Balancing Memory Efficiency and Accuracy in Spectral-Based Graph Transformers

by

Kelly Ho

S.B. in Electrical Engineering and Computer Science
Massachusetts Institute of Technology (2022)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2023

© 2023 Kelly Ho. All rights reserved.

The author hereby grants to MIT a nonexclusive, worldwide, irrevocable, royalty-free license to exercise any and all rights under copyright, including to reproduce, preserve, distribute and publicly display copies of the thesis, or release the thesis under an open-access license.

Authored by: Kelly Ho
Department of Electrical Engineering and Computer Science
August 11, 2023

Certified by: Arvind
Johnson Professor of Computer Science and Engineering
Thesis Supervisor

Accepted by: Katrina LaCurts
Chair, Master of Engineering Thesis Committee

Balancing Memory Efficiency and Accuracy in Spectral-Based Graph Transformers

by

Kelly Ho

Submitted to the Department of Electrical Engineering and Computer Science
on August 11, 2023, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

The transformer architecture has been a significant driving force behind advancements in deep learning, yet transformer-based models for graph representation learning have not caught up to mainstream Graph Neural Network (GNN) variants. A major limitation is the large $O(n^2)$ memory consumption of graph transformers, where n is the number of nodes. Therefore, we develop a memory-efficient graph transformer for node classification, capable of handling graphs with thousands of nodes while maintaining accuracy. Specifically, we reduce the memory use in the attention mechanism and add a random-walk positional encoding to improve upon the SAN graph transformer architecture. We evaluate our model on standard node classification benchmarks: Cora, Citeseer, and Chameleon. Unlike SAN, which runs out of memory, our memory-efficient graph transformer can be run on these benchmarks. Compared with landmark GNN models GCN and GAT, our graph transformer requires 27.92% less memory while being competitive in accuracy.

Thesis Supervisor: Arvind
Title: Johnson Professor of Computer Science and Engineering

Acknowledgments

Foremost, I would like to thank my MEng thesis supervisor Arvind and my project supervisor Dr. Jie Chen for mentoring me and providing me with this opportunity to work on this project. Their thoughtful insights and unwavering encouragement helped me work through obstacles and complete this project.

A big thank you to my research scientist mentor Xuhao Chen for his helpful guidance and constructive feedback on my ideas. Thank you to my UROP collaborator, Siddhant Mukherjee, for tackling this project with me.

To the other members of the Computation Structures Group, Tianhao Huang, Muhua Xu, and Michael Hadjiivanov, thank you for giving helpful suggestions at the lab meetings.

And finally, a heartfelt gratitude to my family and friends for the frequent reminders to stay on track and be focused. I appreciate the endless support for all my endeavors and thank you for making my college years an unforgettable experience.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Graphs are a natural representation for data with intricate relationships in the real world. Social networks [40, 18], biological networks [14, 18], and recommendation systems [15] are all examples of phenomenon that can be modeled as graphs. Within machine learning, most the models operating on graph structure data are graph neural networks (GNNs)[43]. Generally, this class of deep learning models iteratively update node representations based on neighboring nodes to obtain local and global patterns within the graph. However, long range dependencies are difficult for GNNs to understand as nodes that are connected through intermediary nodes have a weaker influence on one another [2]. Over-smoothing can also occur when information propagates too broadly during message passing [33]. For tasks where the global graph structure plays a crucial role, this can be problematic. Furthermore, GNNs have limited interpretability because the intricate transformations within the layers make it difficult to comprehend the specific nodes and edges that contribute to the decision making [45].

As a result, another graph architecture framework, the graph transformer is gaining popularity. Transformers have gained notable achievements in natural language processing (NLP) [9, 3, 28] and computer vision (CV) [21], so it is logical to explore their potential for graphs. By leveraging the self-attention mechanism, the transformer can selectively attend to relevant nodes and features, which preserves discriminative information and prevents over-smoothing. The attention module as-

signs weights to nodes and edges based on their relevance for a certain task, which allows the model to be interpretable and therefore determines trustworthiness.

Although graph transformers have many benefits, one major drawback is that graph transformers are not easily scalable [32]. The computation complexity increases quadratically as the number of nodes increase due to updating the attention scores for each node. On the other hand, because GNNs often include message passing schemes [35], when the number of nodes increases linearly, the computational complexity also increases linearly. Thus, we aim to explore methods of improving the scalability of graph transformers while maintaining the overall accuracy.

The Spectral Attention Network (SAN) [26] is graph transformer that produces competitive results for node classification, graph classification, and link prediction benchmarks. It exhibits a significant performance advantage over other attention based models. SAN employs graph spectral theory, which analyzes eigenvalues and eigenvectors of matrices associated with the graphs and encodes relevant positional encoding information to node features. The models were evaluated on node classification for graphs that typically have 100-200 nodes. But, when applied to graphs with thousands or tens of thousands of nodes, SAN exhausts the memory resources of the GPU.

Hence, our objective is to develop a memory efficient graph transformer for node classification for graphs on the order of thousands of nodes while preserving their accuracy. Our models utilize random walk landing probabilities as each node's positional encoding as well as eliminate redundant components of the self attention module. Our evaluation on the Cora [27], Citeseer [16] and Chameleon [30] benchmarks showed a significant decrease in memory consumption while maintaining competitive accuracy.

Chapter 2

Related Works

2.1 Graph Neural Networks

Graph neural networks (GNNs) [34, 42, 47] aim to propagate information across graph nodes using neural network layers. A key component of GNNs is representing graph structure information through a message passing scheme. The graph nodes iteratively update their representations by exchanging information with neighboring nodes [42]. While there are many variations of this idea, these architectures often have an over-smoothing and over-squashing problem, which makes it difficult for the graph to learn deep representations and propagate information to distant nodes [17].

