Exploring Graph Attention Networks

Kaloyan Aleksiev, Samuel Barrett, Damon Falck, Filip Mihov*

University of Oxford

Abstract

This report is an excursion into the Graph Attention Networks (GATs) of [VCC+17] as well as several theoretical and empirical extensions to their work. We first examine the ideas from the original paper and reproduce their experimental results; we then discuss and implement various other powerful attention-based methods suggested by [VSP+17] and [BAY21]. We next conduct a theoretical investigation into the expressive power of GATs and propose a novel modification to the original attention mechanism, which we prove exhibits the powerful kind of attention mentioned in [BAY21]. Finally, we perform two empirical studies comparing various versions of the mechanisms we've discussed, all implemented from scratch, on a range of well-known graph datasets.

1 Introduction

Graph-structured data is found abundantly in the real world. From transportation and drug-discovery to social media and human interactions, graph structures prove to be a powerful tool for representing dependencies between entities. It is natural, then, to wish to apply the techniques of deep learning, which have enjoyed great success in a wide range of applications, to graph-structured data. To this end, Graph Neural Networks (GNNs) were proposed by [SGT+08] as a computational mechanism which allows deep learning on graph structures. Since then GNNs have flourished into an active area of research, with many architectures achieving state of the art performance on common datasets.

One drawback of most graph network models, however, is an inability to easily learn different importance weights between different pairs of graph vertices. Recognising this problem, [VCC+17] propose Graph Attention Networks (GATs), a class of models which learn attention weights between neighbouring vertices from the feature vectors of those vertices. As with other GNNs, they construct successive representations of each vertex based on neighbourhood aggregation — the novelty in their mechanism lies in the way they construct the weights for this representation: by stacking layers in which nodes are able to attend over their neighborhoods' features, they implicitly enable specifying different weights to different nodes in a neighborhood. Their proposed architecture achieves state-of-the-art performance on several well-known tasks.

There are still a number of drawbacks to the GAT architecture, though, and in this work we take GAT as a starting point for an exploration of various other attention-based graph learning techniques. **Our main contributions are as follows:**

- A full reimplementation of the GAT models introduced in [VCC+17] and reproduction analysis of their experimental results.
- The implementation and benchmarking of a number of other existing attention mechanisms, specifically focusing on transformer-inspired architectures [VSP+17, NNP19].
- The proposal (and implementation) of a new modification to the original GAT layer which is proven to exhibit the powerful dynamic attention from the theoretical framework of [BAY21],

*All joint-first authorship.

as well as surpassing GAT on a new artificial dataset designed to illustrate the differences in expressive power of different attention mechanisms.

- A comparison of various versions of the GAT and GATv2 [BAY21] attention mechanisms on a wider array of datasets.
- A strengthening of the notions of "static" and "dynamic" attention from [BAY21] into "super-static" and "super-dynamic" attention. These stronger definitions still apply to GAT and GATv2 respectively, and we argue should replace the old definitions.

We start in Section 2 by motivating and summarising the original ideas from [VCC+17], as well as comparing the performance of our reimplementations of their models to the empirical results from the original paper. Section 3 moves on to a discussion of a number of other related attention-based techniques we test too, providing background on each of the papers inspiring our extensions. In Section 4 we present our theoretical findings on expressiveness of attention and describe our novel dynamic-attention mechanism, as well as empirically demonstrating its superiority on a new artificial dataset. Last, in Section 5 we compare our implementations of the mechanisms from Sections 3 and 4 against our GAT implementation on two key real-world datasets, as well as studying the effect of various minor variations to GAT and GATv2 [BAY21] across a number of more varying tasks. All of our implementations are from scratch using the PyTorch deep learning framework.

Rationale for choice. We chose the GAT paper for several reasons. Firstly, attention mechanisms are powerful computational tools that have proven extremely effective in a variety of situations; discovering attention mechanisms with desirable computational and mathematical properties could lead to profound improvements on a number of tasks. Secondly, understanding attention mechanisms more deeply from a theoretical perspective and forming a general framework for their expressive power is still an open problem; drawing on several fields of mathematics and presenting a chance to establish a firm foundation for building provably expressive models, we believed we have mathematical backgrounds well-suited for contributing in this area. Finally, we were drawn to the implementation challenge that attention mechanisms present: finding the most efficient ways to execute the computations in parallel while producing readable and extensible code was something we thoroughly enjoyed.

Notation. Throughout this report unless otherwise specified, working over a graph $G = (V, E)$, we denote the old and new feature vectors of a node $v \in V$ as $\mathbf{h}_v \in \mathbb{R}^F, \mathbf{h}'_v \in \mathbb{R}^{F'}$ respectively, and we write $N(v)$ for the set of its immediate neighbours (including itself); we use $[n]$ to refer to the set $\{1, \ldots, n\}$.

2 Graph Attention Networks

2.1 Background

Most existing graph neural network models can be described within a common framework called Message-Passing Neural Networks (MPNNs) [GSR+20] where each layer aggregates the neighbourhoods of each node to compute the subsequent representation of that node: the feature vector update has the form

$$\mathbf{h}'_v = \texttt{combine}(\mathbf{h}_v, \texttt{aggregate}(\{\mathbf{h}_u : u \in N(v)\})) \tag{1}$$

where `combine` and `aggregate` are (possibly layer-dependent) functions specified by the model. Most common convolutional architectures do not provide a mechanism for learning the dependencies between different nodes in the graph; the `aggregate` function, which normally consists of a linear map followed by some form of sum, weights contribution from the various nodes according to the corresponding edge-weights in the underlying graph (if any weighting is applied at all).

In contrast, the GAT approach *learns* the weights for each edge, i.e. implicitly learns the importance and the co-dependence between pairs of nodes. Formally, given a set of node vectors $\{\mathbf{h}_1, \ldots, \mathbf{h}_n\} \subseteq \mathbb{R}^F$

for some $F \in \mathbb{N}$, the GAT attention layer utilises a learnable weight matrix $\mathbf{W} \in \mathbb{R}^{F' \times F}$ and attention-vectors $\mathbf{a}_1, \mathbf{a}_2 \in \mathbb{R}^{F'}$ to compute *attention weights*

$$\alpha_{ij} \coloneqq \frac{\exp(\beta_{ij})}{\sum_{k \in N_i} \exp(\beta_{ik})}, \qquad i, j \in [n] \tag{2}$$

between each pair of nodes, where

$$\underbrace{\boxed{\beta_{ij} \coloneqq \text{LeakyReLU}(\mathbf{a}_1^\top \mathbf{W}\mathbf{h}_i + \mathbf{a}_2^\top \mathbf{W}\mathbf{h}_j).}}_{\textit{GAT attention rule}^1} \tag{3}$$

Here the LeakyReLU has a fixed slope parameter $\gamma \in \mathbb{R}$, which [VCC+17] set to 0.2 (and we follow suit in our experiments).

