

The self-attention mechanism captures the pair-wise connection between nodes, which needs projection of each node feature to *Key* vector. In this work, we explore this self-attention to see the relation of the attention matrix with *Key* representation. As the *Query* is also the projection of node feature, where *Query* and *Key* are the central part of the attention matrix. We aim to explore the question that if a GR feature can replace this *Key*, which opens the connections between nodes.

The contribution of this paper is as follows. We take the similar low-rank attention on *Query* and *Key* as in paper Guo et al. (2019b), where low-rank denotes the low-dimensional representation. However, unlike the work in Guo et al. (2019b), we introduce a low-rank matrix that connects each node to **one** global virtual vertex instead of neighbors of the node. At the same time, we do not assume there is only one value score (or rank-1) necessity associations between token (word) and a virtual context as in Mehta et al. (2019). We give mathematical equations that show the possibility of low-rank property (a pre-defined rank r attention matrix) in the proposed model. Furthermore, we experiment with the proposed model compared to the generalized graph transformers' baseline, which gives an around 2.4% accuracy increase on the CLUSTER dataset. Moreover, this achievement uses only a simple low-rank global-information-based attention structure, which can be further improved by adopting other modules. Meanwhile, we try to generalize this low-rank Global-Representation-based model in this work. We believe the proposed model can be formed with corporate edge-augmented techniques or different encoding mechanisms for better performance.

In the rest of the paper, we first present the related work in Section 2. This is followed by our proposed GR-based graph transformer model in Section 3. Section 4 gives the environment settings related to the experiment. The results are presented and discussed in Sections 5. In Section 6, we conclude that the proposed technique can be considered as a baseline Transformer as it performs Competitively and has the potential of being further improved by using different encoding mechanisms or edge enhancement techniques.

2. Related work

As a preliminary, we introduce classic graph Transformers and its relevant research works in recent years. First, we briefly introduce positional encoding, which benefits the GNN to differentiate nodes. Then some research on GNN with pros and cons is presented. At last, we introduce the self-attention mechanism from Transformer and related research trying to develop a model with the low-rank property.

Vaswani et al. (2017) initially designed the Transformers for the sequence to sequence architecture. However, the position and order of words are the crucial parts of any language; hence one specific element of the Transformer's architecture is the positional encoding. In graph-related data, authors in Dufter et al. (2021) incorporate the positional encoding into the GNN to differentiate nodes in isomorphism or complex graphs. The positional embedding includes Laplacian eigenvectors positional encoding (Belkin and Niyogi, 2003), absolute (Vaswani et al., 2017; Gehring et al., 2017) or relative position encoding (You et al., 2019), stochastic positional embedding (SPE) (Liutkus et al., 2021), virtual node (Li et al., 2015), and structural position representations (SPR) (Wang et al., 2019b), etc.

There are several attempts to apply Transformers to graph data directly by abandoning the graph sparsity. For example, the model proposed in Yun et al. (2019) leaves the graph sparsity and applies the attention to all nodes of a graph. The author Yun et al. (2019) tends to capture the global information and feedback the information to all nodes rather than a node's neighbors. Other authors Dwivedi and Bresson (2021) question that it is inefficient compared to the model preserving the sparsity and using different ways of capturing global information, such as positional encoding. On the other hand, Dwivedi and Bresson (2021) employs the classic structure of the Transformers on graph representation and limits the attention mechanism on node's neighbors, which protects the graph's structural sparsity.

Furthermore, the author in Hussain et al. (2021) enhance the performance on *node classification* by employing the edge-augmenting techniques, which argue that global self-attention can serve as a flexible and adaptive alternative. At the same time, Ying et al. (2021) focuses on the improvement of graph representation learning by incorporating Transformers. The insight from their model is the effective encoding of graph structural information (GSI). Besides, they mathematically characterize the encodings and present the relations between popular GNN variants and GSI. The authors in Zhang et al. (2020) propose a subgraph batching scheme that passes fixed-size linkless subgraphs to the model with pre-trained and parallelized learning. The authors in Yun et al. (2019); Hu et al. (2020) emphasis on heterogeneous graphs.

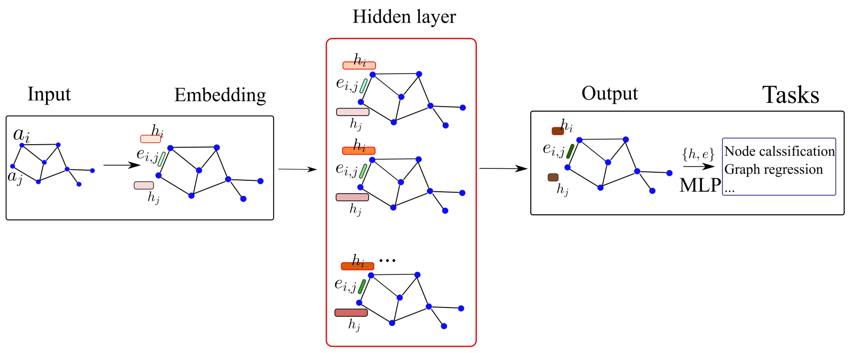


Figure 1: Graph neural network workflow

Self-attention is one of the fundamental structures of Transformers. Wang et al. (2020) mathematically proved that the attention matrix is low-rank. The attention matrix captures the low-rank property of self-attention and has attracted lots of detailed studies (Fan et al., 2021). For example, the low-rank property attention, the Sparse attention (SA), Pattern-based mechanisms. SA reduces the connections among nodes (vertices) and approximates long-range dependencies with a lightweight structure, such as Star-Transformer (Guo et al., 2019a), Longformer (Beltagy et al., 2020), Extended Transformer Construction (Ainslie et al., 2020), by incorporating global attention, band and dilated patterns, etc. Several attempts improve efficiency on the low-rank attention by assuming a low-rank structure in the $N \times N$ attention matrix. It requires less computational complexity, whereas the classic Transformers require $\mathcal{O}(N^2)$ time and memory. Wang et al. (2020) mathematically proved that the attention matrix is low-rank, and they propose to project the *Key* and *Value* to a lower-dimensional representation ($N \rightarrow r$). Therefore the $N \times N$ attention matrix is now compacted to ($N \times r$) matrix. The authors in Guo et al. (2019b) use singular value decomposition (SVD) to combine smaller attentions through linear mapping (the mapping components must be re-trained) forms approximately the attention QK^t. Another kind of low-rank constraint is letting the *Query* and *Key* have a lower dimension and making the *Value* the same dimension as the node feature.

The low-rank attention version of Transformers has become a popular toolkit for language data, with no complicated problem for corporating on graph-specific datasets. Nevertheless, as the field grows, it becomes necessary to identify critical architectures and validate the attention score that explores the connection between global representation and nodes in a low-rank attention form.

This work study the graph transfomers in a low-rank attention. And the proposed multi-head attention explores the connection between the node to its global representation instead of node's neighbor.