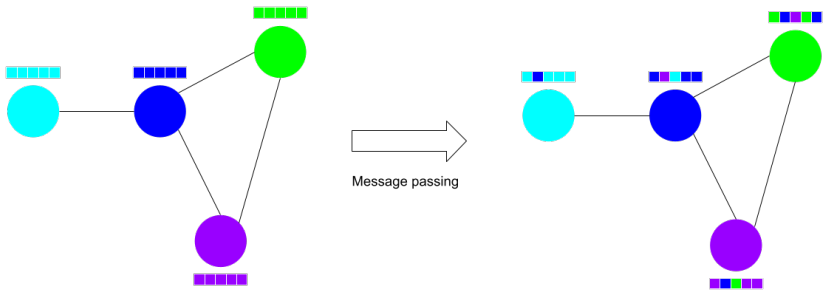

Figure 2-1: One iteration of message passing for graphs

2.1.1 Graph Convolutional Network

Graph Convolutional Network (GCN) [25] is a semi-supervised neural network architecture for graph structured data. To encode graph structures, GCN takes advantage of first-order approximations of spectral graph convolutions. We display this in eq. (2.1), where $h_j^{(l)}$ = node j's feature in layer l, $W^{(l)}$ = convolution weight matrix for layer l, $\mathcal{N}(i)$ = set of node i's neighbors, normalization constant $c_{ij} = \sqrt{|\mathcal{N}(i)|}\sqrt{|\mathcal{N}(j)|}$, and σ=activation function (GCN uses ReLU). Give these values, we can compute the node features $h_i^{(l+1)}$ for the next layer $(l+1)$.

$$h_i^{(l+1)} = \sigma\left(\sum_{j \in \mathcal{N}(i)} \frac{1}{c_{ij}} W^{(l)} h_j^{(l)}\right) \tag{2.1}$$

This layer-wise propagation rule allowed GCN to outperform other models released at the time on citation networks and knowledge graphs. Since the advent of GCN, future works have built upon this foundation. GraphSage [19] uses the same update rule in eq. (2.1) but replace $c_{ij} = |\mathcal{N}(i)|$.

2.1.2 Graph Attention Network

The Graph Attention Network (GAT) [38] is a landmark model that utilizes masked self-attention layers that allows different weights to be specified for each node in a neighborhood. Specifically, it replaces the convolution in the update rule of GCN [25] with an updated rule using attention as seen in eq. (2.2). We use the embeddings of layer l to calculate $h_i^{(l+1)}$, node i's embedding in layer $l+1$. See Figure 2-2 as an illustration of this attention-based update rule described in equation 2.2.

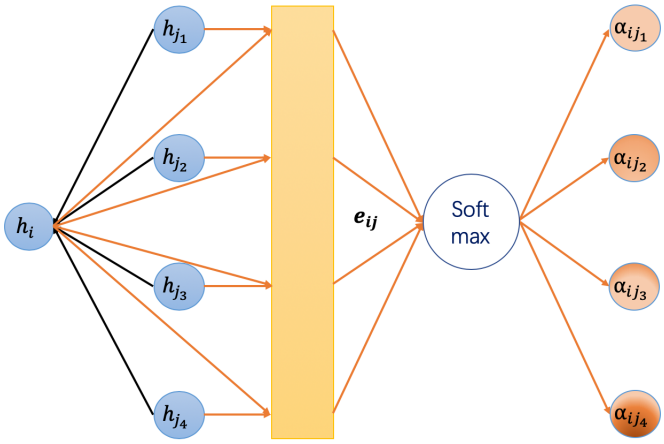

Figure 2-2: Illustration of attention mechanism in Graph Attention Networks (GAT). Equation 2.2 and figure are borrowed from [46]

$$
\begin{aligned}
z_i^{(l)} &= W^{(l)} h_i^{(l)}, \\
e_{ij}^{(l)} &= \text{LeakyReLU}\left(\vec{a}^{(l)^T}\left(z_i^{(l)} \| z_j^{(l)}\right)\right), \\
\alpha_{ij}^{(l)} &= \frac{\exp\left(e_{ij}^{(l)}\right)}{\sum_{k \in \mathcal{N}(i)} \exp\left(e_{ik}^{(l)}\right)}, \\
h_i^{(l+1)} &= \sigma\left(\sum_{j \in \mathcal{N}(i)} \alpha_{ij}^{(l)} z_j^{(l)}\right),
\end{aligned}
\tag{2.2}
$$

First, $z_i^{(l)}$ is computed from a linear transformation of embedding $h_i^{(l)}$ and learnable weight matrix $W^{(l)}$. Then, we calculate the unnormalized attnetion score $e_{ij}^{(l)}$ between nodes i and j by concatenating ($||$) the embeddings $z_i^{(l)}$ and $z_j^{(l)}$, taking the dot product with learnable weight vector $\vec{a}^{(l)^T}$, and then applying a LeakyReLU function at the end. Afterwards, we normalize the attention score through the softmax function to yield normalized attention scores between two nodes $\alpha_{ij}^{(l)}$. Finally, to calculate $h_i^{(l+1)}$, node i's embedding in layer $l+1$, we take a weighted sum of the embeddings from neighbors $z_j^{(l)}$ based on the attention scores $\alpha_{ij}^{(l)}$ and normalize with the sigmoid function.

2.2 Limitations of GNNs

Although graph neural networks have shown great promise in graph-based learning tasks, they also come with certain limitations. GNNs heavily rely on the graph structure to perform well [39]. So, if the graph structure changes significantly or if new nodes or edges are added, the models may require retraining, making them less adaptive to dynamic graphs. Furthermore, GNNs often have a limited receptive field and may not capture the global context of the entire graph effectively [41]. This can be a limitation for tasks that require understanding the broader relationships in the graph.