Subsequently, the updated feature vector for each node is computed as

$$\mathbf{h}_i' \coloneqq \sigma\left(\sum_{j \in N_i} \alpha_{ij} \mathbf{W}\mathbf{h}_j\right) \tag{4}$$

for some non-linear activation function σ. [VCC+17] found it helpful to leverage several of these attention "heads" to obtain a multi-head attention mechanism, which usually takes the form

$$\mathbf{h}_i' = \Big\|_{k=1}^{K} \mathbf{h}_{ik}', \tag{5}$$

i.e. K of these heads are stacked together. The aggregation of the heads need not be concatenation; in particular, concatenation is not sensible in the final layer of a network, so instead an averaging procedure is used, *before* applying the activation function τ:

$$\mathbf{h}_i' = \tau\left(\frac{1}{K}\sum_{k=1}^{K} \mathbf{h}_{ik}'\right), \qquad \sigma = \text{identity}. \tag{6}$$

This attention mechanism has several advantages which make it a compelling choice as a layer in a deep neural network:

- **Efficiency.** Computationally, it is highly efficient as the operation of the self-attention layer can be parallelised across all heads, within a head across all edges, and the computation of output features can be parallelised across all nodes. The time complexity of a single GAT layer attention head is thus $O(|V|FF' + |E|F)$.
- **Expressivity.** GAT allows for implicitly assigning different importance between nodes in the same neighbourhood and could also allow different importance in the different directions of an edge between two nodes. This has the advantage of increased model capacity and potentially increased interpretability of the model if the computed attention coefficients are observed and analysed.
- **Generalisation.** Almost no structure is assumed on the graph. This has several desirable properties. Firstly, the model can be applied to unseen graphs and hence can be used in inductive settings. Secondly, the model is capable of handling both directed as well as undirected graphs, and finally, we are able to impose arbitrarily modify the graph structure before applying the model, for example to introduce domain-specific knowledge, or to ensure aggregation within a wider neighbourhood.
- **Adaptability.** The proposed attention mechanism is computationally flexible, i.e. the underlying framework allows for various tweaks of the original model and attention mechanism.

[1] We have slightly rewritten this expression to facilitate comparison with other attention mechanisms later in the report.

2.2 Reimplementation and reproduction

In this section we reimplement from scratch the GAT layers and models from the original paper [VCC+17] and reproduce their experimental results.

2.2.1 Implementation details

We found it convenient to use the PyTorch deep learning framework for our implementation, along with PyTorch Geometric for low-level graph data handling; we did not make use of any pre-implemented models or model components from either package. Specifically, we built two separate versions of our reimplementation:

1. A version using methods such as batch matrix multiplication and `torch.einsum` (optimised Einstein tensor summation) to compute attention in a pseudo-parallel fashion on CPU. In this version training logic was custom-written in Jupyter notebooks for each experiment.
2. A version using PyTorch Lightning and sparse tensor operations to compute attention in an accelerated memory-efficient manner on GPU. In this version training logic was automated as part of the data structure for each model.

Our full code is available at `https://github.com/samuelbarrett1234/atml-group-11`. The two versions are equivalent in logic, differing only in efficiency and/or interface. Indeed, we found that the performance of the two versions was identical in all cases up to randomisation error, as should be expected. The experimental results in Sections 4.5 and 5.1 are based on the first version and those of this section and Section 5.2 are based on the second. For this section we implemented the GAT layer and the subsequent model exactly as they are presented in the paper, in particular paying attention to closely replicating the model architectures, hyperparameter values and training procedures (including early-stopping strategies) they used.

2.2.2 Reproduction of experimental results

Our results on the Cora, CiteSeer, PubMed [SNB+08] and PPI [ZL17] datasets evaluated in the original paper are shown in Table 1. We did not have the computational resources to re-run each experiment 100 times as in the original paper and calculate accurate variances for our accuracies, but it should be noted that the accuracies reported on the transductive datasets are all within one standard deviation of the means reported in the original paper except for the CiteSeer result, which is about 1.6 standard deviations out. The reason for this could be purely random error or could be due to subtle differences in the early-stopping strategies used (see the next section). For the PPI dataset, our score fell slightly short of the reported score from the original paper: in this case we had to stop our training early (after approximately 1000 epochs) due to computational constraints despite the validation accuracy continuing to improve, and we believe this explains the difference in result.

Table 1: Experimental performance of our GAT reimplementation on the original datasets. For comparison, the test scores reported in [VCC+17] were $83.0 \pm 0.7\%$, $72.5 \pm 0.7\%$, $79.0 \pm 0.3\%$ and 0.973 ± 0.002 respectively.

Dataset	Task type	Validation score	Test score
Cora	Transductive node classification	80.2% accuracy	83.4% accuracy
CiteSeer	Transductive node classification	74.4% accuracy	71.4% accuracy
PubMed	Transductive node classification	79.2% accuracy	79.6% accuracy
PPI	Inductive node multi-classification	0.908 micro-F1 score	0.942 micro-F1 score

2.2.3 Discussion and criticism

Our experiments replicate the results from results from [VCC+17] to within an acceptable margin of error, indicating the overall replicability of the paper's empirical findings. It should be noted,

however, that the original authors' code-base and issues raised on their GitHub repository[2] show that they applied different early-stopping strategies to each dataset (and we followed suit); we found that applying one uniform strategy across all datasets always resulted in poorer performance than reported on at least some of them. This raises questions about the ease of overfitting on these small-scale datasets, an issue discussed more in [SMBG18].

3 Alternative attention mechanisms

The idea of learning attention coefficients is not unique to [VCC+17], and we start our extension of the original paper by discussing various alternatives to the GAT attention mechanism that have been proposed in the literature.

3.1 GATv2

[BAY21] observe that the original GAT attention mechanism suffers from some inherent limitations. They introduce the concepts of *dynamic* and *static* attention, and prove that GAT's attention is only static. Intuitively speaking, static attention forces all nodes to attend mostly to some fixed unique set of nodes, while dynamic attention allows different nodes to vary their attention according to what's most suitable for them. We give the formal definitions and a more in-depth theoretical discussion in Section 4.