3. Global Representation based Graph Transformer

The self-attention mechanism has shown its remarkable ability for language modeling, even in the graph presentation. Researchers seek the low-rank attention strategies to alleviate the burden of $Q_{d\times d} \otimes K_{d\times d} \otimes V_{d\times d}$ attention. This section generalizes a low-rank attention mechanism for graph Transformers. However, instead of using three times projections on node features to get *Query*, *Key* and *Value*, we create a virtual Global feature (vector) to replace the *Key* without extra linear mapping in the proposed model.

That is, we define the *Global Representation* as a virtual feature vector that combines all nodes' features of a graph or batch data.

This paper proposes a generalized compacted transformer with a virtual global vector (e.g. GR) as GGformer, **which stands for GR based graph transformer model**. At the same time, we give a mathematical representation showing the construction of low-rank and global-representation-based attention in the proposed GGformer architecture.

3.1. Main architecture of GGformer

Fig 1 depicts a simple flowchart of graph neural network with input of node embedding, then The hidden layers update the node features, subsequently with the output with new node features. Finally, the output will pass to task-specific blocks, e.g., to a multilayer perception (MLP).

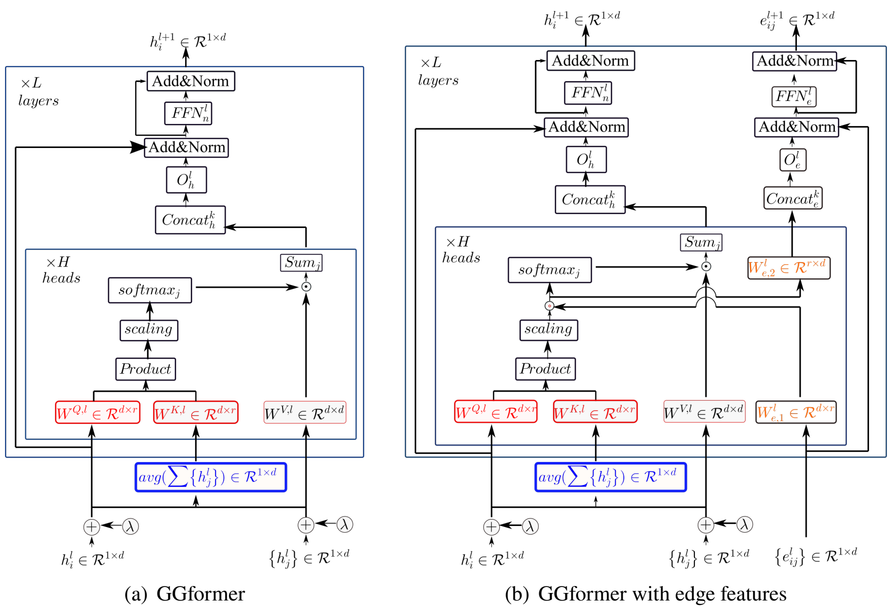


Figure 2: Block Diagram of GGformer with one layer architecture. (a): GGformer operating on node features; (b): GGformer with edge/node features, that edge features processing is designed to maintain attention-wise edge representations. $\textcircled{\lambda}$ denotes the position encoding. This model projects the node feature to *Query* with low dimension, to *Value* with the same dimension. The $\tilde{K}$ey is derived from the GR (average features of nodes) into a low dimension. The arrows '→' indicates the data processing sequence. $\times L$: indicates the model composes L layers in GGformer, H: the number of head; $\odot$: broadcast product of value to vector. $\circ$ is Hadamard product. Readers can refer to Section 3 for other symbols.

Input. Let $\mathcal{G}$ be a graph with embedding of vertices (nodes) $a_i \in \mathcal{R}^{1\times d_n}$ and edges $b_{ij} \in \mathcal{R}^{1\times d_n}$ between nodes i and j. Let $h_i^l, e_{ij}^l \in \mathcal{R}^{1\times d}$ be the node, edge feature representation, respectively, where l is the label of current network layer. The linear projection projects the embedding a_i, b_{ij} to d-dimensional hidden features h_i^0, e_{ij}^0 as:

$$\hat{h}_i^0 = Lin(a_i);\ e_{ij}^0 = Lin(b_{ij}), \tag{1}$$

where Lin denotes the linear mapping. The positional embedding is also passed via a linear projection to embed a d-dimension position feature. We add the pre-prepared node positional embedding λ of dimension d to the node features as:

$$h_i^0 = \hat{h}_i^0 + Lin(\lambda), \tag{2}$$

where the positional features are only added at the input layer. For the simplicity of the presentation, we skip the bias embedding parts for all operations above.

Hidden layer (Node feature updating). After getting the node feature embedding from input, the network will update the node features based on the formula in Eq 3 designed in the module, namely the GGformer module. The updated node feature will be fed into the next layer until the last layers:

$$h_i^{l+1} = f(h_i^l, \{h_j^l\}_{j\in V}), \tag{3}$$

where f represents the updating function, which defines the updating rules on node features or edge features based on a specific GNN, which will be described in Section 3.2.

Output. Then, a task-based MLP will process the node features and generate the outputs:

$$output = MLP(\{h_i^L, \{h_j^L\}\}) \tag{4}$$

The following operation is applied to the node representations at the final layer of GGformer. It is a task-specific procedure, then to learn the parameters by minimizing the task-dependent loss function. Readers could find the detailed task-based layers definitions in Appendix A. And the reader also can refer to Dwivedi and Bresson (2021), we followed the same definitions.

3.2. Low Rank GGformer attention layer

Fig 2 shows a layered architecture of GR-based graph Transformers (GGformer). We design Fig 2(a) for graphs with node features only and without explicit edge features, and Fig 2(b) models the layers preserves edge features to learn the available edge information.

This low-rank GGformer is significantly different from the canonical Transformers proposed in Vaswani et al. (2017) or the Graph Transformers in Dwivedi and Bresson (2021) on two aspects. First, the *Key* features are derived from the GR instead of individual node features. Second, we employ a twice projection on edge features to match the low ranked attention. We start to construct the node feature updating for a layer l.

A typical Transformers architecture uses the *Query* $\rightarrow Q$, *Key* $\rightarrow K$ and *Value* $\rightarrow V$ to formulate the attention matrix. Consider one head, and the node feature generates the attention using Eq 5:

$$Att(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d}}\right) V, \tag{5}$$

where $Q = hW^Q; K = hW^K; V = hW^V$, and three parameter mappings $\{W^Q, W^K; W^V\} \in \mathcal{R}^{d\times d}$ are learnable pamerters and h represents node features.

Although the *Query* $\rightarrow Q$ and *Key* $\rightarrow K$ effectively capture pairwise interactions among vertices, it might lead to *overfitting* due to the massive parameters. To counter overfitting, we re-written the Q, K attention in Eq 5 as low rank matrices as:

$$QK^T = hW^Q\,(hW^K)^T = hW^Q\,W^{K^T} h^T, \tag{6}$$

where the $W^Q(W^K)^T$ can be low rank matrices, and there is $W^Q, W^K \in \mathcal{R}^{d\times r}$, $r << d$. To further capture the relation among node features. We replace the *Key* component of $K = hW^K$ with $K = gW^K$. Hence, the attention matrix becomes:

$$QK^T = hW^Q\,gW^{K^T} = hW^Q\,W^{K^T} g^T \tag{7}$$

$$g = \frac{1}{n}\sum_{j=1}^{n} h_j, \;\; n = |\mathcal{V}|,$$

where g is the GR, which captures the general representation of nodes features. n denotes the number of nodes in a graph or batch data.