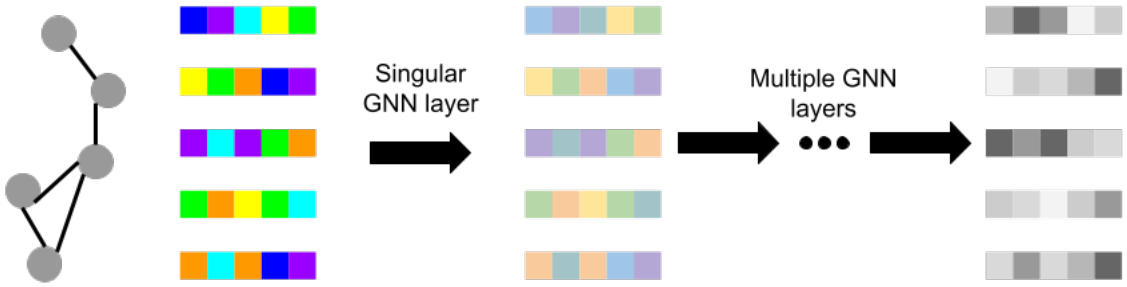

Figure 2-3: The over-smoothing problem: as the model gets deeper, node features become similar everywhere.

In deep Graph Neural Network architectures, information can be excessively smoothed out across multiple layers, resulting in the loss of discriminative features for node classification tasks [5, 24]. Generally, information is propagated between neighboring nodes through message passing mechanisms. And at each layer, a node aggregates features from its neighbors. As the layers of the GNN increase, nodes aggregate information from their neighbors repeatedly. Over time, this can lead to the blending of node features, causing nodes that were initially dissimilar to become more alike [4]. As a result, the model loses the ability to distinguish between nodes that should have different representations, leading to diminished classification performance.

Additionally, graph neural networks (GNNs) with message passing based schemes have node features which traverse the input graph. It has been highlighted that the

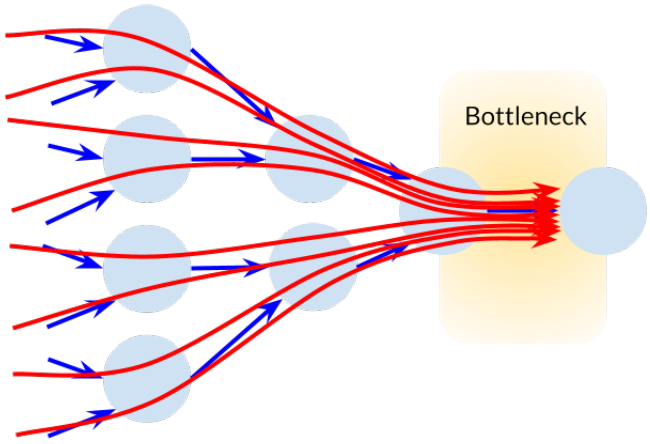

Figure 2-4: The over-squashing problem [36] [2]: information from a node's exponentially-growing receptive field is compressed into a fixed-size vector. Black arrows are graph edges; red curved arrows illustrate information flow.

effectiveness of message passing for tasks involving distant interactions is hindered by the distortion of information from faraway nodes [36]. The over-squashing phenomenon, is due to the presence of graph bottlenecks; the number of k-hop neighbors dramatically increases with the increase of k [2].

Transformers offer a resolution to these limitations. They are global context aware and less susceptible to over-smoothing due to the lack of message passing scheme.

2.3 Graph Transformers

Transformers [37] have had great success in language [8, 31] and vision [20, 10] because it is adept at modeling sequential data and represent long range temporal dependencies [7]. However, graph transformers have not yielded as high performance as other graph neural networks in common graph representation datasets [44, 23, 22]. A challenge specific to graphs is how to encode graph structure and node position. Unlike sentences, where each word has a position within a sequence, nodes in a graph are not inherently ordered. Multiple techniques provide the transformer with information about the structural relationships between nodes. These include utilizing the eigenvectors of the graph Laplacian [11], pair-wise graph distances [44, 1], and relative

positional encoding [29].

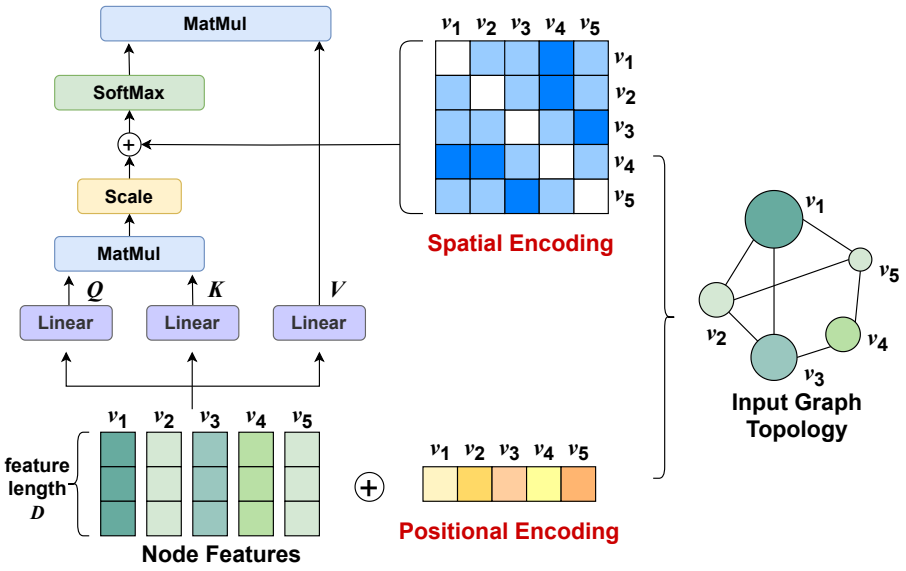

Figure 2-5: An example of graph transformer architecture; borrowed from [44]

Graphormer [44] (see fig. 2-5) proposes three structural encoding methods to help the transformer understand the structural information of graphs. Specifically, it adds a centrality encoding to signify which node is important, a spatial encoding to represent spatial relation such as distance between two nodes, and an edge encoding that can propagate information through the whole graph. Another architecture, GraphGPS [32], combines positional and structural encoding, local message-passing, and global attention and is able to achieve linear complexity in the number of nodes and edges $O(N + E)$. We focused on a class of graph transformers with spectral attention because its mechanisms can serve as a form of graph regularization [6]. They implicitly encouraging the model to attend to relevant neighbors and suppress noise or irrelevant information during feature aggregation.