[BAY21] propose a simple modification: to switch the order of operations, applying the LeakyReLU non-linearity *before* the final linear transformation instead of after. This essentially turns the attention mechanism into a multi-layer perceptron (MLP) which thus exhibits desirable function-approximation properties that the original mechanism did not. Formally, the original attention calculation eq. (3) is replaced with

$$\boxed{\beta_{ij} \coloneqq \mathbf{a}^\top \operatorname{LeakyReLU}(\mathbf{W}_1\mathbf{h}_i + \mathbf{W}_2\mathbf{h}_j),} \tag{7}$$

GATv2 attention rule

where here $\mathbf{a} \in \mathbb{R}^{F'}$ and $\mathbf{W}_1, \mathbf{W}_2 \in \mathbb{R}^{F' \times F}$ are learnable parameters. The authors of [BAY21] prove that this new attention mechanism is dynamic, and they call it GATv2.

> *Remark.* Note that the new parametrisation in eq. (7) compared to eq. (3) consists of $F' + 2FF'$ learnable scalar parameters, compared to $2F' + FF'$ in the original mechanism. This means GATv2 is fundamentally more heavily parametrised than GAT, an observation which makes meaningful comparison of the two difficult. For this reason, [BAY21] employ *weight-sharing* in their experiments by requiring $\mathbf{W}_1 = \mathbf{W}_2$, in order to eliminate increased model complexity as a source of performance differences between the two.

There are two points of ambiguity in [BAY21], which we explore more in our experiments in Section 5.2:

- **Bias vector.** The applied a vector bias term inside the LeakyReLU of eq. (7) in at least some of their experiments, something they allude to in a footnote but do not mention again. As we discuss in Section 4, the addition if a bias term is actually a non-trivial alteration.
- **Weight choice for feature updates.** The need to learn *two* weight matrices $\mathbf{W}_1, \mathbf{W}_2$ leaves ambiguity over which should be used in the subsequent feature update, eq. (4). There seem to be three sensible choices: use $\mathbf{W}_1$ (the 'target' weight matrix), $\mathbf{W}_2$ (the 'source' weight matrix), or learn another weight matrix altogether specifically for use in the feature updates. The authors do not address this question in their work, but it appears from their code that they opt for the second option, i.e. using $\mathbf{W}_2$.

[2] `https://github.com/PetarV-/GAT/issues/12`, `https://github.com/PetarV-/GAT/issues/14`

3.2 Transformers

The *transformer* architecture introduced in [VSP+17] quickly produced shockwaves throughout the field of natural language processing (NLP), and it was soon ported over to other domains such as music [HVU+18] and computer vision [ADH+21]. While GAT was inspired by the transformer, it differs in its definition of the attention scores as well as its use of attention masking (early versions of transformers instead simply prevent positions in the sequence from attending 'forward').[3] There have been several attempts at porting the transformer more directly to the graph setting [BAY21, KO20, DB20, RBX+20], often focussing on the challenge of positional embeddings for nodes (shown to be important to transformers [WTWS19], but with no obvious analogy in the graph world), e.g. GraphBERT [ZZXS20] or extracting positional embeddings from the eigenvectors of the graph's Laplacian matrix [DB20]. There is also a line of work adapting transformers to the notion of *heterogeneous graphs* [YJK+19, HDWS20, DB20], although this is orthogonal to the setting considered in this report. [RBX+20] developed a self-supervised pretraining approach (independently of [ZZXS20]) to modelling molecular graphs with transformers.

To apply transformer networks to the graph modelling setting, we first describe the multi-headed self-attention sublayer. Here we take the attention coefficients to be

$$\boxed{\beta_{ij} \coloneqq (\mathbf{W}_k \mathbf{h}_j)^\top (\mathbf{W}_q \mathbf{h}_i),} \tag{8}$$

Transformer attention rule

where $\mathbf{W}_k, \mathbf{W}_q \in \mathbb{R}^{F' \times F}$ are called the *key* and *query* matrices respectively, and are learnt by the model. Then, the attention coefficients are set to $-\infty$ if there is no edge connecting i and j (except when $i = j$), and finally the attention weights α_{ij} are computed by a softmax over j.

This differs from the architecture from [VSP+17] in two ways. Firstly, we don't mask 'autoregressively' as is typically done in NLP. Instead masking is done according to node neighbourhoods, as is done in GAT.

Once the attention weights α_{ij} have been calculated, the updated value for node i is

$$\sum_j \alpha_{ij} \mathbf{W}_v \mathbf{h}_j \tag{9}$$

for a (learnt) *value* matrix $\mathbf{W}_v \in \mathbb{R}^{F' \times F}$.

One of the novel ideas of the transformer is to use multiple attention heads, as explained in Section 2. There is a different learnt key, query and value matrix for each head. The values produced for each head are concatenated, and are then projected using a final matrix $\mathbf{W}_o$.[4]

The main steps in computing multi-headed self-attention can be elegantly summarised in the following snippet of code, modified from our repository (with neighbourhood masking removed):

```
keys = torch.einsum('ij,mjn->min', node_matrix, Wk)
queries = torch.einsum('ij,mjn->min', node_matrix, Wq)
values = torch.einsum('ij,mjn->min', node_matrix, Wv)
atts = torch.einsum('ikn,imn->ikm', queries, keys)
atts = torch.nn.functional.softmax(atts, dim=-1)
return torch.einsum('ikl,ilm,imn->kn', atts, values, Wo)
```

After applying the multi-headed self-attention sublayer, a transformer layer consists of a position-wise feedforward neural network. In the same manner as in [VSP+17], we use a 2-layer multilayer perceptron with ReLU activation in the middle.

[3] An architecture half-way between GAT and the transformer is the *set transformer* introduced in [LLK+19] which does no attention masking but is permutation-invariant.

[4] Note that, in the terminology of GAT, we are using *concatenation* rather than *averaging* as our aggregation function. This is consistent with the way transformers are typically implemented. The node embedding dimension stays the same between transformer layers because the dimensions are *split evenly between the heads*. Thus, after concatenation of the heads at the end, we are back to the input dimension.

Finally, a transformer *model* consists of a stack of these transformer layers, alternating between attention and the feedforward network.

One important component of the transformer not yet discussed are the positional embeddings - that is, how the model is able to distinguish nodes in different positions which are otherwise equivalent. In [VSP+17] this is done by a kind of sinusoidal positional embedding, calculated from the word's position in the modelled sentence. However graph models have to be permutation-equivariant. We follow the approach of [DB20] which uses the eigendecomposition of the graph's Laplacian matrix to obtain positional embeddings. Specifically, we take the first k eigenvectors with strictly positive eigenvalue[5] ordered by ascending eigenvalue. Then, this forms an $n \times k$ matrix, giving a k-dimensional embedding for each node. This is projected and then added to the node embeddings before the first layer only, as opposed to in NLP where it is added at every layer.

