Reproduction of a Paper: "How Attentive Are Graph Attention Networks?"

Vladimir Radenković
Signal Processing Department
School of Electrical Engineering, University of Belgrade
Belgrade, Serbia, 11000
`vladimirradenkovic27@gmail.com`

1 Introduction

Graph Attention Networks (GATs) [2] are a widely used Graph Neural Netowrk (GNN) architectures that has achieved state-of-the-art performance in graph representation learning. The paper "How Attentive Are Graph Attention Networks" [1] argues that GAT computes very limited kind of attention which authors formally define as *static* attention. This *static* attention mechanism can limit the ability of GATs to express more complex relationships in graphs, which motivates the development of a more expressive attention mechanism. To address this limitation, the paper proposes GATv2, a *dynamic* graph attention variant that modifies the order of operations to enable a strictly more expressive attention mechanism.

The objective of this project is to implement both GAT and GATv2 models, compare dynamic and static attention mechanisms and replicate the experiments used in the paper for the evaluating their performance. This includes providing descriptions of the model implementations, datasets, and experiment settings and aims, as well as reproducing the experimental results in a consistent environment.

Through this project, the goal is to gain a deeper understanding of the theoretical workings of graph attention networks, as well as practical experience in implementing deep learning methods on graph data. Furthermore, this project provides a valuable opportunity to gain insights from reproducing and validating the findings of a recent and influential paper in the field of graph representation learning.

2 Model Overview and Theoretical Background

A directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ contains nodes $\mathcal{V} = \{1, ..., n\}$ and edges $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ where $(j, i) \in \mathcal{E}$ denotes edge from node *j* to a node *i*. It is assumed that every node $i \in \mathcal{V}$ has an initial representation $\mathbf{h_i}^{(0)} \in \mathbb{R}^{d_0}$. An undirected graph can be represented with bidirectional edges.

2.1 Graph Neural Networks

A graph neural network(GNN) layer updates every node representation by aggregating the representations of its neighboring nodes. The input of a layer is a set of node representations $\{\mathbf{h_i} \in \mathbb{R}^d | i \in \mathcal{V}\}$ and the set of edges $\mathcal{E}$. The layer outputs a new set of node representations $\{\mathbf{h_i}' \in \mathbb{R}^{d'} | i \in \mathcal{V}\}$, where the same parametric function is applied to every node given its neighbors $\mathcal{N}_i = \{j \in \mathcal{V} | (j, i) \in \mathcal{E}\}$:

$$\mathbf{h_i}' = f_\theta(\mathbf{h_i}, \text{aggregate}(\{\mathbf{h_j} | j \in \mathcal{N}_i\}) \qquad (1)$$

The primary distinction between various GNN types lies in the design of the *f* and *aggregate* functions.

2.2 Graph Attention Networks

Many popular GNN architectures weigh all neighbors $j \in \mathcal{N}_i$ with equal importance by using mean or max-pooling as aggregate function. To address this limitation, the aggregate function of GAT computes learned weighted average of the representations of $\mathcal{N}_i$. A scoring function $e : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ computes a score for every edge (j, i) which indicates the importance of the features of the neighbor *j* to the node *i*:

$$e(\mathbf{h_i}, \mathbf{h_j}) = \text{LeakyReLU}(\mathbf{a}^\top \cdot [\mathbf{W h_i} \parallel \mathbf{W h_j}]) \tag{2}$$

where $\mathbf{a} \in \mathbb{R}^{2d'}$, $\mathbf{W} \in \mathbb{R}^{d' \times d}$ are learned and $\parallel$ denotes vector concatenation. These attention scores are normalized across all neighbors $j \in \mathcal{N}_i$ using softmax, and the attention function is defined as:

$$\alpha_{ij} = \text{softmax}_j(e(\mathbf{h_i}, \mathbf{h_j})) = \frac{\exp(e(\mathbf{h_i}, \mathbf{h_j}))}{\sum_{j' \in \mathcal{N}_i} \exp(e(\mathbf{h_i}, \mathbf{h'_j}))} \tag{3}$$

Then GAT computes a weighted average of the transformed features of the nodes (followed by a nonlinearity σ) as the new representation of *i*, using the normalized attention coefficients:

$$\mathbf{h_i}' = \sigma\big(\sum_{j \in \mathcal{N}_i} \alpha_{ij} \cdot \mathbf{W h_j}\big) \tag{4}$$

2.3 The expressive power of Graph Attention Mechanisms

Attention is a mechanism for computing a distribution over a set of input *key* vectors, given an additional *query* vector. For each *query* the computed distribution represents the significance or the ranking of the *keys* in relation to the *query*. If the attention function always weighs one key over others, unconditioned on the query, the authors say that this attention function is *static*. Static attention scoring functions are very limited because one key is always selected regardless of the query. Such functions cannot model situations where different keys have different relevance to distinct queries. In contrast, dynamic attention can select any key using any query.