2.3.1 Spectral Attention Network

The Spectral Attention Network (SAN) [26] (see fig. 2-6) tackles the lack of an explicit graph structure with a learned positional encoding (LPE) [12] that determines the position of nodes by Laplacian eigen-functions across the full spectrum of a graph. In

theory, by concatenating the LPE to the node features and utilizing a fully connected graph, the model should have improvements in differentiating graphs and recognizing similar sub-structures [26]. The fully-connected property allows the transformer to avoid the over-squashing problem that exist for many GNNs.

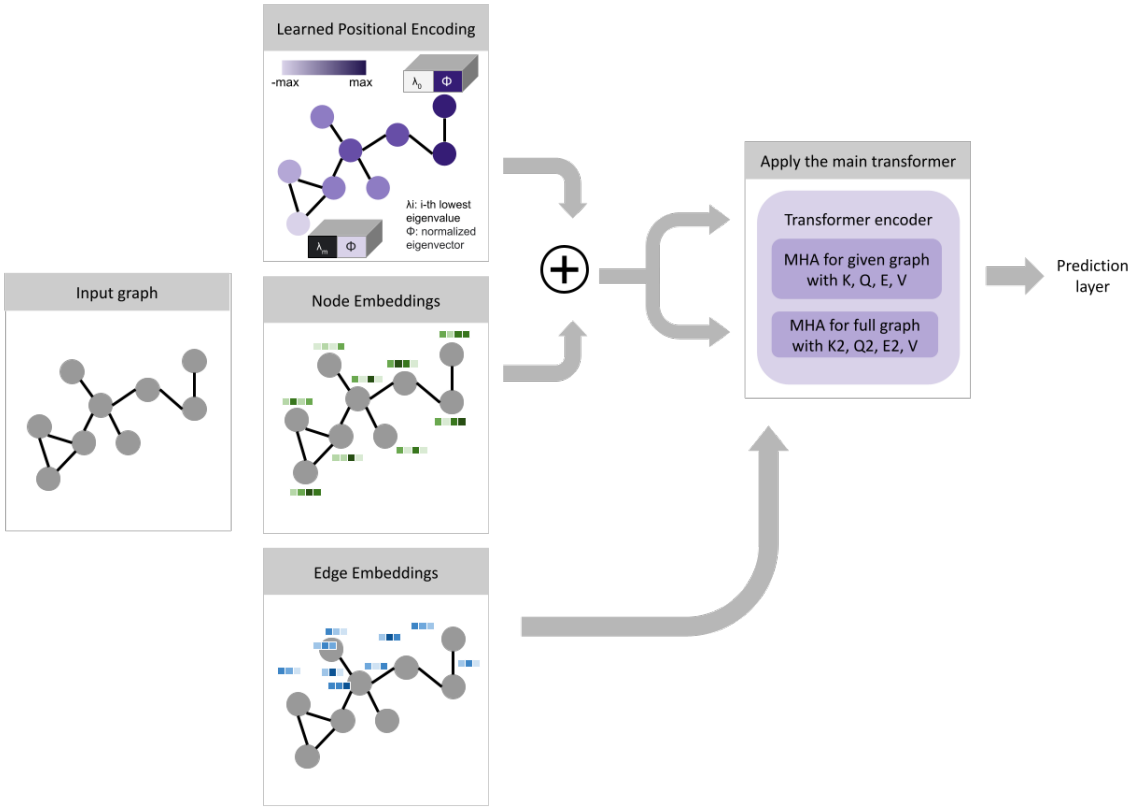

Figure 2-6: The Spectral Attention Network (SAN) graph transformer architecture [26]

Our attention mechanism in the main Transformer is based on SAN [26] because the model combines the key, query, and value of graphs with "real" and "fake" edges. However, it cannot evaluate graphs with more than thousands of nodes. The attention mechanism combines the key and query linear layers in a manner that results large intermediary matrices. Therefore, our model attempts to re-purpose the original Transformer to graphs by considering the graph structure and improving attention estimates with edge feature embeddings.

Chapter 3

Methodology: Redesigning Model Architecture

After carefully weighing the merits and drawbacks of the preceding models, we designed model guided by SAN that reduces memory consumption while preserving accuracy. fig. 3-1 illustrates our proposed graph transformer architecture. In the following section, we discuss the details of this architecture design.

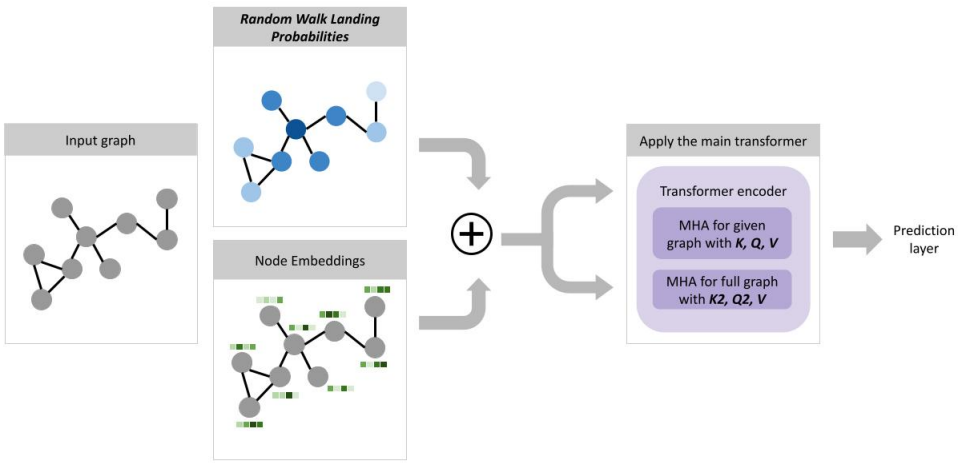

Figure 3-1: Our proposed graph transformer architecture.