3.3 Universal transformers

Universal transformers are a slight yet elegant extension of the transformer architecture, introduced in [DGV+18], which have found recent applications to graph modelling in [NNP19]. Essentially the only difference is that the transformer layers share the same weights. That is, we take a single transformer layer, and 'apply it' to the input graph a fixed number of times. As explained in [DGV+18], this creates an inductive bias towards learning iterative functions, which is highly relevant to the kind of graph modelling we are doing here. We investigated whether this weight-sharing was beneficial, or not worth the loss in model capacity. To distinguish the universal transformer from the transformer described in section 3.2, we will refer to the latter henceforth as the "vanilla transformer".

4 Theoretical results

We move now to investigating in greater depth the dynamic and static attention types mentioned in Section 3. After giving the formal definitions of static and dynamic attention, we suggest an alteration which is mathematically stronger. We then propose a simple modification to the original GAT mechanism and prove in that it exhibits dynamic attention, as well as showing that a slightly weaker version would not. We introduce an artificial classification task which illustrates the advantages of dynamic over static attention, and we show that our new mechanism outperforms others empirically on this task. Finally, we prove and discuss a possible limitation of our new mechanism.

4.1 Background on static and dynamic attention

As touched upon in Section 3, [BAY21] introduce the theoretical notions of *static* and *dynamic* attention as a basis for the theoretical study of expressivity of attention mechanisms. Formally, the definitions they introduce are as follows:

Definition 1 (Static attention)**.** A (possibly infinite) family of scoring functions $\mathcal{F} \subseteq (\mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R})$ *computes static scoring* for a given set of key vectors $K = \{\mathbf{k}_1, \ldots, \mathbf{k}_n\} \subseteq \mathbb{R}^d$ and query vectors $Q = \{\mathbf{q}_1, \ldots, \mathbf{q}_m\} \subseteq \mathbb{R}^d$, if for every $f \in \mathcal{F}$ there exists a "highest scoring" key $j_f \in [n]$ such that for every query $i \in [m]$ and key $j \in [n]$, $j \neq j_f$ it holds that

$$f(\mathbf{q}_i, \mathbf{k}_{j_f}) \geqslant f(\mathbf{q}_i, \mathbf{k}_j). \tag{10}$$

We say that a family of attention functions *computes static attention* given K and Q, if its scoring function computes static scoring, possibly followed by monotonic normalization such as softmax.

Definition 2 (Dynamic attention)**.** A (possibly infinite) family of scoring functions $\mathcal{F} \subseteq (\mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R})$ *computes dynamic scoring* for a given set of key vectors $K = \{\mathbf{k}_1, \ldots, \mathbf{k}_n\} \subseteq \mathbb{R}^d$ and query vectors

[5] Recall that graph Laplacians are positive semidefinite symmetric, so all eigenvalues are real and non-negative.

$Q = \{\mathbf{q}_1, \ldots, \mathbf{q}_m\} \subseteq \mathbb{R}^d$ if for any mapping $\phi : [m] \to [n]$ there exists $f \in F$ such that for any query $i \in [m]$ and any key $j \in [n], j \neq \phi(i)$ it holds that

$$f(\mathbf{q}_i, \mathbf{k}_{\phi(i)}) > f(\mathbf{q}_i, \mathbf{k}_j). \tag{11}$$

We say that a family of attention functions *computes dynamic attention* for K and Q, if its scoring function computes dynamic scoring, possibly followed by monotonic normalisation such as softmax.

In practice, this means that static attention mechanisms have an inherent limitation: there is some node which maximises the attention weights for all the others, meaning all the other ones attend "mostly" to that node.

4.2 Are these the 'right' definitions?

The definitions of static and dynamic attention from Section 4.1 pertain to the model's ability to learn to pick a *highest-scoring* attendee key, for any given query. However, more general than this would be to look at the model's ability to learn attention according to an arbitrary *permutation* of keys:

Definition 3 (Super-static attention)**.** A (possibly infinite) family of scoring functions $\mathcal{F} \subseteq (\mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R})$ *computes super-static scoring* for a given set of key vectors $K = \{\mathbf{k}_1, \ldots, \mathbf{k}_n\} \subseteq \mathbb{R}^d$ and query vectors $Q = \{\mathbf{q}_1, \ldots, \mathbf{q}_m\} \subseteq \mathbb{R}^d$, if for every $f \in \mathcal{F}$ there exists a total order $\preceq_f$ on $[n]$ such that for every query $i \in [m]$ and keys $j_1, j_2 \in [n]$,

$$f(\mathbf{q}_i, \mathbf{k}_{j_1}) \leq f(\mathbf{q}_i, \mathbf{k}_{j_2}) \iff j_1 \preceq j_2. \tag{12}$$

Definition 4 (Super-dynamic attention)**.** A (possibly infinite) family of scoring functions $\mathcal{F} \subseteq (\mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R})$ *computes super-dynamic scoring* for a given set of key vectors $K = \{\mathbf{k}_1, \ldots, \mathbf{k}_n\} \subseteq \mathbb{R}^d$ and query vectors $Q = \{\mathbf{q}_1, \ldots, \mathbf{q}_m\} \subseteq \mathbb{R}^d$, if for every possible assignment of partial orders over $[n]$ to queries $i \in [m]$, denoted by the mapping $i \mapsto \preceq_i$, there exists $f \in \mathcal{F}$ such that for all queries $i \in [m]$, keys $j_1, j_2 \in [n]$,

$$f(\mathbf{q}_i, \mathbf{k}_{j_1}) \leq f(\mathbf{q}_i, \mathbf{k}_{j_2}) \iff j_1 \preceq_i j_2. \tag{13}$$

Indeed, [BAY21] allude to this, arguing (after after showing that GAT exhibits static attention) that GAT learns a fixed permutation of keys for any query — in our terms, that GAT is super-static. This is more restrictive, and therefore a more powerful result, than proving only that GAT learns a fixed highest-scoring key for any query.

We claim moreover that GATv2 is capable of super-dynamic attention, i.e. GATv2 is actually *more expressive* than the GATv2 paper claims:

Theorem 1 (Super-dynamicity of GATv2)**.** *Let $\mathcal{F}$ be the family of all functions expressible as a GATv2 attention layer. Then $\mathcal{F}$ satisfies Definition 4.*

Proof. The proof of this is a slight extension of the proof that GATv2 exhibits dynamic attention, in Appendix A of [BAY21]. The only difference is the construction of the function g, defined at finitely-many points before being continuously-extended.

Firstly, observe that any total order on $[n]$ can be embedded in an order-preserving fashion to $\mathbb{R}$, for example by assigning each integer its 'index' in the ordering. The fact that distinct integers can have the same index in the order is not a problem. Let $\sigma(\preceq, i)$ denote the index of $i \in [n]$ in the total order $\preceq$, with the minimal element(s) starting at index 0.