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a graph with node representations $\mathbf{h_1}, ..., \mathbf{h_n}$ and **a** and **W** the parameters of the GAT layer. The learned vector **a** can be expressed as concatenation $\mathbf{a} = [\mathbf{a_1} \parallel \mathbf{a_2}] \in \mathbb{R}^{2d'}$ with $\mathbf{a_1}, \mathbf{a_2} \in \mathbb{R}^{d'}$. Equation (2) can be reformulated as:

$$e(\mathbf{h_i}, \mathbf{h_j}) = \text{LeakyReLU}(\mathbf{a_1}^\top \mathbf{W h_i} + \mathbf{a_2}^\top \mathbf{W h_j}) \tag{5}$$

There exists a node $j_{max} \in \mathcal{V}$ such that $\mathbf{a_2}^\top \mathbf{W h_{j_{max}}}$ is maximal among all nodes $j \in \mathcal{V}$. Due to monotonicity of LeakyReLU and softmax for each query node $i \in \mathcal{V}$, the node j_{max} also leads to the maximal value of its attention distribution $\{\alpha_{ij} | j \in \mathcal{V}\}$. Consequently key $\mathbf{h_{j_{max}}}$ is selected for every query $\mathbf{h_i}$. For any set of nodes $\mathcal{V}$ and a trained GAT layer, the attention function α defines a constant ranking of the nodes independent of the query nodes *i*. The only impact of $\mathbf{h_i}$ lies in the "sharpness" of the produced attention distribution. That is why attention function defined as in Equation (3) computes *static* attention. Veličković [2] discovered that utilizing H distinct attention heads and concatenating their outputs was advantageous. They found the role of the multi-head to be stabilizing the learning process. Each attention head uses different parameters **a** and **W** and determines unique attention distribution $\{\alpha_{ij} | j \in \mathcal{V}\}$ with different node $j_{max} \in \mathcal{V}$ corresponding to the maximum value within the distribution.

2.4 Dynamic Graph Attention Network: GATv2

The primary issue with the conventional GAT scoring function (Equation(2)) is that the learned layers **W** and **a** are applied sequentially, which leads to their collapse into a single linear layer. To fix this limitation, the GATv2 applies **a** after the nonlinearity (LeakyReLU), and the parameter **W** after the concatenation. This approach effectively utilizes an MLP to calculate the score for each query-key pair:

$$e(\mathbf{h_i}, \mathbf{h_j}) = \mathbf{a}^\top \text{LeakyReLU}(\mathbf{W} \cdot [\mathbf{h_i} \parallel \mathbf{h_j}]) \tag{6}$$

By altering the sequence of internal operations in GAT, the scoring function of GATv2 enables the layer to compute attention distributions $\{\alpha_{ij} | j \in \mathcal{V}\}$ with varying rankings of the neighbouring nodes and allows the queries to focus on different relevant inputs. The paper's authors demonstrate through a theorem that the GATv2 layer computes dynamic attention for any set of node representations $\{\mathbf{h_1}, ..., \mathbf{h_n}\}$. By modifying the order of operations in GAT, GATv2 achieves universal approximator attention function and is thus strictly more powerful than GAT.

2.5 Complexity Analysis

Single GATv2 head has the same time-complexity as GAT's declared complexity: $\mathcal{O}(|\mathcal{V}|dd' + |\mathcal{E}|d')$. Single GAT head has a learned vector and a matrix: $\mathbf{a} \in \mathbb{R}^{2d'}$ and $\mathbf{W} \in \mathbb{R}^{d' \times d}$, thus overall $2d' + dd'$ learned parameters. Single GATv2 head has the learned vector $\mathbf{a} \in \mathbb{R}^{d'}$ and it's learned matrix is twice large: $\mathbf{W} \in \mathbb{R}^{d' \times 2d}\mathbf{W} \in \mathbb{R}^{d' \times 2d}$ because it is applied on the concatenation $[\mathbf{h_i} \parallel \mathbf{h_j}]$. Thus, the overall number of learned parameters is $d' + 2dd'$. In the experiments, to rule out the increased number of parameters over GAT the authors constrained $\mathbf{W} = [\mathbf{W}' \parallel \mathbf{W}']$ and thus the number of parameters are $d' + dd'$