3.1 Reducing Memory Consumption

Graph transformers are difficult to scale as the computational complexity increases quadratically as nodes increase linearly. Specifically, the propagate attention module in SAN causes datasets with graphs on the order of thousands of nodes to run out of memory. For example, SAN was unsuccessful in running CORA, a citation network with 2708 nodes, on a NVIDIA V100 32GB GPU due to the lack of memory. In the self-attention block of the original architecture, functions was applied to all the edges to update node features was the root cause of the memory shortage.

SAN's self-attention module combines four components — key (K), query (Q), value (V), and edge (E). The keys and queries are first multiplied element wise and then the available edge features multiplied element wise as well to modify the self-attention weights. Joining the elements in this manner creates the need to store intermediary data that will eventually collapsed. Another issue is that the target datasets do not have edge features. Thus, the only differentiating factor is whether the edge is "real" or "fake." To reduce the memory footprint, we explored two new implementations of self-attention.

3.1.1 Iteration 1: Removing Redundant Edge Embeddings

In the first iteration of strategies to circumvent the edge function application problem, we only kept track of two edge embeddings — real or fake edge embeddings — and combined the components in a different order. Thus, we was able to compress the E matrix shape from $(N(N-1) \times H \times D)$ to $(2 \times H \times D)$ where N is the number of nodes, H is the number of heads and D is output dimensions for the self-attention module. By performing element wise multiplication on the K and E first and then combining it with Q afterwards, the self-attention block significantly reduced the number of intermediary computations.

3.1.2 Iteration 2: Removing All Edge Data for Better Performance

For the second iteration, the edge embeddings were completely removed. Thus, the only element wise multiplication was between the key and query matrices. Instead of updating node features by function applications for each edge, K and Q were combined via matrix multiplication. Removing the edge embeddings completely showed major improvements compared to keeping embeddings for real and fake edges. Both versions the memory efficient spectral based graph transformer were able to run on a singular NVIDIA 16GB V100 GPU.

3.1.3 Main Graph Transformer

The attention block leverages the ideas behind the classic transformer architecture and is adapted to graph structured data. Let n_i^l be the features of the i-th node features in the l-th layer and $d_h = \frac{d}{H}$ is the hidden dimension d of a head. The equation below applies SAN's multi-head attention over the nodes in the graph.

$$\hat{n}_i^{l+1} = O_n^l \Big\Vert_{h=1}^{H} (\sum_{j \in V} w_{ij}^{h,l} V^{h,l} n_j^l) \tag{3.1}$$

where $O_n^l \in \mathbb{R}^{d \times d}, V^{h,l} \in \mathbb{R}^{d_h \times d}$. H is the number of heads while $\|$ represents concatenation.

To maintain the difference between edges that are originally connected (real) and those that are linked due to construct a full graph (fake), the attention weights $w_{ij}^{k,l}$ for layer l and head h are defined as follows:

$$\hat{w}_{ij}^{k,l} = \begin{cases} \dfrac{Q^{1,h,l} n_i^l \circ K^{1,h,l} n_i^l}{\sqrt{d_h}} & \text{if i and j are connected by real edge} \\ \dfrac{Q^{2,h,l} n_i^l \circ K^{1,h,l} n_i^l}{\sqrt{d_h}} & \text{otherwise} \end{cases} \tag{3.2}$$

$$w_{ij}^{k,l} = \begin{cases} \frac{1}{1+\gamma} \cdot \text{softmax}(\sum_{d_h} \hat{w}_{ij}^{h,l}) & \text{if i and j are connected by real edge} \\ \frac{\gamma}{1+\gamma} \cdot \text{softmax}(\sum_{d_h} \hat{w}_{ij}^{h,l}) & \text{otherwise} \end{cases} \tag{3.3}$$

where $Q^{1,h,l}, Q^{2,h,l}, K^{1,h,l}, K^{2,h,l} \in \mathbb{R}^{d_j \times d}$ and $\circ$ represents element-wise multiplication. Q^1 and K^1 are the queries and keys for all pairs of nodes with a "real" or connected edge while Q^2 and K^2 are "fake" or disconnected edges. The hyperparameter γ controls the degree of full graph attention as different datasets require varying amounts of long range dependencies. To mitigate the exploding gradients problem, the softmax outputs are clipped to a value between -5 and 5.

3.2 Improving Accuracy

After establishing a model that worked within the memory constraints of a single GPU, we addressed parts of the architecture that potentially negatively impacted the accuracy.

3.2.1 Positional Encoding

To encode graph structure into transformers, many models leverage the eigenvalues and eigenvectors of the graph Laplacian. SAN, in particular, takes the m smallest normalized eigenvalues and eigenvectors where m is a tuned hyperparameter. After a linear layer transforms these values, a multi-layer transformer encoder is a applied. The resulting learned positional encoding (LPE) is then included in the node embeddings of the graph. While this approach works well for graphs on the order of hundreds of nodes that is a singular connected component, it causes mismatched magnitudes for eigenvalues and eigenvectors for the target datasets.