Then, defining the following for each $i \in [m], j \in [n]$,

$$g(\mathbf{q}_i \parallel \mathbf{k}_j) \coloneqq \sigma(\preceq_i, j) \tag{14}$$

we continuously extend this to the whole domain $\mathbb{R}^{2d}$. This is possible as we have only prescribed g at finitely many points. Observe that, crucially, g has the property that

$$g(\mathbf{q}_i \,\|\, \mathbf{k}_{j_1}) \leq g(\mathbf{q}_i \,\|\, \mathbf{k}_{j_2}) \iff j_1 \preceq_i j_2 \tag{15}$$

for any query $i \in [m]$ and any pair of keys $j_1, j_2 \in [n]$.

Finally, in the same manner as the proof of dynamicity in [BAY21], we appeal to the universal approximator theorem [HSW89, Cyb89, Fun89, Hor91] applied to a single GATv2 layer to conclude that there exists an $f \in \mathcal{F}$ which approximates g with arbitrary precision on a compact set. □

Corollary 1.1 (Dynamicity of GATv2)**.** *$\mathcal{F}$ satisfies Definition 2.*

The fact that these stronger versions of static and dynamic attention are still satisfied by GAT and GATv2 respectively is strong evidence to suggest that we should 'update' our definitions of static and dynamic attention to that of super-static and super-dynamic attention.

4.3 GAT with Dynamic Biases

An even simpler way we propose to correct the static behaviour of GAT is to add a bias term before applying the LeakyReLU — a modification we call GAT with Dynamic Bias (GATDB). Formally, GATDB replaces the attention calculation from eq. (3) with

$$\underbrace{\boxed{\beta_{ij} \coloneqq \text{LeakyReLU}(\mathbf{a}_1^\top \mathbf{W}\mathbf{h}_i + \mathbf{a}_2^\top \mathbf{W}\mathbf{h}_j + \xi_{ij}).}}_{\textit{GATDB attention rule}} \tag{16}$$

for some scalar biases $\xi_{ij} \in \mathbb{R}, i, j \in [n]$.

Theorem 2 (Dynamicity of GATDB)**.** *Let $\mathcal{F}$ be the family of all functions expressible as a GATDB attention layer. Then $\mathcal{F}$ satisfies Definition 2.*

Proof. Let $\phi : [n] \to [n]$ be any mapping, and fix $\mathbf{a}_1$, $\mathbf{a}_2$ and $\mathbf{W}$; we will show that the statement holds even in this case, i.e. we can still choose the ξ_{ij} accordingly.

Take $M \in \mathbb{R}$ such that $M > 2\max\{1, \gamma\} \cdot \max\{|\mathbf{a}_r^\top \mathbf{W}\mathbf{h}_s| : r \in \{1, 2\}, s \in [n]\}$ where γ is the slope parameter of the LeakyReLU; in particular, writing $c_{ij} \coloneqq \text{LeakyReLU}(\mathbf{a}_1^\top \mathbf{W}\mathbf{h}_i + \mathbf{a}_2^\top \mathbf{W}\mathbf{h}_j)$, this means that $M > c_{ij}\ \forall i, j$. Then let us construct the biases as follows:

$$\xi_{ij} \coloneqq \begin{cases} 2M + 1 & \text{if } j = \phi(i), \\ M & \text{otherwise.} \end{cases} \tag{17}$$

We claim the resulting function $f \in \mathcal{F}$ satisfies the dynamic condition. Because softmax is strictly increasing, it is enough to show that the dynamic condition is satisfied for the β_{ij}.

Indeed, fix i; by the choice of M and the fact that LeakyReLU is the identity over the positive real line, we have from eq. (16) that

$$\beta_{i\phi(i)} = \mathbf{a}_1^\top \mathbf{W}\mathbf{h}_i + \mathbf{a}_2^\top \mathbf{W}\mathbf{h}_{\phi(i)} + 2M + 1. \tag{18}$$

But then, by construction,

$$M > 2\max\{1, \gamma\} \cdot \max\{|\mathbf{a}_r^\top \mathbf{W}\mathbf{h}_s| : r \in \{1, 2\}, s \in [n]\} \geqslant \mathbf{a}_2^\top \mathbf{W}\mathbf{h}_j - \mathbf{a}_2^\top \mathbf{W}\mathbf{h}_{\phi(i)} \quad \forall j \tag{19}$$

and combining this with eq. (18) gives for all $j \neq \phi(i)$ that

$$\begin{aligned} \beta_{i\phi(i)} &> \mathbf{a}_1^\top \mathbf{W}\mathbf{h}_i + \mathbf{a}_2^\top \mathbf{W}\mathbf{h}_j + M + 1 && (20) \\ &> \mathbf{a}_1^\top \mathbf{W}\mathbf{h}_i + \mathbf{a}_2^\top \mathbf{W}\mathbf{h}_j + M = \beta_{ij}, && (21) \end{aligned}$$

where the last equality follows once again from the fact that LeakyReLU is the identity over the positive reals and because $\mathbf{a}_1^\top \mathbf{W}\mathbf{h}_i + \mathbf{a}_2^\top \mathbf{W}\mathbf{h}_j + M > 0$ by construction. □

In fact GATDB exhibits super-dynamic attention, as can be seen by choosing the weights ξ_{ij} to comply with the desired partial ordering $\preceq$. We omit the details to comply with the word-limit requirements.

4.4 Node biases

Our initial idea behind adding the bias terms was that in cases where we have some prior knowledge about the graph and the importance of some individual nodes, we would like to be able to manually increase the attention that these nodes receive. For example, in the Cora dataset we might know some papers that have been very influential in their field, and thus their neighbours would be expected to attend to them more. This could be done by introducing individual bias terms ξ_i for all nodes $i \in V$. Then, the attention coefficient between nodes i and j could be written as

$$\beta_{ij} \coloneqq \text{LeakyReLU}(\mathbf{a}_1^\top \mathbf{W}\mathbf{h}_i + \mathbf{a}_2^\top \mathbf{W}\mathbf{h}_j + \xi_j). \tag{22}$$

Thus, more influential nodes would be expected to have larger bias terms.

Let's call this mechanism GAT with Node Biases (GATNB). This approach may seem reasonable, but here we prove that adding node biases in fact keeps the attention static.