3 Reproduction setting

For the reproduction of the experiments, I used publically available code of the authors, available at `https://github.com/tech-srl/how_attentive_are_gats`.
Generally, all experiments depend on PyTorch 1.7.1 and PyTorch Geometric version 1.7.0. In this project, the code was written and run in PyCharm and can be run on either GPU or CPU.

The main aim of the experiments is to evaluate the performance of dynamic and static attention mechanisms across diverse tasks while highlighting the limitations of static attention. In order to ensure a fair comparison between the types of attention, the same architecture of the model is used for each attention layer. The selection of the specific attention layer of the model to be tested is determined by specifying it as a parameter during the experiment runs. This approach ensures that any differences in performance between the layers can be attributed to their individual parameters and not to differences in the overall architecture. The implementation of the models allows for the specification of various hyperparameters: base_layer: the type of attention layer to be used (GAT or GATv2), the number of input, hidden and output features, the number of attention layers to be used in the model and the number of attention heads in each layer, the dropout rate to be used during the training, the use of layer normalization, residual connections and graph sampling methods and their parameters. Moreover, the implementation of the models in this projects is designed to be flexible, modular, and easy to extend for future research. I utilized the implementation of GAT attention layer from torch_geometric library, which is a popular and well-documented library for graph neural networks and for the GATv2 attention layer, I employed the author's original implementation.

To train and evaluate the models, various datasets were used, including both synthetic and real-world graphs. In the following subsections, I provide a detailed analysis of the experimental settings and training details specific to each of the reproduced experiments. This includes information on the datasets used, the hyperparameters chosen, the number of epochs trained, the optimization algorithms employed, and the evaluation metrics utilized to assess the performance of the models. By providing a comprehensive analysis of these details, I aim to ensure reproducibility and transparency of the experiments and enable future researchers to build upon this work.

In all experiments learned matrix of GATv2 model is constrained by setting $\mathbf{W} = [\mathbf{W}' \parallel \mathbf{W}']'$ to rule out the increased number of parameters over GAT as the source of empirical difference. All models have residual connections as in [2].

3.1 Node Prediction

For the first reproduced experiment, the aim was to compare the performance of GATv2 and GAT on the node-prediction dataset "ogbn-arxiv" from Open Graph Benchmark (OGB) [3]. OGB is a widely recognized benchmark suite in graph machine learning, providing standardized datasets, data loaders, and evaluation metrics for various graph-related tasks. The use of OGB is important for reproducible research in graph machine learning and has been widely adopted by the research community as a standard benchmark.

The ogbn-arxiv dataset is a directed graph, representing the citation network between all Computer Science (CS) arXiv papers. Each node is an arXiv paper and each directed edge indicates that one paper cites another one. Each paper comes with a 128-dimensional feature vector obtained by averaging the embeddings of words in its title and abstract. The task is to predict the 40 subject areas of arXiv CS papers, e.g., cs.AI, cs.LG, and cs.OS, which are manually determined (i.e., labeled) by the paper's authors and arXiv moderators. I used the OGB Python package to work with this dataset, available at `https://ogb.stanford.edu/`.

To ensure a fair comparison, the same model architecture was used for all three models, with the only difference being the type of attention mechanism used. I used provided data splits of OGB, Adam optimizer and cross-entropy loss function. The authors tuned hyperparameters according to validation score and early stopping but did not specify which model was used for tuning. Experiment was carried out for all models with 1 and 8 attention heads. Following hyperparameters are used: number of layers = 3, hidden layer size = 256, batch size = 20000, learning rate = 0.001, number of epochs = 50, probability of dropout = 0.25 and GraphSAINT [4] with the Random walk sampler was used as sampling method with number of steps = 30 and the length of the random walk=30. GraphSAINT improves large-scale graph learning efficiency by partitioning graphs into smaller, normalized subgraphs using three samplers, which are then used for training GNNs.