Remark 1. Any calculation of GR from vertices can be g. In our work, we use the mean value of all node features.

Remark 2. The GR g can be the average node features where nodes are from a graph. At the same time, the g can be the average node features where the node are from batch data, which alleviate the burden of processing debatch graphs and lower the computations.

Equation 7 is also written as follow:

$$QK^T = hW^Q\,W^{K^T} g^T = \sum_{d=1}^{r} h\,w_d^Q\,w_d^{K^T}\,g^T = \mathbb{1}^T\left(W^{Q^T} h^T \circ W^{K^T} g^T\right), \tag{8}$$

where $W^Q = [w_1^Q, w_2^Q, \dots, w_r^Q] \in \mathcal{R}^{d\times r}$ and $W^K = [w_1^K, w_2^K, \dots, w_r^K] \in \mathcal{R}^{d\times r}$ are two low rank matrices. $\circ$ is the Hadamard product which calculate the element-wise multiplication of two vectors, $\mathbb{1}^T$ is an all-one vector and r is the latent feature rank of the attention.

Remark 3. The low rank parameter r used in the attention matrix influences the performance on task, that the user has to pre-define it.

3.2.1. GGformer attention layer with only node features

Fig 2(a) presents one layer architecture of GGformer. The input is a node feature $h_i^l \in \mathcal{R}^{1\times d}$. The attention module projects one node feature into $Query \to Q^{k,l} \in \mathcal{R}^{1\times r}$ and $Value \to V^{k,l} \in \mathcal{R}^{1\times d}$. We project the GR g as $\tilde{K}$ey $\to \tilde{K}^{k,l} \in \mathcal{R}^{1\times r}$ and the equations are as:

$$Q^{k,l} = h_i^l\, W^{Q,l},\;\; V^{k,l} = h_i^l\, W^{V,l},\;\; \tilde{K}^{k,l} = g^l\, W^{K,l}; \tag{9}$$

where $\{W^{Q,l}, W^{K,l}\} \in \mathcal{R}^{d\times r}$ and $W^{V,l} \in \mathcal{R}^{d\times d}$ are learnable weights. And $Q^{k,l}, \tilde{K}^{k,l} \in \mathcal{R}^{1\times r}$, $V^{k,l} \in \mathcal{R}^{1\times d}$, and k denotes the current head number and l is the current layer number.

We now define the node feature updating equation for l_{th} layer:

$$\hat{h}_i^{l+1} = O_h^l \mathop{\Big\Vert}_{k=1}^{H} \left(\sum_{j\in\mathcal{V}_i} w_{j,g}^{k,l}\, h_j^l W^{V,l} \right) \tag{10}$$

$$\text{where, } w_{j,g}^{k,l} = softmax_j \left(\frac{hW^{Q,l} \bullet (gW^{K,l})^T}{\sqrt{r}} \right) = softmax_j \left(\frac{Q^{k,l}\{\tilde{K}^{k,l}\}^T}{\sqrt{r}} \right),$$

where $w_{j,g}^{k,l}$ is the attention score between a node j to g on layer l of number k head. $O_h^l \in \mathcal{R}^{d\times d}$, and $k = 1 \to H$ indicates the head label of attention, and $||$ denotes concatenation operation. $\mathcal{V}_i$ represents the neighbor nodes set of node i.

Refer to the GGformer diagram Fig 2(a). The attention score $w_{j,g}^{k,l}$ is the results that correspond to $\boxed{\text{Product}}$ as in $hW^{Q,l} \bullet (gW^{K,l})^T$, $\boxed{\text{Scaling}}$ as in $\frac{\star}{\sqrt{r}}$, and the $softmax_j$ as the $\boxed{Softmax_j}$. Then the attention score passes to a broadcast dot product $\odot$ with *Value* followed by a $\boxed{Sum_j}$ for all neighbors $j \in \mathcal{V}_i$ of node i as in $\left(\sum_{j\in\mathcal{V}_i} w_{j,g}^{k,l}\, h_j^l W^{V,l}\right)$. Note, $\mathcal{V}$ denotes all the nodes, $\mathcal{V}_i$ denotes the neighbors of node i. In the last, the result of Sum_j will pass to a $\boxed{Concat_h^k}$ for concatenating the number of H head results and a linear mapping $\boxed{O_h^k}$.

Remark 4. For numerical stability, we set the output values in the interval [-5, 5] after taking softmax.

Remark 5. The initial learnable parameter $\{W^{Q,l}, W^{K,l}\}$ is also clamped to a value in [-5, 5]. Other parameters are initialized as a value in [-1, 1]. In the experiments section, we show the comparisons on different initial values of parameters between value [-1,1] and value [-5,5]. The outputs of attention $\hat{h}_i^{l+1}$ are then passed to a Feed-Forward Network (FFN), and before and after FFN there are residual connections and normalization layers as follows:

$$\begin{aligned} \hat{\hat{h}}_i^{l+1} &= Norm(h_i^l + \hat{h}_i^{l+1}), \\ \hat{\hat{\hat{h}}}_i^{l+1} &= ReLU\,(\hat{\hat{h}}_i^{l+1} W_1^l)\, W_2^l, \\ h_i^{l+1} &= Norm(\hat{\hat{h}}_i^{l+1} + \hat{\hat{\hat{h}}}_i^{l+1}), \end{aligned} \tag{11}$$

where $W_1^l \in \mathcal{R}^{d\times 2d}, W_2^l \in \mathcal{R}^{2d\times d}$. $\hat{\hat{\hat{h}}}_i^{l+1}, \hat{\hat{h}}_i^{l+1}$ are the intermediate representations, and Layer-Norm (Ba et al., 2016) or Batch-Norm (Ioffe and Szegedy, 2015) can replace the Norm operation.

Refer to the GGformer diagram in Fig 2(a). The $\hat{\hat{h}}_i^{l+1}$ is the result that correspond to the first $\boxed{\text{Add\&Norm}}$, then $\hat{\hat{\hat{h}}}_i^{l+1}$ is the result that correspond to the $\boxed{FFN_h^l}$, next the node feature pass to the second $\boxed{\text{Add\&Norm}}$, which produce the result h_i^{l+1}. At last, h_i^{l+1} will pass to the MLP layer for further analysis based on specific task (see Appendix A).