Theorem 3 (Staticity of GATNB)**.** *Let $\mathcal{F}$ be the family of all functions expressible as a GATNB attention layer. Then $\mathcal{F}$ satisfies Definition 1.*

Proof. Let $G = (V, E)$ be a graph modeled by a GATNB layer according to eq. (22), and fix the values of $\mathbf{a}_1$, $\mathbf{a}_2$, $\mathbf{W}$ and $(\xi_i)_{i\in[n]}$. Since V is finite, there exists a node j_{max} for which the value of $\mathbf{a}_2^\top \mathbf{W}\mathbf{h}_j + \xi_j$ is maximal amongst all nodes in V. Because both LeakyReLU and softmax are monotonic, it follows that for all nodes i, the attention coefficient will be largest for the node j_{max}, i.e. $\beta_{ij_{max}} \geqslant \beta_{ij}$ for all $i, j \in V$.

Thus j_{max} is the "highest scoring" key j_f required by Definition 1, and so GATNB computes static attention. □

4.5 Performance on an artificial dataset

To demonstrate the practical benefits of the edge biases in GATDB, we consider a 'toy' node classification problem which illustrates the differing expressive power between static and dynamic attention.

4.5.1 Task design

Consider a dataset consisting of a number of fully-connected graphs, each with $2k$ nodes for some $k \in \mathbb{N}$. Each node has one of k possible labels and each label is present in exactly 2 nodes; in particular, the nodes are divided into two groups — those with indives 0 to $k-1$ and those with indices k to $2k-1$ — and each node in the first group has the same label as the corresponding node in the second group with the same index mod k.

Every node has a $2k$-dimensional feature vector: for nodes in the first group, the first half of each feature vector one-hot encodes its index (which will be less than k) and the second half is filled with zeros; for nodes in the second group, on the other hand, the first half of each feature vector is filled with zeros and the second half one-hot encodes the relevant node's *label*. The idea is that each node in first group will have to learn to attend exclusively to its corresponding 'twin' in the second group, which holds the information needed for finding the correct label.

We would expect to see that models which are capable of dynamic attention perform better on this task. Static attention methods are expected to struggle, because every node will learn to mostly attend to a some unique common node, which is incorrect in this situation. To be precise, the static attention itself does not necessarily mean the classifier would have low capacity. This is because one one hand, the computed attention is only part of the layer — the attention weights are simply coefficients in a linear combination that gives that subsequent representation — so the layer might

still learn a sensible representation if all the other parameters are calibrated properly. On the other hand, usually we would expect to have more layers before or after the attention layer and that clearly affects the capacity of the model.

4.5.2 Empirical performance and discussion

We test this theory by running experiments with the three attention mechanisms — GAT, GATv2, and GATDB — over various realisations of our toy dataset. We train models with exactly one layer (the corresponding attention layer) on datasets with $k = 3, 4, 5, 6, 7, 8$ for at most 100 epochs each time with standard Adam optimiser with learning rate of 0.005. We report the training accuracy metric — that is, whether the corresponding layer is able to fit the data or not — a clear measure of the model's capacity. Our findings are summarised in Figure 1.

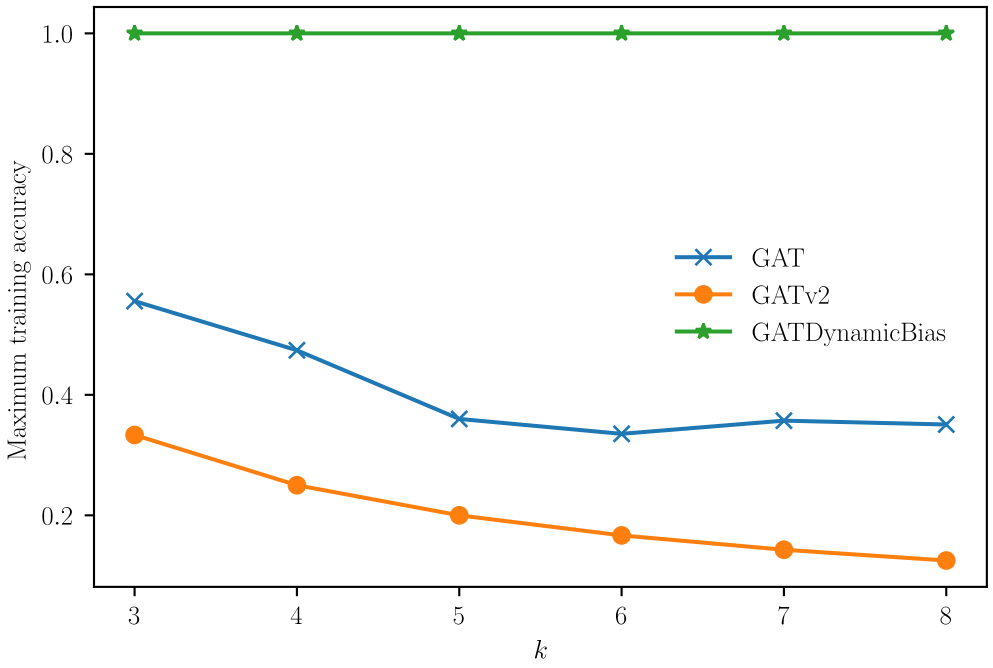

Figure 1: Results of the experimentation of different attention layers on the artificial dataset construction of Section 4.5.

Unsurprisingly, the dynamic attention of the flexible GATDB is able to perfectly fit the data every time. The flexible attention biases are able to adapt to the attentional structure in a powerful way. The only disadvantage with this approach is that we need to know the number of nodes in advanced, albeit not necessarily the structure of the graph itself.

The more surprising result is that GAT outperforms GATv2 on this task. Notice that the proof that GATv2 is dynamic actually relies on an assumption that the representation dimension can be arbitrarily large, which is not the case here because we want to do classification with this single layer so we must set the representation dimension to k. Therefore GATv2 loses its asymptotic guarantees and the order of operations in GAT make it the more powerful approach in this case.

4.6 Limitations

Adding dynamic biases increases model expressivity substantially, but can cause generalisation issues in the transductive setting as illustrated by our following claim:

Theorem 4. *Consider any undirected transductive graph modelling problem (where we consider the adjacency matrix to be constructed in the natural way - i.e. symmetric with 1 in where there is an edge between the corresponding nodes and 0 otherwise). For a GATDB layer ℓ, let ξ_{ij}^{ℓ} denote the dynamic bias from node i to node j in the network. Let $k \geqslant 0$ and let i be any node which is distance at least k from any training-set node[6]. Then, for any GATDB layer ℓ amongst the last k layers in the network, ξ_{ij}^{ℓ} is not a trained parameter, for any $j \in N(i)$.*

Proof. To show this, we have to show that the partial derivative of the loss function L with respect to ξ_{ij}^{ℓ} is zero, i.e.

$$\frac{\partial L}{\partial \xi_{ij}^{\ell}} = 0.$$

It suffices to show that, fixing all other parameters in the network, the loss L is constant with respect to this parameter.