To account for the stochastic nature of the experiments, the performance metric used for comparison of the models is the average accuracy after 5 runs of the experiment for each models. During each epoch, the model was trained on the training set and then evaluated on both the validation and test sets. Early stopping was implemented based on the validation accuracy, with a patience of 20 epochs. This means that if the validation accuracy did not improve for 20 consecutive epochs, the training was stopped early to prevent overfitting.

3.2 Robustness To Noise

The authors examined the robustness of dynamic and static attention to noise. Given an input graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and noise ratio $0 \geq p \leq 1$, they randomly sample $|\mathcal{E}| \times p$ non-existing edges $\mathcal{E}'$ from $\mathcal{V} \times \mathcal{V} \backslash \mathcal{E}$. They train the GNN on noisy graph, $\mathcal{G}' = (\mathcal{V}, \mathcal{E} \cup \mathcal{E}')$. To evaluate the robustness of the GAT and GATv2 models with 8 attention heads, I examined their accuracy on node prediction on the ogbn-arxiv dataset with induced structural noise. I used the same best-found hyperparameters and experimental settings as in the node-prediction experiment. For both models, and for noise ratios $p \in \{0.1, 0.2, 0.3, 0.5\}$, the average test accuracy over 5 runs is reported.

3.3 Link-prediction

In this experiment, the objective is to compare the performance of GATv2 and GAT in the link-prediction task using the ogbl-collab dataset.

The ogbl-collab dataset is an undirected graph representing a collaboration network between authors, with nodes as authors and edges indicating collaborations. Nodes have 128-dimensional features, obtained by averaging the word embeddings of papers that are published by the authors. All edges are associated with two meta-information: the year and the edge weight, representing the number of co-authored papers published in that year. The graph is a dynamic multi-graph with multiple edges between nodes if they collaborate in multiple years. The prediction task is to predict the future author collaboration relationships given the past collaborations. The goal is to rank true collaborations higher than false collaborations. Specifically, the task is to rank each true collaboration among a set of 100,000 randomly-sampled negative collaborations, and count the ratio of positive edges that are ranked at K-place or above (Hits@K). The data is split according to time, in order to simulate a realistic application in collaboration recommendation. Collaborations until 2017 are used as training edges, those in 2018 as validation edges, and those in 2019 as test edges.

The authors evaluated their models' performance on link prediction using the Average Hits@50 metric after 10 runs. Hyperparameters were tuned using the same approach as the node prediction experiments, and the experiment was carried out for all models with 1 and 8 attention heads. The hyperparameters used were: number of layers = 3, hidden layer size = 256, learning rate = 0.001, batch size = 65536, number of epochs = 400, and no sampling method was used. To train the model on the dataset, binary cross-entropy loss function with both positive and negative samples was used. Negative training samples were randomly sampled node pairs from the dataset, while the provided negative validation and test edges of OGB were used. The data splits of OGB were used for the experiments, and the adam optimizer was used, as in the previous experiments. In the paper both models were assessed with and without incorporating validation edges in the training prrocess. This approach aimed to evaluate the performance and robustness of each model under different degrees of available training data and edge information. Notably, the models were not designed to incorporate

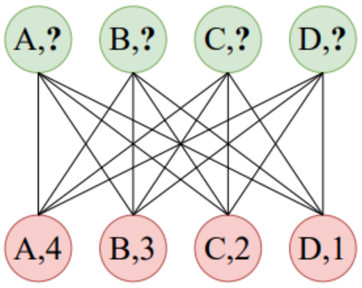

Figure 1: The DictionaryLookup problem of size $k = 4$: every node in the bottom row has an alphabetic *attribute*(A,B,C,D and a *numeric* value (1,2,3,4); every node in the upper row has only an attribute; the goal is to predict the value for each node in the upper row, using its attribute.

the weights of these edges during either training or inference. In contrast to the node prediction experiment, early stopping was not used in the link prediction experiment. Instead, the models were trained for a fixed number of epochs without monitoring the validation accuracy.

In addition to the authors' evaluation using the Average Hits@50 metric, I further assessed the models' performance by computing the Average Hits@10 and Hits@100 metrics after 10 runs to provide a more comprehensive understanding of their performance in the link prediction task

Graph neural network is utilized to learn node representations from the dataset and then trained fully connected feedforward network, with the same number of hidden layers and the same hidden size as the GNN, is used for link prediction. The feedforward network uses ReLU activation functions for the hidden layers and a sigmoid activation function for the output layer. The input to the feedforward network was the element-wise product of the node representations output by the GNN.