3.2.2. GGformer attention layer with node and edge features

The low-rank GGformer with edge features also utlizes the *Query* and global $\tilde{K}$ey. Furthermore, we tie the global-based attention score from node to Global Presentation to implicit edge features. From the matrix operation view, we calculate the vector multiplication part, excluding the addition process in the Hadamard product. This can be calculated as:

$$\begin{aligned}
\tilde{W} &= \sum_{d=1}^{r} h_i \, w_d^Q \, w_d^{K^T} \, g^T = \left(W^{Q,l^T} h^T \circ W^{K,l^T} g^T \right) \\
E^{k,l} &= e_{i,j}^l \, W_1^e; \\
\hat{w}_{j,g}^{k,l} &= E^{k,l} \circ \frac{\tilde{W}}{\sqrt{r}} \\
w_{j,g}^{k,l} &= softmax_j \left(\mathbb{1}^T \hat{w}_{j,g}^{k,l} \right) \\
\hat{e}_{i,j}^{l+1} &= O_e^l \Big\|_{k=1}^{H} \left(\hat{w}_{j,g}^{k,l} \, W_2^e \right) \\
\hat{h}_i^{l+1} &= O_h^l \Big\|_{k=1}^{H} \left(\sum_{j \in \mathcal{V}_i} w_{j,g}^{k,l} \, h_j^l W^{V,l} \right)
\end{aligned} \tag{12}$$

where the $W^{Q,l}, W^{K,l}, W^{V,l}$ are the same defined learnable parameters for layer l introduced above, $\{O_h^l, O_e^l\} \in \mathcal{R}^{d \times d}$,. $W_1^e \in \mathcal{R}^{d \times r}, W_2^e \in \mathcal{R}^{r \times d}$ are learnable weights. We have the same clamp operation as mentioned in Remark 5 for the values after introducing softmax in architecture without edge features above.

Refer to the GGformer diagram in Fig 2(b). $\hat{e}_{i,j}^{l+1}$ is the result that correspond to operation $\boxed{Concat_e^k}$ and $\boxed{O_e^l}$.

Then the output of $\hat{h}_i^{l+1}$ and $\hat{e}_{i,j}^{l+1}$ goes to FFN with two residual connections and normalization preceded and succeeded as:

$$\begin{aligned}
\hat{\hat{h}}_i^{l+1} &= Norm(h_i^l + \hat{h}_i^{l+1}), \\
\hat{\hat{\hat{h}}}_i^{l+1} &= ReLU\ (\hat{\hat{h}}_i^{l+1} W_{h,1}^l)\ W_{h,2}^l, \\
h_i^{l+1} &= Norm(\hat{\hat{h}}_i^{l+1} + \hat{\hat{\hat{h}}}_i^{l+1}),
\end{aligned} \tag{13}$$

and,

$$\begin{aligned}
\hat{\hat{e}}_{ij}^{l+1} &= Norm(e_{ij}^l + \hat{e}_{ij}^{l+1}), \\
\hat{\hat{\hat{e}}}_{ij}^{l+1} &= ReLU\ (\hat{\hat{e}}_{ij}^{l+1} W_{e,1}^l)\ W_{e,2}^l, \\
e_i^{l+1} &= Norm(\hat{\hat{e}}_{ij}^{l+1} + \hat{\hat{\hat{e}}}_{ij}^{l+1}),
\end{aligned} \tag{14}$$

where $\{W_{h,1}^l, W_{e,1}^l\} \in \mathcal{R}^{d \times 2d}$, and $\{W_{h,2}^l, W_{e,2}^l\} \in \mathcal{R}^{2d \times d}$. $\hat{\hat{\hat{e}}}_{ij}^{l+1}, \hat{\hat{e}}_{ij}^{l+1}$ are the intermediate representations.

Refer to the GGformer diagram in Fig 2(b). Two $\boxed{\text{Add\&Norm}}$ on edge feature updating side correspond to the result of $\hat{\hat{e}}_{ij}^{l+1}$ and e_i^{l+1}. The result of $\hat{\hat{\hat{e}}}_{ij}^{l+1}$ is following the operation of $\boxed{FFN_e^l}$. The rest of blocks operations are the same as in Section 3.2.1.

The final layer output h_i^{l+1} and e_i^{l+1} are the updated node feature and edge feature, respectively. Then it follows the task based MLP layers (see Appendix A).

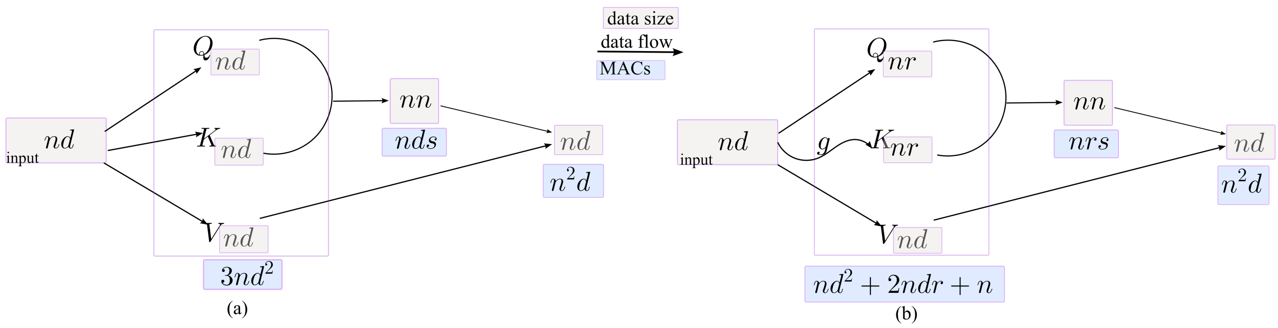


Figure 3: Computation complexity. (a) complexity of GraphTransformer (b) complexity of GGformer

3.3. Complexity

We compare the complexity with baseline model GraphTransformer (Dwivedi and Bresson, 2021). GGformer shares a similar structure as in GraphTransformer in one-layer architecture, except for the multi-head attention module. So we only calculate the multiply-accumulate operations (MACs) of the multi-head attention part.

Suppose there are n nodes, each node feature-length is d, the number of ranks is r. Note that the model process is on graph-specific data. We only need to calculate the neighbors' attention scores and set others to be zero. However, the neighbor size varies with the data. Here we suppose the data has an average neighbor size of s. The MACs of GraphTransformer and GGformer are shown in Fig 3, where the complexity of GraphTransformer as in Fig 3(a) is $\Omega(3nd^2 + n^2d + nds)$, while the complexity of GGformer as in Fig 3(b) is $\Omega(nd^2 + 2ndr + nrs + n^2d)$.

Here the complexity of GGformer is much lower than GraphTransformer as both $\Omega(nd^2 + 2ndr + n))$ and $\Omega(nrs)$ are much lower than $\Omega(3nd^2)$ and $\Omega(nds)$. This is also because $r < d$.

4. Experiment

We conduct the experiment and give the detailed performance of the proposed GGformer on three benchmark data sets, namely ZINC, PATTERN, and CLUSTER, the three used in Dwivedi and Bresson (2021), which are all widely used benchmark datasets. We run experiments on Intel Xeon Gold 6154 server with 2 Nvidia Tesla V100 GPUs. All experiments use the Laplacian positional encodings (LapPE), and the details of LapPE can be found in Belkin and Niyogi (2003).

4.1. Datasets

A brief description of each dataset as follows:

ZINC, Graph-level task, Regression (Irwin et al., 2012). ZINC is molecular data connected with undirected edges. Each molecule represents a graph with nodes from atoms and edges from bonds. We predict the constrained solubility as a regression value of each molecular graph. The dataset we use is a 12K randomly sampled subset as in Dwivedi and Bresson (2021) and Dwivedi et al. (2020).