No Positional Encoding

Through informal experiments of removing different combinations of the linear layer, transformer, and eigenvalues of the positional encoding in Section 3.2.3, it was determined that SAN performed better after the positional encoding block was completely

removed. As a result, my modifications aligned with the findings. The absence of a positional encodings for nodes does not does not equate to the lack of graph structure information in a model though. The location of each node within a graph can be determined through the differences between the sparse graph with all real edges and the full graph that includes the fake edges as well.

Random Walks

Another alternative to the Laplacian based positional encoding is to use random walk (RW) landing probabilities to encode graph structure [13]. This RW diffusion process leverages the chance a walk lands on a specific node given a specific number of steps. Experimentally, it was proven that positional encoding based on a random walk scheme performs better than a graph Laplacian scheme [13]. Interestingly, based on experiments results (Section 4.3), the accuracy is similar for models with no positional encoding and utilizing random walk probabilities.

3.2.2 Multi-Layer Perceptron Readout Layer

After updating the weights of the nodes via the graph transformer layers, the multi-layer perceptron (MLP) readout layer reduces each node's embeddings from the output dimensions into the number of classes. Theoretically, running multiple linear and ReLU on a matrix to reduce the vector size of each node embedding should assist with the representation power of the weights as well as the vanishing gradients issue. Contrary to expectations, this causes the resulting values in the vector to be indiscriminate and therefore bias towards choosing the first class. Thus, the model applies a singular linear and ReLU function to alleviate this issue.

3.2.3 Ineffective Strategies

Through the process of refining the architecture of the new model, there were a few notable changes that appeared promising but were ultimately ineffective. After discovering the mismatched eigenvalue and eigenvector magnitude problem, we wanted

to determine whether a singular component of the original SAN positional encoding scheme was the issue. Note: all of the following architecture changes are independent of one another.

First, we removed the eigenvalues because experimentally, they were 10 magnitudes smaller than the eigenvectors. The average accuracy only increased approximately 3% and was still significantly below the results of landmark models such as GAT and GCN.

Another segment within this scheme was the linear layer. We wanted to check whether the inputs to the PE Transformer required a more complex and nonlinear transformation, so we changed the linear to a multi-layer perceptron (MLP) layer. The changes were not notable; in fact, there was a $<1\%$ drop in accuracy.

We also removed the PE Transformer to simplify the architecture and it produced a $<1\%$ improvement. Thus, we concluded that the numerical instability of the eigenvalues and eigenvectors cannot be attributed to one specific part of the graph Laplacian scheme.

Chapter 4

Experiments

We validated the model on a number of node classification benchmarks and evaluated accuracy, space, and time metrics.

4.1 Benchmarks

The benchmarks Cora [27], Citeseer [16] and Chameleon [30] were chosen as they were node classification tasks with a few thousand nodes. SAN was evaluated on graphs that were tens to hundreds of nodes, thus we determined that an increase of magnitude would be suitable.

Cora Dataset The Cora dataset is a citation network benchmark dataset for node classification tasks. The dataset consists of nodes represented by scientific papers belonging to different research topics and edges represent citations between papers. Each publication has 1433 features that are unique binary word vectors — 0 or 1 depending on the presence of the word within the paper. The goal is to classify each paper into 1 of 7 topics based on its features and the citation relationships between papers.

CiteSeer Dataset The CiteSeer dataset also represents a citation network and is very similar to Cora as each publication is represented as a node in the graph, and the

edges denote the citation relationships between papers. There are 3703 0/1-valued word vectors as features and each publication is assigned to 1 of 6 research areas for node classification.

Chameleon Dataset Chameleon is a page-to-page network where its 2,277 nodes are Wikipedia articles about chameleons and its edges represent mutual links between the articles. The node features indicate the presence of nouns in the articles. The task is to classify the nodes into 5 classes in terms of their average monthly traffic.

The following table is a summary of benchmark attributes.

	Cora	CiteSeer	Chameleon
# of Nodes	2708	3312	2277
# of Edges	5429	4732	36101
# of Features	1433	3703	2325
# of Classes	7	6	5

Table 4.1: Benchmark properties

4.2 Experiment Methodology

To evaluate our model, we compare it to two landmark models: GCN and GAT. As each model implementation tunes their hyperparameter and architectures, we standardize the neural network pipeline and specifically change the graph transformer layer. By comparing the landmark models' results with our Graph Transformer (GT) layer, we are able to determine whether our results are competitive. We run four different models on the benchmarks:

1. Model with two GT layers and random walk probabilities as positional encoding.
2. Model with two GT layers and no positional encoding.
3. Model with two GCN layers

4. Model with two GAT layers

To ensure that these models are compared equally, we use the same hyperparameters. For example, we standardized the number of epoch by setting a limit 1000 epochs. To ensure reproducibility, we specify a seed for each of the runs. The results are the average of 15 random seeds.

4.3 Results

The following tables summarize the average runs of each of four models on the space, accuracy, and time consumption metrics. First, we look at the average memory consumption for each of these models as our main goal was to reduce the space (in GB) required for graph transformers.

	Cora	**Citeseer**	**Chameleon**
Ours w/ PE	11.4751 ± 0.00	9.2920 ± 0.00	12.8026 ± 0.00
Ours w/o PE	11.5737 ± 0.00	9.3905 ± 0.00	12.8047 ± 0.00
GCN	15.7596 ± 0.00	15.6799 ± 0.00	15.7596 ± 0.00
GAT	15.5058 ± 0.00	15.2395 ± 0.00	15.4450 ± 0.00
SAN	OoM	OoM	OoM

Table 4.2: Total Space (GB) utilized by GPU to evaluate benchmark. PE: Positional Encoding

Notably, SAN resulted an out of memory error when run on these datasets, which was the intial premise for modifying the model. Our models with PE and without PE, required a similar amount of space. But, it is unexpected to see that the model without positional encoding needed more memory as space was required to store the extra PE information. Between runs, the memory consumption stayed exactly the same, which is expected.