3.4 Synthetic Benchmark: DictionaryLookup

The DictionaryLookup problem is a synthetic benchmark that authors designed to demonstrate the limitations of static attention mechanism. In this problem, a complete bipartite graph with 2k nodes is created, consisting of "key nodes" with attributes and values, and "query nodes" with attributes only. The goal is to predict the value of each query node based on its attribute. Each graph in the dataset has a different mapping from attributes to values. Different datasets are created for each k = 1, 2, 3, ..., with separate models trained to measure per-node and per-graph accuracy. This contrived problem is relevant to any subgraph with keys that share more than one query, and each query needs to attend to the keys differently. Such subgraphs are very common in real-world domains. The problem tests the layer itself because it can be solved using a single GNN layer, without suffering from multi-layer side-effects such as over-smoothing, over-squashing, or vanishing gradients. Figure 1 shows one sample graph from the dataset.

I conducted experiments with both GATv2 and GAT models with 1 and 8 attention heads, employing k values of 10, 20, 30, 40, 50. In all experiments I used a hidden size of d = 128 and batch size of 1024, Adam optimizer, cross-entropy loss function and ReduceLROnPlateau callback function with learning rate decay of 0.5 and patience to 10 epochs. Datasets were split to train and test with ratio of 80:20. Since this is a contrived problem, there is no validation set, and the reported results can be thought as validation results. Key nodes are encoded as a sum of learned attribute embedding and a value embedding, followed by ReLU activation function. Various normalization techniques, dropout, activation functions, and learning rates were explored but did not alter the general trend. so the experiments were conducted without any normalization, without dropout and a learning rate of 0.001.

Similar to the node prediction experiment, the models were trained on the training set and evaluated on the test sets during each epoch to monitor its performance. Early stopping was applied based on the test accuracy, although the specific patience hyperparameter was not specified in the original experiment. For consistency with previous work, a patience value of 20 epochs was chosen. The authors did not specify the number of epochs and batch size to be used in the experiments. In the original code, there is another parameter which requires specification at the input called eval_every. The name of this parameter and its default values in the code suggest that it specifies the evaluation of the model to be performed in eval_every epoch. However, in the implementation the number of epochs used is equal to the specified number of epochs at the input devided by the eval_every parameter. In each epoch the authors expand the training set by replicating each sample graph in the train set eval_every times. In the experiment description it is stated that the dataset consist of k! graphs which is the number of all permutations of the keys in the graph. Expanding the size of training set from $0.8k!$ to $0.8k! \cdot$ eval_every allows for the use of bigger batches during the training which proves to be beneficial to the performances of the models. In the experiments the value of eval_every is set to 10.

4 Evaluation

In this section I compare and evaluate the performance of GAT, GATv2, and other non-attention-based models across the experiments detailed in the Experimental Settings section. The goal is to validate the original paper's claims and assess the practical impact of GATv2's more expressive attention mechanism and determine its effectiveness relative to the original GAT model and other baseline approaches. In the subsequent subsections, I will present the results for each task, compare them with the findings reported in the paper, and provide a comprehensive analysis of the outcomes.

4.1 Node Prediction

The experimental results demonstrate the performance of GAT, GATv2 and two widely-used non-attentive GNNs, namely GCN [5] and GraphSAGE [6], on the ogbn-arxiv dataset under previously described setting. For the non-attentive models, the results reported in the original paper were used instead of conducting new experiments.

Table 1: Average accuracy in ogbn-arxiv dataset after 5 runs ± std. † results were reported in the original paper.

Model	Attn. Heads	Average Accuracy
GCN†	0	71.74 ± 0.29
GraphSAGE†	0	71.49 ± 0.27
GAT	1	**71.93** ± 0.27
	8	71.68 ± 0.25
GATv2	1	71.81 ± 0.35
	8	71.63 ± 0.08

The experimental results are consistent with those reported in the paper. The paper reported that GATv2 consistently outperforms GAT across different settings and datasets, with even a single head of GATv2 outperforming 8 heads of GAT in some cases. However, our experimental results do not demonstrate a clear superiority of GATv2 over GAT in the node prediction task on the ogbn-arxiv dataset. In fact GAT with 1 head ouperforms all models. In both single-head and multi-head settings, the performance of GAT and GATv2 models is quite similar, with minimal differences between the two models.