CLUSTER, Node-level task, Classification (Dwivedi et al., 2020). The Stochastic Block Model (SBM) generates CLUSTER for the node classification task. ClUSTER dataset has six different classes. We use 12K undirected graphs dataset and each graph have 41-190 nodes. Each graph has discrete node features and does not have explicit edge features.

PATTERN, Node-level task, Classification (Abbe, 2017). SBM also produces a node classification dataset PATTERN with two pattern labels (classes). The data has discrete features for nodes but no edge features. We chose a 14K graphs dataset as in Dwivedi and Bresson (2021).

4.2. Model Configurations

All experiments are implemented in Python3.6 under PyTorch (Paszke et al., 2019) and DGL (Wang et al., 2019a) platform, we follow the same protocol as in Dwivedi and Bresson (2021). As for the basic experiments, we use the same environmental setting, ten layers, eight attention heads of each layer. We use a low rank of attentions matrix in the model, so the number of parameters will be around 400K on some datasets. All the settings are shown in Result

Table 1
Results of GGformer in Rank 4 on ZINC dataset with L=10

Dataset	Graph	Global	Pos	Norm	Param	Test Acc	Train Acc	Seeds	#Epoch	Epoch/Total
ZINC	Full	g-debatch	LapPE	BN	547969	0.570±0.001	0.431±0.026	2	348	118.74s/12.10hr
				LN		0.681±0.003	0.545±0.004	2	234.5	114.00s/8.03hr
			NoPE	BN	547393	0.672±0.002	0.573±0.001	2	186	100.84s/5.29hr
				LN		0.675±0.002	0.558±0.003	2	218.5	107.72s/6.61hr
		g-batch	LapPE	BN	547969	0.570±0.003	0.449±0.027	2	308.5	14.55s/1.82hr
				LN		0.680±0.000	0.552±0.000	1	229	29.33s/1.91hr
			NoPE	BN	547393	0.675±0.001	0.563±0.002	2	222	15.67s/0.99hr
				LN		0.677±0.002	0.558±0.008	2	218.5	19.82s/1.23hr
	Sparse	g-debatch	LapPE	BN	547969	0.315±0.028	0.096±0.014	2	298	121.16s/10.65hr
				LN		0.371±0.011	0.091±0.016	2	258	107.86s/8.28hr
			NoPE	BN	547393	0.901±0.071	0.837±0.064	2	179	438.12s/22.03hr
				LN		0.339±0.016	0.042±0.003	2	307.5	98.41s/8.44hr
		g-batch	LapPE	BN	547969	**0.249±0.004**	0.093±0.004	2	297	12.50s/1.57hr
				LN		0.278±0.001	0.065±0.004	2	334.5	14.75s/1.92hr
			NoPE	BN	547393	0.716±0.033	0.651±0.035	2	184	373.88s/19.24hr
				LN		0.327±0.014	0.083±0.025	2	297	13.50s/1.12hr

Rank 4 ($r = 4$), lower value the better

Table 1 to Table 9. Experiments on Section 5.3 will set to be an arbitrary number of layers and ensure the trainable parameters are in the range of 500K.

During the whole experiments, we set the learning rate decay strategy same as in Dwivedi and Bresson (2021) to ensure the fair comparisons. We analyzed the basic model's, low rank-based, and fixed interval parameters results. There may be different runs on various experiments. So we put the runs times as '#seed' to indicate how many runs for a particular experiment.

5. Results and Discussion

We now present the performance of the proposed model on the task of graph regression and node classification. We first test the proposed model with various settings on data with fully connected graph and sparse graph in Section 5.1. Then pick a proper environment setting (with Norm, Pos, Global, Graph...) as the default setting for the GGformer, which will be used in later experiments.

The second part in Section 5.2 tested the proposed model on the ZINC, PATTERN, and CLUSTER dataset with different pre-defined rank r values. Then, the initial trainable weights parameters may affect the model's performance. Therefore, we experiment on GGformer with different initial weights.

At last, for a fairness of comparison, we tune the hyperparameters of GGformer to be 500K number of trainable parameters, then test GGformer to compare with other baseline models in Section 5.3.

5.1. Performance on ZINC data

Table 1 reports the GGformer performance on ZINC data with rank $r = 4$ and 10 layers. The performance measure for ZINC is mean absolute error (MAE), the lower the value, the better. In Table 1, **Sparse** denotes the given data in graph form and **Full** denotes the modified data with fullly connected edges among all nodes. **Seeds** denotes the independed run times. g-**batch** and g-**debatch** execute the GGformer with two different GR calculations, where g-**batch** use the global g from nodes of batch data, and g-**debatch** use g from nodes of single graph. **LN** and **BN** indicate layer-Norm and batch-Norm operation, respectively. **Param** denotes the number of trainable parameters in current GGformer. **LapPE** and **NoPE** experiment on model with and without Laplacian positional encoding. L: the number of layers. **Bold** result indicates the best result for tested dataset at the current table.

From Table 1, we can see that GGformer in BN and with LapPE on sparse graph performs better than other combinations (see Graph Sparse, Global g-batch, Pos LapPE, and BN Norm in Table 1). First, all the fully connected graph results are worse than the results on the sparse graph, which shows the GGformer effectiveness on graph-specific data ((see rows in Graph Sparse VS Graph Full, with Pos LapPE, and BN Norm in Table 1)). Second, the results with LapPE of GGformer perform better than those without using LapPE (see rows in Pos LapPE Vs. Pos NoPE in Table 1). Finally, the BN is more effective than LN on GGformer architecture (see rows in Norm BN Vs. Norm LN in Table 1).

Table 2
Results of GGformer with different Ranks on ZINC dataset

ZINC-Sparse in g-batch with L=10													
		LapPE						NoPE					
r	Norm	Param	Test Acc	Train Acc	Seeds	#Epoch	Epoch/Total	Param	Test Acc	Train Acc	Seeds	#Epoch	Epoch/Total
2		507009	0.238±0.022	0.074±0.020	9	337.11	34.61s/5.16hr	506433	0.844±0.133	0.780±0.129	4	179.5	469.92s/23.61hr
4	BN	547969	**0.235±0.016**	0.075±0.019	8	324.75	14.55s/1.99hr	547393	0.716±0.033	0.651±0.035	2	184	373.88s/19.24hr
6		588929	**0.234±0.022**	0.076±0.019	8	314.12	14.55s/1.86hr	588353	0.967±0.065	0.912±0.065	2	172.5	394.81s/19.10hr
2		507009	0.321±0.090	0.151±0.147	4	302.75	16.82s/2.08hr	506433	0.312±0.023	0.101±0.035	4	318.75	16.29s/1.45hr
4	LN	547969	0.278±0.001	0.065±0.004	2	334.5	14.75s/1.92hr	547393	0.327±0.014	0.083±0.025	2	297	13.50s/1.12hr
6		588929	0.316±0.006	0.113±0.014	2	332	12.89s/1.76hr	588353	0.286±0.008	0.053±0.007	2	346.5	11.71s/1.14hr

r is Rank

In Table 2, **Rank** denotes the hyperparameter rank r in low-rank multi-head attention module, which we set it as two, four and six in the experiment as $r = [2, 4, 6]$. g-batch of GR is faster to compute than g-debatch, hence in this experiment, we only execute g-batch version of GGformer. Table 2 shows the performance of GGformer with Rank 2 to Rank 6 with LapPE. From Table 2, we see the GGformer with $r = 6$ in BN performs the best as the error is in 0.234. Then, when the $r = 4$, the GGformer get a result of 0.235. Next, when the model is set to $r = 2$, it gets an error of 0.238, which performs closely with the results r equals to four and six. We can conclude that the higher ranks does not boost the performance much. On the other hand, the GGformer in NoPE mode performs poor with both BN and LN operations as the 0.286 is the best performance in NoPE columns of Table 2.