To see this, we first observe that for any $k \geq 0$, any node's embedding in the $(k+1)^{\text{th}}$ last layer will only affect the final embeddings of nodes at a distance $\leq k$ from it. This is because, going backwards through the layers, each layer increases the receptive field of a node by 1 (which is clear from the fact that nodes can only attend to their neighbours, and that the graph is undirected), so this claim follows by a simple induction argument.

In order for a bias parameter for a node i to affect the loss L, it must affect the final embedding of a training node (which are then converted to class probabilities, independently of all other node embeddings). However, it follows from the above remark that if a node i has distance at least k from any training node, then the biases ξ_{ij}^{ℓ}, for any layer ℓ amongst the last k layers and for any $j \in N(i)$, do not affect the final embeddings of nodes further than distance k from i, so in particular they do not affect the final embedding of any training node.

Hence varying such ξ_{ij}^{ℓ} does not change L, as required. □

This theorem means that how we initialise the dynamic biases for certain non-training nodes is more important than at first glance. Otherwise, the attention distribution at test time will be influenced by unlearnt random noise from the biases. While this sounds bad, there are some reasons 'on our side' which alleviate this limitation:

- Firstly, and most importantly, with weight decay enabled, these biases decay to zero. So, in effect, at test time these biases do not exist and the model reverts to standard GATv2 attention.
- Secondly, every node is still restricted to attend to its neighbours, and the values being attended over are still learnt properly. Therefore this noise might be negligible for many applications.
- Finally, we might be able to force the biases to be learnt if we modify the structure of the graph explicitly. For example, consider the "n-path" adjacency matrix A where $A_{ij} = 1 \neq 0$ if and only if there is a path of length k between nodes i and j. The introduction of explicit structure can force the biases to influence the subsequent representations of distant nodes and thus make them learnable.

5 Empirical studies

We perform two orthogonal empirical studies in this section to complement our theoretical discussions. First, we compare a number of the attention mechanisms we've discussed on the Cora and PPI

[6]Where, by convention, every node is distance 0 from itself.

datasets; second, we perform a comparison of various versions of GAT and GATv2 on a broader range of benchmark datasets.

5.1 Study 1: Comparing different attention methods on Cora and PPI

Due to limited computational resources we could not run all of the methods discussed in this report on the wider array of datasets we consider in the next subsection, so we start by comparing the transformer- and bias-based attention mechanisms we've discussed and introduced respectively in this report on the Cora and PPI datasets used in the original GAT paper.

Both of these are node classification tasks. The Cora dataset [SNB+08] consists of 2708 papers grouped into 7 classes, with edge relations representing (symmetrised) citations between papers; the task is transductive, with a subset of the nodes' classes available at training and the remaining classes revealed only for testing. Each node's feature vector is a binary encoding of the presence/absence of each of 1433 words in the abstract of the paper. On the other hand, the PPI (Protein-Protein Interactions) dataset [ZL17] consists of a number of distinct graphs, each representing a particular human tissue; in each graph, nodes represent particular proteins, with features containing information on 50 positional gene sets, motif gene sets and immunological signatures, and edges represent interactions between pairs of proteins in cells of the relevant tissue type. Nodes are labelled by 121 gene ontology sets (this is a multi-label classification problem). The task is inductive, with full access to the graphs selected for training and a number of graphs withheld for validation and testing purposes.

5.1.1 Experimental setup

We trained the following architectures on these datasets:

- The GAT models from Section 2.2.
- GATv2 models
- Vanilla transformer models
- Universal transformer models
- GATDB models (CORA only)

We closely followed the experimental setup of [VCC+17] where relevant, in particular using the train-test split from [YCS16] for Cora.

During training, models were trained for fixed number of epochs, and the version of the model which achieved the best performance on the validation set (where 'performance' refers to 'accuracy' for CORA and 'micro F1 score' for PPI) would be selected, and its performance on the test set recorded.

GAT and GATDB. For these architectures we tried to match exactly the hyperparameters and optimisation settings set out in [VCC+17]. We used the 2-layer model mentioned in the paper with the corresponding dropout and early-stopping procedure. For the two types of layers we simply exchanged whether GAT or GATDB was used in the bigger architecture. For GATDB we tried to also use a hyperparameter ε as a multiplier of the node-biases ξ_{ij}, i.e. so that effectively the biases that get added are $\varepsilon\xi_{ij}$ and we experimented with various possible values of ε, running each experiment 5 times to select the best performing one. In the end, the best one was $\varepsilon = 1$ which corresponds to the original definition of GATDB.

GATv2, Vanilla Transformers and Universal Transformers. For these architectures we performed hyperparameter optimisation to select certain hyperparameter values. Wherever this was done, it was done using a grid search over a predefined set of values. The selected model was chosen according to the best validation set accuracy (or micro-F1 score; whichever was applicable) observed during training. Every model was trained for to a fixed number of epochs: 100 during grid search, and

then after the best hyperparameters were selected, the model was trained for a further 200 epochs, with the final model weights being selected according to the same validation set performance.

The two transformer architectures consisted of a linear layer, followed by a stack of transformer layers, followed by a final linear layer and then the appropriate activation function. This is necessary to decouple the input/output dimensions of the datasets, which are generally inappropriate to model at, with the "internal dimension" of the transformers. Moreover, on the CORA dataset only, we applied dropout annealing to these models: gradually increasing the dropout up to some limit as training progresses. We found that setting the dropout high initially slowed training, but setting the dropout too low resulted in overfitting. On the PPI dataset, unlike in [VCC+17], the batch size was set to 1 during training due to memory issues on the GPU. All of our final hyperparameters are shown in Table 2.

Table 2: Hyperparameters for Study 1.

	Cora		**PPI**	
Model	Number of Layers	Hidden Dimension Size	Number of Layers	Hidden Dimension Size
GAT	2	64	3	1024
GATv2	2	64	3	1024
Vanilla Transformer	2	64	1	128
Universal Transformer	2	128	2	128
GATDB	2	64	N/A	N/A

5.1.2 Results and discussion

The validation and test scores for each of these models on the two datasets are shown in Table 3. As can be seen, GAT matches (and slightly exceeds) the reported score in [VCC+17], which is 83.0%. It is the best architecture on this task, which is somewhat to be expected: the Cora dataset is a small citation network with a few particularly important nodes and a shortage of complex generalisable relationships to be learnt. This means that the benefits of dynamic attention are arguably unlikely to be realised on this task: even a few static attention heads can 'manually' learn which the most important nodes to attend to are, and this will transfer well to the test set. We hypothesise that likely that dynamic attention becomes more important, and more powerful, when training on either:

(a) larger, more complex transductive datasets, where there are both genuine learnable heuristics for which types of node to attend to more and too many nodes to 'memorise' the important ones, or

(b) inductive datasets, where memorisation of important nodes in the training set does not help with prediction on unseen graphs.