The ogbn-arxiv dataset may not be the ideal dataset for highlighting the advantages of GATv2's more expressive attention mechanism. The dataset may not have complex enough patterns or structure for the additional expressiveness of GATv2 to make a significant impact on the node prediction task.

Furthermore, other factors, such as hyperparameter tuning, data preprocessing, and experimental setup, could influence the results. The effectiveness of GATv2 might be more apparent under specific experimental conditions that were not explored in these studies.

Based on the reported results for other datasets, such as ogbn-products, ogbn-mag, and ogbn-proteins, it appears that the advantages of GATv2's more expressive attention mechanism might not be as pronounced for node prediction task as initially suggested.

While the theoretical advantages of GATv2's more expressive attention mechanism are evident, its practical impact on the node prediction task appears to be more subtle and context-dependent. Further research on a wider range of datasets, tasks, and experimental settings is necessary to better understand the conditions under which GATv2's increased expressiveness provides significant performance benefits over the original GAT model for node prediction task.

4.2 Robustness to noise

I explored the robustness of the models to structural noise by repeating the experiments from node prediction task with added structural noise. Figure 2 shows the accuracy on ogbn-arxiv dataset as

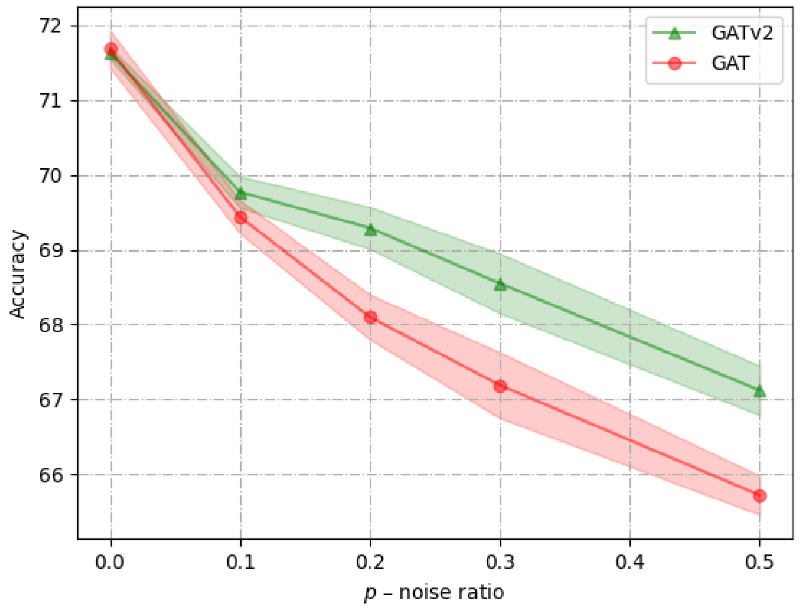

Figure 2: Test accuracy compared to the noise ratio. Each point is average of 5 runs, error bars show standard deviation.

function of nose ratio p. As p increase both models naturally exhibit a decrease in test accuracy. However due to the ability of GATv2 to compute dynamic attention, the decline in accuracy is more gradual compared to GAT, which experiences a sharper drop. The authors hypothesize that the ability to perform dynamic attention dynamic attention helps the models to differentiate between original data edges $(\mathcal{E})$ and noise edges $(\mathcal{E}')$. They argue that GAT is unable distinguish between edges, because it scores the source and target nodes separately. These results clearly demonstrate the robustness of dynamic attention over static attention in noisy settings.

4.3 Link Prediction

I evaluated GATv2, GAT, and two non-attentive GNNs on the link-prediction dataset ogbl-collab from OGB. I conducted the experiments and report the results for GAT and GATv2 myself. For the non-attention models GCN and GraphSAGE, I did not conduct experiments and instead used the results reported in their original papers for comparison.

The non-attentive GraphSAGE outperforms all attentive graph neural networks (GNNs), leading the authors to hypothesize that attention mechanisms may not be essential for datasets in the paper. Generally, many GNNs, particularly Message-Passing Neural Networks (MPNNs), perform poorly compared to simple heuristics on link prediction tasks [7]. The main reason why MPNNs tend to be poor link predictors is due to the equivalence of message passing to the Weisfeiler-Leman graph

Table 2: Average Hits@10, Hits@50 and Hits@100 after 10 runs ± std. The best result among GAT nad GATv2 is marked in **bold**; the best result among all GNNs is marked in **bold and underline**. † results were reported in the original paper.