We will remove the tests on the Full graph at the following experiments and only experiment on the original sparse graph. Besides, we find the LapPE encoding has different effects on PATTERN and CLUSTER dataset, hence we will conduct experiments on these two datasets that differ on LapPE and Norm operations.

5.2. Performance of GGformer with various settings

The performance measure for PATTERN and CLUSTER data is accuracy (ACC). Table 3 and Table 4 reports the results of GGformer in rank 4 with 10 layers. From Table 3 and Table 4, we see the GGformer performs poor in LN than BN, and the resutls are slightly better in g-debatch calculation than g-batch model on both PATTERN and CLUSTER dataset. Moreover, the positional encoding is more useful on PATTERN than the CLUSTER dataset, where the results GGformer with LapPE performs better than NoPE with all LN/BN modes; On the contrary, GGformer with BN performs better than LN with all LapPE/NoPE modes. The best results of GGformer on CLUSTER and PATTERN data are model with LapPE and BN and in g-debatch. The corresponding accuracy on these datasets were 73.68% and 85.003%.

Table 5 and Table 6 give the results of GGformer in various ranks from $r = 1$ to 6. Table 5 shows the GGformer gives the competitive results on rank two and rank 4 in BN and LapPE mode. Table 6 shows the GGformer gives the competitive results with and without LapPE modes on rank 2 and 1, respectively. Both Table 5 and Table 6 results indicate that GGformer does not improve performance much with the increasing rank numbers.

We now present the experiment on GGformer with little changes on designs. **GGformer-ini**, which indicates the initial trainable parameters (weights) on $W^{Q,l}, W^{K,l}$ are values in the interval [-1, 1]. GGformer initialize the trainable $W^{Q,l}, W^{K,l}$ parameters in the value of [-5,5]. We tested the performance on GGformer with different initialized parameters on $W^{Q,l}, W^{K,l}$. The results is in Table 7, which shows the results of GGformer with different initial parameters on $W^{Q,l}, W^{K,l}$ on CLUSTER dataset. The best performance is GGformer with LapPE inBN of rank four, which achieves an accuracy of 73.729%. The results of GGformer with initial parameters [-1,1] and [-5,-5] do not show a huge difference, as compared with GGformer get an accuracy of 73.642% with [-5,5] initial parameters.

5.3. Performance of GGformer with around of 500K parameters

From Table 5 and Table 6, we learn that GGformer performs slightly better with g-detach GR calculation than g-batch. In this experiment, we use GGformer with g-batch calculation as it is fast to compute. Moreover, for a fairness comparison with state-of-art algorithms as the baseline performance, we tune the hyperparameter of the proposed GGformer to a model with around 500K number of parameters. In this experiment, we tune the rank (r) between two and four, number of layers at 20 and 18, respectively. And the BatchSize **(BS)** start from low 13/16 to high 104/128 for CLUSTER/PATTERN data, respectively. Most importantly, we use node feature length with 64 instead of 80 (all tables except for Table 8 and Table 9 use node feature length 80 for each data).

Table 8 and Table 9 show the results of GGformer on PATTERN and CLUSTER data, respectively. The best performance of CLUSTER is 75.566%, which increases the accuracy 2% to the one without tuning parameters. Furthermore, GGformer gets 85.217% accuracy on the PATTERN data, which is also better than the one without hyperparameter tuning.

Table 3
Results of GGformer in Rank 4 on PATTERN dataset; Read the abbr from Section 5.1

PATTERN-Sparse with L=10													
			g-batch					*g*-debatch					
Pos	Norm	Param	Test Acc	Train Acc	Seeds	#Epoch	Epoch/Total	Test Acc	Train Acc	Seeds	#Epoch	Epoch/Total
LapPE	BN	446182	**84.979±0.076**	85.64±0.104	8	166	535.76s/28.90hr	**85.003±0.027**	85.72±0.036	3	164.33	180.40s/8.91hr
	LN	446182	84.853±0.092	85.68±0.191	8	183.12	115.93s/7.56hr	84.822±0.062	85.78±0.091	4	177.25	201.18s/10.47hr
NoPE	BN	445942	69.926±0.175	81.77±1.022	4	143.25	141.24s/5.69hr	69.938±0.205	81.49±1.062	4	139.75	174.07s/6.84hr
	LN	445942	67.907±0.088	83.87±1.383	4	138.25	159.75s/6.17hr	67.739±0.329	83.72±0.749	3	139.67	187.01s/7.35hr

Acc, the higher value the better

Table 4
Results of GGformer in Rank 4 on CLUSTER dataset results; Read the abbr from Section 5.1

CLUSTER-Sparse with L=10												
			g-batch					*g*-debatch				
Pos	Norm	Param	Test Acc	Train Acc	Seeds	#Epoch	Epoch/Total	Test Acc	Train Acc	Seeds	#Epoch	Epoch/Total
LapPE	BN	447226	73.419±0.404	80.356±1.462	8	143.88	331.93s/17.53hr	**73.680±0.121**	78.985±0.649	4	133.75	157.73s/6.57hr
	LN	447226	57.532±23.62	58.266±24.08	4	260	109.61s/11.51hr	64.636±9.197	66.007±10.42	3	252.67	169.14s/11.89hr
NoPE	BN	446346	73.601±0.238	78.658±0.816	8	131.62	64.20s/2.37hr	73.504±0.305	78.905±0.812	4	138.5	155.79s/6.09hr
	LN	446346	44.846±21.29	45.112±21.618	4	156.75	141.83s/6.08hr	34.834±18.69	34.888±18.73	4	111.25	151.30s/4.76hr

Table 5
Results of GGformer with different Ranks on PATTERN dataset; Read the abbr from Section 5.1

PATTERN-Sparse in *g*-batch with L=10												
	BN+ LapPE						LN+LapPE					
Rank	Param	Test Acc	Train Acc	Seeds	#Epoch	Epoch/Total	Param	Test Acc	Train Acc	Seeds	#Epoch	Epoch/Total
1	407782	84.953±0.044	85.729±0.088	4	169.5	83.98s/5.02hr	407782	84.782±0.152	85.696±0.289	3	174	70.38s/4.03hr
2	420582	**84.995±0.079**	85.703±0.147	8	165.38	75.17s/4.26hr	420582	84.711±0.194	86.043±0.246	8	176.5	63.70s/3.77hr
4	446182	**84.979±0.076**	85.648±0.104	8	166	535.76s/28.90hr	446182	84.853±0.092	85.689±0.191	8	183.12	115.93s/7.56hr
6	471782	84.944±0.088	85.702±0.142	8	171	97.03s/5.62hr	471782	84.634±0.518	85.741±0.509	8	175.38	113.09s/6.27hr