The models with random walk landing probabilities and no positional encoding consistently require less GPU memory compared to the GCN and GAT models for all

three benchmarks (Cora, Citeseer, and Chameleon). In general, graph transformers tend to consume more memory compared to traditional GNNs because graph neural networks typically operate on a fixed-size neighborhood around each node and use message passing mechanisms to aggregate information from neighboring nodes. Thus, it is promising that our model demonstrates a reduced space requirement compared to GCN while maintaining competitive accuracy.

	Cora	**Citeseer**	**Chameleon**
Ours w/ PE	0.7862 ± 0.0169	0.6330 ± 0.0101	0.6304 ± 0.0093
Ours w/o PE	0.7376 ± 0.0152	0.6375 ± 0.0093	0.6308 ± 0.0132
GCN	0.7794 ± 0.0016	0.6276 ± 0.0030	0.5746 ± 0.0052
GAT	0.7770 ± 0.0153	0.6410 ± 0.0081	0.6029 ± 0.0285

Table 4.3: Node Classification Accuracy. Proportion of nodes with correct accuracy. PE: Positional Encoding; OoM: Out of memory.

Through our experiments, our model with random walk landing probabilities tend to do better or similar to models with no positional encoding. Each of the datasets saw different accuracy performances. For Cora, the random walk model had higher accuracy than both GCN and GAT. On the other hand, Citeseer only performed better than GCN. Both of our models performed better than GCN and GAT for Chameleon. And for all three datasets, our models performed better than GCN by 2.91% and 0.58% better than GAT. Thus, we can conclude that our random walk model accuracy is generally competitive with GCN and GAT.

Interestingly, the results for our models with PE and without PE for both Chameleon and Citeseer are quite similar but very different for Cora. This could be attributed to Cora having the greatest number of classes but the least amount of features. Last but not least, we look at the inference time (in seconds) for each of the models.

	Cora	Citeseer	Chameleon
Ours w/ PE	156.39 ± 0.63	230.76 ± 4.28	74.16 ± 4.73
Ours w/o PE	155.80 ± 0.98	229.25 ± 2.73	72.82 ± 3.28
GCN	45.64 ± 0.97	61.15 ± 0.76	90.31 ± 0.27
GAT	163.59 ± 5.30	338.18 ± 5.84	352.77 ± 88.92

Table 4.4: Number of seconds to evaluate benchmark. PE: Positional Encoding

Within our models, the presence of a positional encoding with random walk increased the time, but not significantly. Both our models took less time than GAT. Two of the benchmarks took more time than GCN, which is expected as the localized nature of GNN operations allows for better parallelism, enabling faster computations on GPUs.

For the Chameleon benchmark, our models did better – perhaps due to the nature of the task or the smaller number of classes and nodes. And even though Chameleon has a much larger number of edges compared to Cora and Citeseer, the lack of edge embeddings for our model allowed the inference time to be decreased significantly compared to the other graph transformer benchmark, GAT. GAT had a much higher standard deviation than all other models which means the time it takes for inference tasks has high variance.

In summary, the model significantly saves memory consumption compared to two landmark graph networks while achieving competitive accuracy for Cora, Citeseer, and Chameleon benchmarks.

Chapter 5

Conclusion

In this work, we designed a new graph transformer (GT) architecture that reduces memory while maintaining a competitive accuracy. The architecture decreases memory consumption by strategically removing edge embeddings. We then incrementally refine the architecture by adjusting various sections of the model to improve accuracy. Specifically, we use random walks landing probabilities for positional encoding, and applied a singular linear layer within the multi-layer perceptron readout layer. This design allows the model to leverage nodes of higher importance based on frequency of visit in the graph. The modified MLP readout layer prevents the graph from over-fitting on dataset as it decreases the model's ability to memorize noise.

We evaluate our proposed GT architecture using three benchmark datasets: Cora, CiteSeer, and Chameleon, on which an existing GT architecture SAN runs out of memory. We compared the prediction accuracy of our design against representative GNN models, GCN and GAT. Specifically, we found that our proposed GT architecture with random walk positional encoding achieves competitive accuracy, but consumes 28.68% and 27.16% less memory than GCN and GAT, respectively. We were able to maintain a competitive accuracy with a 2.91% and 0.58% increase in node classification correctness for GCN and GAT, respectively. In terms of training speed, our model is 38.58% faster than GAT but 166.51% slower than GCN.

This study demonstrates that memory-efficient graph transformers, which is important for real-world applications, can yield competitive accuracy. The exciting

results can encourage exploration of more advanced GT architectures and enable training on massive-scale graph data to further improve model accuracy.

Chapter 6

Future Work

There are several potential avenues for further research and areas of improvement to enhance the model's performance. We hope to explore other alternatives to random walks landing probabilities and the graph Laplacian for positional encoding. An improved positional encoding scheme would not only allow for better presentation of the graph structure, but also improve the attention mechanism. Furthermore, we aim to run our experiments on benchmarks with increased number of nodes. Graphs datasets that have a magnitude greater of nodes include Computer, Pubmed, and Physics with 13k, 20k, and 34k nodes, respectively. We also seek to compare datasets with homophily and heterophily graphs; homophily refers to the tendency of edges in a network to connect similar nodes.

Our model has the potential to be a building block for future model architecture and extend its applicability to real-world scenarios. Current graph transformers are not feasible for node classification on graphs in the range of 100k - 1 million nodes on a singular GPU due to memory limitations. Thus, a next step would be to use this architecture as the starting point for creating a model that fits within those requirements.

Bibliography

[1] Wasi Uddin Ahmad, Nanyun Peng, and Kai-Wei Chang. Gate: graph attention transformer encoder for cross-lingual relation and event extraction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 12462–12470, 2021.