Model	Attn. Heads	Hits@10		Hits@50		Hits@100	
		w/o val edges	w/ val edges	w/o val edges	w/ val edges	w/o val edges	w/ val edges
No attention	GCN[†]	–	–	44.75 ± 1.07	44.75 ± 1.07	–	–
	GraphSAGE[†]	–	–	**<u>48.10 ± 0.81</u>**	**<u>54.63 ± 1.12</u>**	–	–
GAT	1	18.22 ± 1.11	17.08 ± 2.88	41.44 ± 2.10	47.79 ± 2.44	49.36 ± 2.18	58.08 ± 3.55
	8	**<u>19.30 ± 2.27</u>**	17.26 ± 3.79	42.38 ± 2.81	48.52 ± 3.79	49.19 ± 2.77	57.82 ± 2.72
GATv2	1	17.08 ± 3.30	17.16 ± 4.03	40.33 ± 2.09	47.99 ± 2.92	48.40 ± 0.86	59.08 ± 1.17
	8	19.20 ± 3.19	**<u>21.69 ± 5.05</u>**	**43.37 ± 1.91**	**49.15 ± 2.89**	**<u>50.82 ± 1.18</u>**	**<u>59.30 ± 1.81</u>**

isomorphism test. [8; 9]. Recent work on GNNs focuses on subgraph-based methods, which have achieved state-of-the-art for link prediction. [7].

Moreover, the authors hypothesize that a dynamic attention mechanism is particularly beneficial for selecting the most relevant neighbors when the total number of neighbors is high, i.e., in graphs with a high node degree. The ogbl-collab dataset has a low average node degree of 8.2, suggesting that it may not be the ideal dataset to showcase the importance of a dynamic attention mechanism for this task. Nonetheless, considering the known limitations of MPNNs, I hypothesize that GATv2 does not achieve state-of-the-art performance on link prediction tasks.

GATv2 models surpass GAT models both with and without validation edges for Hits@50 and Hits@100. However, the results for Hits@10 are similar, with GAT slightly outperforming GATv2. When ranking K = 10, both models exhibit poor performance, further highlighting the limitations of MPNNs for this task.

4.4 Synthetic Benchmark: DictionaryLookup

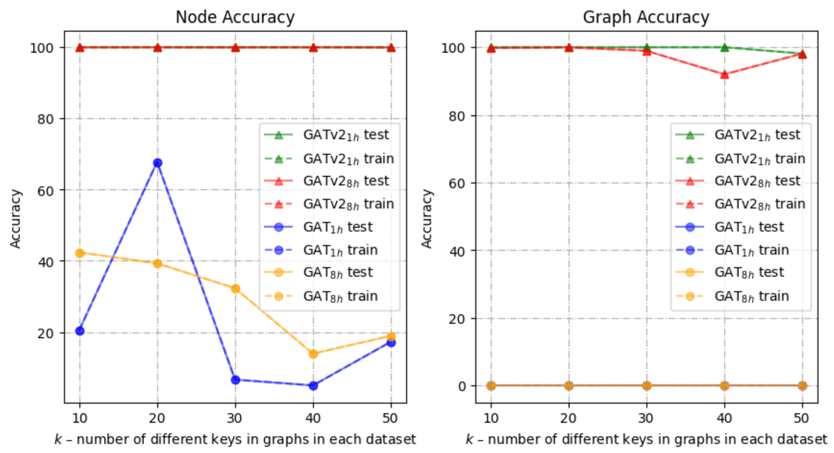

Figure 3: Node and graph Accuracy for GAT and GATv2 with 1 and 8 attention heads: GATv2 easily achieves 100% node train and test accuracies with both 1 and 8 attention heads.

Figure 3 presents the resultsthat align with the authors findings in the paper: GAT model with 1 and 8 attention heads failed to fit the training set for any value of k, regardless of the number of iterations or the training methods employed. This demonstrates its explicit failure to generalize. In contrast, GATv2 easily achieves 100% train and test accuracy on node prediction task.For graph prediction tasks, GATv2 with a single attention head attains 100% accuracy on both training and testing sets, while the version with 8 heads achieves near-perfect accuracy. The latter's performance is believed to be due to the experimental setup, with the expectation that appropriate hyperparameter tuning would allow the model to reach 100% graph accuracy on both sets. These findings highlight the limitations of the original GAT model, which are easily solved by GATv2 thanks to its ability to perform dynamic attention.