Table 6
Results of GGformer with different Ranks on CLUSTER dataset; Read the abbr from Section 5.1

CLUSTER-Sparse in *g*-batch with L=10												
	BN+LapPE						BN+NoPE					
Rank	Param	Test Acc	Train Acc	Seeds	#Epoch	Epoch/Total	Param	Test Acc	Train Acc	Seeds	#Epoch	Epoch/Total
1	408826	73.480±0.286	79.546±0.822	4	140.5	63.43s/3.50hr	407946	**73.689±0.090**	78.635±0.182	4	130	56.75s/2.08hr
2	421626	**73.642±0.291**	79.300±0.678	8	132.5	109.33s/5.00hr	420746	73.619±0.216	78.734±0.294	8	133.5	115.27s/4.30hr
4	447226	73.419±0.404	80.356±1.462	8	143.88	331.93s/17.53hr	446346	73.601±0.238	78.658±0.816	8	131.62	64.20s/2.37hr
6	472826	73.500±0.315	79.037±1.241	8	133.62	66.00s/3.67hr	471946	73.482±0.275	78.716±0.461	8	132.12	48.65s/1.81hr

5.4. Summary of comparison results

Table 10 presents the final comparison results, in which GGformer outperforms the baseline model of generalized GraphTransformers on PATTERN and CLUSTER data by 0.4% and 2.4% improvement in terms of accuracy (see Table 10 row of GraphTransformer and GGformer). While the GGformer does not show an outstanding result on ZINC data, it outperforms the GAT baseline. We see the GGformer is the best with BN and LapPE operation with all

Table 7
Results of GGformer-ini with different Ranks on CLUSTER dataset; Read the abbr from Section 5.1

GGformer-ini: Initial parameters range in [-1, 1] on $W^{Q,l}, W^{K,l}$

CLUSTER-Sparse with L=10 + BN + g-batch

Pos	Rank	Param	Test Acc	Train Acc	#seed	Pos	Rank	Param	Test Acc	Train Acc	#seed
	1	408826	73.138±0.094	79.648±0.798	2		1	407946	73.512±0.238	79.281±0.833	2
LapPE	2	421626	73.512±0.381	79.328±0.981	7	NoPE	2	420746	73.501±0.265	79.637±0.902	6
	4	447226	**73.729±0.117**	79.041±0.671	15		4	446346	73.242±0.168	78.824±0.629	6

Table 8
Results of GGformer on CLUSTER dataset with parameter tuning

CLUSTER-Sparse LapPE/BN/g-batch

	Rank $r = 2$						Rank $r = 4$					
BS	L	Params	Test Acc	Train Acc	#Epoch	Epoch/Total	L	Params	Test Acc	Train Acc	#Epoch	Epoch/Total
16	20	546582	74.777±0.214	85.910±0.737	123.75	216.15s/8.16hr	18	529174	74.812±0.221	85.281±0.472	133.75	102.69s/4.80hr
32	20	546582	75.086±0.412	84.871±1.376	121.75	107.93s/4.34hr	18	529174	74.699±0.265	84.833±0.802	122.5	193.74s/7.36hr
64	20	546582	**75.566±0.114**	82.825±1.189	122.25	52.86s/2.69hr	18	529174	75.508±0.110	82.005±0.640	123.5	52.67s/2.52hr
128	20	546582	75.564±0.070	81.444±0.982	137	35.43s/2.33hr	18	529174	75.440±0.087	80.668±0.833	139.75	32.18s/1.88hr

Table 9
Results of GGformer on PATTERN dataset with parameter tuning; Read the abbr from Section 5.1

PATTERN-Sparse LapPE/BN/g-batch

	Rank $r = 2$						Rank $r = 4$					
BS	L	Param	Test Acc	Train Acc	#Epoch	Epoch/Total	L	Param	Test Acc	Train Acc	#Epoch	Epoch/Total
13	20	545746	85.087±0.076	86.323±0.106	145.25	296.42s/12.65hr	18	528338	85.137±0.081	86.148±0.113	155	136.56s/6.88hr
26	20	545746	**85.217±0.058**	86.226±0.145	141.5	148.21s/6.43hr	18	528338	85.066±0.062	86.171±0.157	145.25	207.30s/9.07hr
52	20	545746	85.128±0.070	86.022±0.149	165.25	79.70s/4.61hr	18	528338	85.140±0.057	85.977±0.156	161.25	49.33s/2.87hr
104	20	545746	85.070±0.057	85.670±0.098	182.75	47.79s/3.42hr	18	528338	85.083±0.059	85.679±0.176	190.5	45.77s/3.08hr

Table 10
Comparison of the best performing scores of GGformer on each dataset against the reference baselines.

GNN baseline scores from Dwivedi et al. (2020) and Dwivedi and Bresson (2021)

Model	ZINC	PATTERN	CLUSTER
GCN (Kipf and Welling, 2016)	0.367±0.011	68.498±0.976	71.892±0.334
GraphSage (Hamilton et al., 2017)	0.398±0.002	50.492±0.001	63.844±0.110
GAT (Velickovic et al., 2017)	0.384±0.007	78.271±0.186	70.587±0.447
GatedGCN (Bresson and Laurent, 2017)	0.214±0.013	86.508±0.085	76.082±0.196
GIN (Xu et al., 2018)	0.526±0.051	85.387±0.136	64.716±1.553
RingGNN (Chen et al., 2019)	0.704±0.003	86.244±0.025	22.340±0.000
GraphTransformers (Dwivedi and Bresson, 2021)	0.226±0.014	84.808±0.068	73.169±0.622
		Our Results	
GGformer (this work)	0.234±0.022	85.217±0.058	75.566±0.114

the experiments. GGformer performs competitively with GraphTransformer while GGformer is at low-rank form with fewer parameters.

From the experiment above, it is reasonable to believe that the pairwise attention between nodes is much related to the GR. In other words, the attention score a is also the similarity degree between two nodes, and the proposed low-rank GR attention score b calculates the similarity between a node to the virtual vector; these two scores $\{a, b\}$ work in the same way with different calculations. Therefore, they can be the alternative replacement to each other. More importantly, the proposed low-rank GR attention score uses a less trainable parameter, which relieves the burden of the model and avoids overfitting due to too many parameters.

6. Conclusion

This paper proposed a straightforward application of Graph Transformers with GR based attention. Instead of calculating the pairwise attention score among nodes, we measure the connection score between the node and the graph-specific GR. The proposed model benefits from preserving the sparsity of graphs and the low-rank property of attention. As a result, the initial results show that the proposed model performs better on the graph data with sparsity than without sparsity. Furthermore, the proposed model achieves 2.4% up accuracy on CLUSTER data and 0.4% up on PATTERN data than the generalized graph Transformers. Given the simple nature of GGformer in terms of using low-rank attention and simple architecture, the proposed model can be another Graph Transformer baseline model for further improvement.