[2] Uri Alon and Eran Yahav. On the bottleneck of graph neural networks and its practical implications, 2021.

[3] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020.

[4] Chen Cai and Yusu Wang. A note on over-smoothing for graph neural networks, 2020.

[5] Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, and Xu Sun. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 3438–3445, 2020.

[6] Huiyuan Chen, Lan Wang, Yusan Lin, Chin-Chia Michael Yeh, Fei Wang, and Hao Yang. Structured graph convolutional networks with stochastic masks for recommender systems. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 614–623, 2021.

[7] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019.

[8] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[10] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[11] Vijay Prakash Dwivedi and Xavier Bresson. A generalization of transformer networks to graphs. *arXiv preprint arXiv:2012.09699*, 2020.

[12] Vijay Prakash Dwivedi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Graph neural networks with learnable structural and positional representations. *arXiv preprint arXiv:2110.07875*, 2021.

[13] Vijay Prakash Dwivedi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Graph neural networks with learnable structural and positional representations, 2022.

[14] Alex Fout, Jonathon Byrd, Basir Shariat, and Asa Ben-Hur. Protein interface prediction using graph convolutional networks. *Advances in neural information processing systems*, 30, 2017.

[15] Chen Gao, Xiang Wang, Xiangnan He, and Yong Li. Graph neural networks for recommender system. In *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*, pages 1623–1625, 2022.

[16] C. Lee Giles, Kurt D. Bollacker, and Steve Lawrence. Citeseer: An automatic citation indexing system. In *Proceedings of the Third ACM Conference on Digital Libraries*, DL '98, page 89–98, New York, NY, USA, 1998. Association for Computing Machinery.

[17] Jhony H. Giraldo, Fragkiskos D. Malliaros, and Thierry Bouwmans. Understanding the relationship between over-smoothing and over-squashing in graph neural networks, 2022.

[18] Michelle Girvan and Mark EJ Newman. Community structure in social and biological networks. *Proceedings of the national academy of sciences*, 99(12):7821–7826, 2002.

[19] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs, 2018.

[20] Kai Han, Yunhe Wang, Hanting Chen, Xinghao Chen, Jianyuan Guo, Zhenhua Liu, Yehui Tang, An Xiao, Chunjing Xu, Yixing Xu, et al. A survey on vision transformer. *IEEE transactions on pattern analysis and machine intelligence*, 45(1):87–110, 2022.

[21] Kai Han, Yunhe Wang, Hanting Chen, Xinghao Chen, Jianyuan Guo, Zhenhua Liu, Yehui Tang, An Xiao, Chunjing Xu, Yixing Xu, Zhaohui Yang, Yiman Zhang, and Dacheng Tao. A survey on vision transformer. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(1):87–110, jan 2023.

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

[23] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *Advances in neural information processing systems*, 33:22118–22133, 2020.

[24] Nicolas Keriven. Not too little, not too much: a theoretical analysis of graph (over)smoothing, 2022.

[25] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2017.

[26] Devin Kreuzer, Dominique Beaini, Will Hamilton, Vincent Létourneau, and Prudencio Tossou. Rethinking graph transformers with spectral attention. *Advances in Neural Information Processing Systems*, 34:21618–21629, 2021.

[27] Andrew Kachites McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. Automating the construction of internet portals with machine learning. *Information Retrieval*, 3(2):127–163, Jul 2000.

[28] OpenAI. Gpt-4 technical report, 2023.

[29] Wonpyo Park, Woonggi Chang, Donggeon Lee, Juntae Kim, and Seung-won Hwang. Grpe: Relative positional encoding for graph transformer. *arXiv preprint arXiv:2201.12787*, 2022.

[30] Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. Geom-gcn: Geometric graph convolutional networks, 2020.

[31] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

[32] Ladislav Rampášek, Mikhail Galkin, Vijay Prakash Dwivedi, Anh Tuan Luu, Guy Wolf, and Dominique Beaini. Recipe for a general, powerful, scalable graph transformer, 2023.

[33] T. Konstantin Rusch, Michael M. Bronstein, and Siddhartha Mishra. A survey on oversmoothing in graph neural networks, 2023.

[34] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

[35] Yunsheng Shi, Zhengjie Huang, Shikun Feng, Hui Zhong, Wenjin Wang, and Yu Sun. Masked label prediction: Unified message passing model for semi-supervised classification, 2021.

[36] Jake Topping, Francesco Di Giovanni, Benjamin Paul Chamberlain, Xiaowen Dong, and Michael M. Bronstein. Understanding over-squashing and bottlenecks on graphs via curvature, 2022.

[37] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[38] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2018.

[39] Ruijia Wang, Shuai Mou, Xiao Wang, Wanpeng Xiao, Qi Ju, Chuan Shi, and Xing Xie. Graph structure estimation neural networks. In *Proceedings of the Web Conference 2021*, WWW '21, page 342–353, New York, NY, USA, 2021. Association for Computing Machinery.

[40] Yongji Wu, Defu Lian, Yiheng Xu, Le Wu, and Enhong Chen. Graph convolutional networks with markov random field reasoning for social spammer detection. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 1054–1061, 2020.

[41] Zhanghao Wu, Paras Jain, Matthew A. Wright, Azalia Mirhoseini, Joseph E. Gonzalez, and Ion Stoica. Representing long-range context for graph neural networks with global attention, 2022.

[42] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24, 2020.

[43] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, 2021.

[44] Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. Do transformers really perform bad for graph representation?, 2021.

[45] Hao Yuan, Haiyang Yu, Shurui Gui, and Shuiwang Ji. Explainability in graph neural networks: A taxonomic survey, 2022.

[46] Hao Zhang, Mufei Li, Minjie Wang, and Zheng Zhang. Understand graph attention network.

[47] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI open*, 1:57–81, 2020.