5 Discussion

In this reproduction project, I confirmed the findings of the original paper. GATv2 outperforms GAT across all examined benchmarks except for the node prediction task on ogbl-arxiv.

GATv2 is significantly more robust to noise than GAT. In the link prediction task on the ogbl-collab, GATv2 does not outeprform the non-attentive GraphSAGE model; however, message-passing GNNs generally show severe limitations on this task and cannot compare to state-of-the-art methods. Despite this, GATv2 outperforms the original GAT model in this task, emphasizing its improved performance. The synthetic DictionaryLookup problem demonstrates the limitations of the original GAT model, which struggles to fit the training data and generalize to the test set. In contrast, GATv2 easily achieves 100% accuracy for both training and testing sets, highlighting its ability to solve simple problems that GAT cannot. In this problem, I demonstrated the ability of GATv2 to compute different ranking distribution of keys nodes for each query node. Every query can select different key node which is not a case with GAT in which all queries attend to the same node.

In the benchmark node classification tasks on the ogbl-arxiv dataset, GATv2 did not show superior performance; however, it is expected to perform better on more complex datasets.

Interestingly, in some benchmarks, non-attentive models achieved higher accuracy than GNNs using attention mechanisms. This suggests that the choice of graph attention mechanism is not always straightforward, and the optimal architecture depends on the complexity of interactions between nodes in the dataset.

Theoretically weaker models may perform better in practice if a task does not require high expressiveness, while stronger models might overfit the training data. The more complex the interactions between nodes are, the more benefit a GNN can derive from theoretically stronger graph attention mechanisms like GATv2. Authors further explaine in the paper: The primary question is whether a problem has a global ranking of influential nodes (in which case GAT is sufficient) or if different nodes have different rankings of neighbors (which would necessitate GATv2).

The process of reproducing the results was not overly difficult, as the authors provided clear, well-structured, and well-written code. Following their implementation, I was able to obtain the expected results that aligned with the findings reported in the paper.

However, some challenges arose during the reproduction process. The main difficulty was understanding certain parts of the code, as there were not many comments or explanations provided. This required additional time and effort to decipher the logic behind specific code segments and to ensure that the implementation was correctly aligned with the paper's methodology. Despite these challenges, the overall reproduction process was successful, and the results I obtained further validated the findings of the original paper. In my implementation of the experiments, I provided comprehensive explanations for various sections of the code. By including detailed comments, I aimed to elucidate the underlying logic and rationale behind each segment, offering valuable insight into the inner workings of the code. This enhanced documentation will facilitate a better understanding of the implementation for future researchers, enabling them to build upon this work with ease and efficiency.

6 Conclusion

In this project, I conducted a comprehensive analysis of the paper "How Attentive are Graph Attention Networks?" and implemented the GATv2 model to compare its performance with the original GAT model. I provided a detailed account of the model implementations and experiment reproductions, further clarifying and refining the original code supplied by the authors. Additionally, I elucidated the inner workings of the model, the concepts of *static* and *dynamic* attention mechanisms, and the theoretical findings presented in the paper. Finally, I reproduced experiments on Node Prediction, Robustness to Noise, Link Prediction, and DictionaryLookup problem, assessing the effectiveness of GATv2's *dynamic* attention mechanism across various tasks and settings.

This project validates the findings of the original paper and demonstrates that GATv2's dynamic attention mechanism yields significant enhancements over the original GAT model in a range of tasks. In complex tasks, domains, and challenging datasets, the GAT model should be replaced with GATv2 to achieve improved performance.

References

[1] Shaked Brody, Uri Alon, Eran Yahav. How attentive are Graph Attention Networks? In *ICLR*, 2022.

[2] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. *Advances in Neural Information Processing Systems*,33,2020.

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

[4] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. Graph-saint: Graph sampling based inductive learning method. *arXiv preprint arXiv:1907.04931*, 2019.

[5] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

[6] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in neural information processing systems*, pages 1024–1034, 2017.

[7] Benjamin P. Chamberlain, Sergey Shirobokov, Emanuele Rossi, Fabrizio Frasca, Thomas Markovich, Nils Hammerla, Michael M. Bronstein, Max Hasmire. Graph Neural Networks for Link Prediction with Subgraph Sketching. *arXiv preprint arXiv:2209.15486*, 2022.

[8] Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *AAAI Conference on Artificial Intelligence*, pages 4602–4609. AAAI Press, 2019.

[9] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *ICLR*, 2019.