Acknowlegement

This work was supported by the Ministry of Education, Youth and Sports of the Czech Republic in project META MO-COP, reg.no.LTAIN19176.

References

Abbe, E., 2017. Community detection and stochastic block models: recent developments. The Journal of Machine Learning Research 18, 6446–6531.

Ainslie, J., Ontanon, S., Alberti, C., Cvicek, V., Fisher, Z., Pham, P., Ravula, A., Sanghai, S., Wang, Q., Yang, L., 2020. ETC: encoding long and structured inputs in transformers. arXiv:2004.08483 .

Ba, J.L., Kiros, J.R., Hinton, G.E., 2016. Layer normalization. arXiv:1607.06450 .

Belkin, M., Niyogi, P., 2003. Laplacian eigenmaps for dimensionality reduction and data representation. Neural Computation 15, 1373–1396.

Beltagy, I., Peters, M.E., Cohan, A., 2020. Longformer: The long-document transformer. arXiv:2004.05150 .

Bresson, X., Laurent, T., 2017. Residual gated graph convnets. arXiv:1711.07553 .

Chen, Z., Villar, S., Chen, L., Bruna, J., 2019. On the equivalence between graph isomorphism testing and function approximation with gnns. Advances in neural information processing systems 32.

Dufter, P., Schmitt, M., Schütze, H., 2021. Position information in transformers: An overview. arXiv:2102.11090 .

Dwivedi, V.P., Bresson, X., 2021. A generalization of transformer networks to graphs. arXiv:2012.09699 .

Dwivedi, V.P., Joshi, C.K., Laurent, T., Bengio, Y., Bresson, X., 2020. Benchmarking graph neural networks. arXiv:2003.00982 .

Fan, X., Liu, Z., Lian, J., Zhao, W.X., Xie, X., Wen, J.R., 2021. Lighter and better: low-rank decomposed self-attention networks for next-item recommendation, in: Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 1733–1737.

Gehring, J., Auli, M., Grangier, D., Yarats, D., Dauphin, Y.N., 2017. Convolutional sequence to sequence learning, in: International Conference on Machine Learning, PMLR. pp. 1243–1252.

Guo, Q., Qiu, X., Liu, P., Shao, Y., Xue, X., Zhang, Z., 2019a. Star-transformer. arXiv:1902.09113 .

Guo, Q., Qiu, X., Xue, X., Zhang, Z., 2019b. Low-rank and locality constrained self-attention for sequence modeling. IEEE/ACM Transactions on Audio, Speech, and Language Processing 27, 2213–2222.

Hamilton, W., Ying, Z., Leskovec, J., 2017. Inductive representation learning on large graphs. Advances in neural information processing systems 30.

Hu, Z., Dong, Y., Wang, K., Sun, Y., 2020. Heterogeneous graph transformer, in: Proceedings of The Web Conference 2020, pp. 2704–2710.

Hussain, M.S., Zaki, M.J., Subramanian, D., 2021. Edge-augmented graph transformers: Global self-attention is enough for graphs. arXiv:2108.03348 .

Ioffe, S., Szegedy, C., 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift, in: International Conference on Machine Learning, PMLR. pp. 448–456.

Irwin, J.J., Sterling, T., Mysinger, M.M., Bolstad, E.S., Coleman, R.G., 2012. ZINC: a free tool to discover chemistry for biology. Journal of Chemical Information and Modeling 52, 1757–1768.

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

Li, Y., Tarlow, D., Brockschmidt, M., Zemel, R., 2015. Gated graph sequence neural networks. arXiv:1511.05493 .

Liutkus, A., Cifka, O., Wu, S.L., Simsekli, U., Yang, Y.H., Richard, G., 2021. Relative positional encoding for transformers with linear complexity, in: International Conference on Machine Learning, PMLR. pp. 7067–7079.

Mehta, S., Rangwala, H., Ramakrishnan, N., 2019. Low rank factorization for compact multi-head self-attention. arXiv:1912.00835 .

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al., 2019. Pytorch: An imperative style, high-performance deep learning library. Advances in Neural Information Processing Systems 32.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I., 2017. Attention is all you need. Advances in Neural Information Processing Systems 30.

Velickovic, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., Bengio, Y., 2017. Graph attention networks. stat 1050, 20.

Wang, M., Yu, L., Zheng, D., Gan, Q., Gai, Y., Ye, Z., Li, M., Zhou, J., Huang, Q., Ma, C., et al., 2019a. Deep graph library: Towards efficient and scalable deep learning on graphs. .

Wang, S., Li, B.Z., Khabsa, M., Fang, H., Ma, H., 2020. Linformer: Self-attention with linear complexity. arXiv:2006.04768 .

Wang, X., Tu, Z., Wang, L., Shi, S., 2019b. Self-attention with structural position representations. arXiv:1909.00383 .

Xu, K., Hu, W., Leskovec, J., Jegelka, S., 2018. How powerful are graph neural networks? arXiv preprint arXiv:1810.00826 .

Ying, C., Cai, T., Luo, S., Zheng, S., Ke, G., He, D., Shen, Y., Liu, T.Y., 2021. Do transformers really perform badly for graph representation? Advances in Neural Information Processing Systems 34.

You, J., Ying, R., Leskovec, J., 2019. Position-aware graph neural networks, in: International Conference on Machine Learning, PMLR. pp. 7134–7143.

Yun, S., Jeong, M., Kim, R., Kang, J., Kim, H.J., 2019. Graph transformer networks. Advances in Neural Information Processing Systems 32.

Zhang, J., Zhang, H., Xia, C., Sun, L., 2020. Graph-BERT: Only attention is needed for learning graph representations. arXiv:2001.05140 .

A. Appendix: Task based MLP layer

This section describes the task based MLP layer mentioned in Section 3.1 for the respective datasets as follows.

ZINC dataset, graph-level task. We average the node features of a graph on the output of final layer, then get a d-dimensional graph-level feature h_{fG} as:

$$h_{fG} = \frac{1}{|\mathcal{V}|} \sum_{i=1}^{|\mathcal{V}|} h_i^L, \tag{15}$$

subsequently, this h_{fG} is passed to an MLP to produce pre-normalized prediction score for classes, $y_{pred} \in \mathcal{R}^C$, which is compued as:

$$y_{pred} = W_{f1} ReLU(W_{f2}, h_{fG}), \tag{16}$$

where $W_{f1} \in \mathcal{R}^{d \times C}$, $W_{f2} \in \mathcal{R}^{d \times d}$, C is one for ZINC dataset with single class. The loss function is the **L**1-loss, which minimizes the different between predicted value and the groundtruth value.

PATTERN, or CLUSTER, node-level task. Each node feature is passed to a MLP, and gets a pre-normalized prediction score, $y_{i,pred} \in \mathcal{R}^C$ as:

$$y_{i,pred} = W_{f1} ReLU(W_{f2}, h_i^L), \tag{17}$$

where W_{f1}, W_{f2} are the same training weights as above *graph-level task*. The loss function is weighted inversely cross-entropy loss based on the size of labels (classes).