Indeed, on the PPI dataset, where GAT does not perform as well as claimed in [VCC+17] despite using the same parameters. In particular it differs from the results of table 1 because the models were trained for less time in this set of experiments. This was an unfortunate necessity, because certain other models required hyperparameter grid searches, so we felt it important for all architectures to train for the same amount of time. As a result, the fact that we aren't matching the performance of [VCC+17] here isn't cause for concern.

GATv2 performs better, perhaps indicating that dynamic attention is required here. For the transformers, the vanilla transformer has a lot of trouble here – we found that it was not able to learn several layers, so the best model ended up consisting of only a *single* transformer layer. Of course this has implications as to the receptive field of each node. Meanwhile, the universal transformer excels. This is interesting, because the only difference between the two architectures is that the universal

transformer applies the same layer many times. This suggests the universal transformer has a very strong inductive bias to learning a single operation which updates a node's embedding based on its neighbourhood.

For the transformer models we found it paramount to use only a handful of layers. This is contrary to NLP where excess of 8 or 16 layers are common.

Table 3: Results for Study 1.

	Cora (Accuracy)		**PPI** (Micro-F1 score)	
Model	Validation	Test	Validation	Test
GAT	81.8%	83.4%	0.782	0.812
GATv2	81.4%	80.3%	0.879	0.900
Vanilla Transformer	80.8%	78.5%	0.789	0.810
Universal Transformer	80.8%	80.4%	0.900	0.923
GATDB	77.8%	81.4%	N/A	N/A

5.2 Study 2: A deeper dive into GAT and GATv2

In the next study, we expand on our empirical results from Section 2.2 by comparing several variants of GAT and GATv2 on a broader range of benchmark graph datasets.

GAT variants. We consider two types of modification to the original GAT attention head: changing the form of the neighbourhoods each node attends over, and modifying the feature vectors of each node to include degree information. Starting with the first, we note that the original GAT paper [VCC+17] mentioned that attention coefficients could be computed over arbitrary attention masks, but that in their work each node would attend only to its first-order neighbours (including itself); we explore modifying this attention mask first by attending over second-order neighbours too, and secondly by attending only over *strict* first-order neighbours (i.e. excluding the node itself). Separately, we consider the addition of one-hot-encoded node degrees to each node's feature vector before applying the original GAT attention layer; to motivate this it suffices to note that attention methods aim to learn heuristics about which other nodes might be more important to a given one, and the connectivity of those nodes may well contain information on this.

The four GAT mechanisms we consider are thus as follows:

(a) **GAT** (as in previous sections, with nodes attending over their first-order neighbours and themselves)

(b) **GAT with attention over second-order neighbourhoods** (including self-attention)

(c) **GAT without self-attention** (nodes attend over their first-order neighbours excluding themselves)

(d) **GAT with node degrees one-hot encoded into the feature vectors** (attention is over first-order self-inclusive neighbours)

GATv2 variants. For our variants of the GATv2 attention mechanism we take inspiration from the discussion on weight-sharing and biases of Section 3.1. Recall that the authors of [BAY21] used *weight-sharing* in their experiments, setting $\mathbf{W}_1 = \mathbf{W}_2$, but that in the general case where this restriction is relaxed, there was ambiguity over which matrix should be used in the feature update rule eq. (4); we proposed three possibilities. There was also a question over whether a bias vector should be used within the LeakyReLU of the attention rule. For this study we compare the following four setups:

(a) **GATv2**: as in [BAY21], with weight-sharing and a bias term.

(b) GATv2 with no bias: still with weight-sharing.

(c) GATv2 without weight sharing (source matrix used for feature updates): in this variant the matrix $\mathbf{W_2}$ is used in the equivalent of eq. (4). A bias vector is still included.

(d) GATv2 without weight sharing (separate feature update matrix): in this variant a third matrix $\mathbf{W}$ is learnt separately from $\mathbf{W}_1, \mathbf{W}_2$ for the feature update. Biases are included.

While these variants are variously heavily-parametrised, their comparison is still of interest.

Datasets. We compare the performance of these attention mechanisms on eight transductive node classification tasks and one graph classification task. For the node classification tasks, in addition to the Cora, CiteSeer and PubMed datasets used in [VCC+17], we included the Amazon Computers and Photo datasets and the Coauthor CS and Physics datasets from [SMBG18], larger datasets they introduced to illustrate the varying performance of different models across different tasks. We also included the original full version of the Cora dataset from [BG17], which we call CoraFull. Finally, we evaluated our models on the graph classification dataset `ogbg-molhiv` from Stanford's Open Graph Benchmark [HFZ+20], a large dataset of graph representations of molecules.[7]

5.2.1 Experimental setup

In all cases, we used the architecture of the original GAT model for Cora in [VCC+17]: 2 attention layers, with 8 and 1 heads respectively, and 8-dimensional hidden features (before concatenation), separated by ELU non-linearities. The same training method was also applied, i.e. cross-entropy loss minimisation using the Adam optimizer with a learning rate of 0.005 and ℓ_2-regularisation constant 0.0005, with $p = 0.6$ dropout on every node feature and attention coefficient and early-stopping on the validation loss and score together with a patience of 100 epochs, restoring the weights achieving the lowest validation loss. We additionally applied a training cutoff of 2000 epochs.

For the graph classification tasks, latent node representations were in all cases globally mean-pooled and then fed to a 2-layer MLP with size-8 hidden layer and a ReLU non-linearity.

Some of the datasets used (CoraFull, Photo, Computers, CompSci, Physics) do not have standard train-val-test splits in the literature; for these we followed the methodology of [SMBG18] and used fixed random splits with 20 nodes of each class in the training set and 500 and 100 arbitrary nodes respectively in the validation and test sets.

5.2.2 Results and discussion

The test scores of each model on the nine datasets are shown in Table 4. In some cases we were unable to complete the experiments due to memory constraints.

Table 4: Test scores for Study 2. Failures to run an experiment due to memory shortage are represented by 'OoM'.

Model	Cora	CiteSeer	PubMed	CoraFull	Photo	Computers	CompSci	Physics	Molhiv
GAT (a)	82.9%	70.4%	77.8%	19.0%	86.6%	75.2%	90.0%	92.4%	56.3 auc
GAT (b)	79.3%	71.0%	OoM	OoM	OoM	OoM	OoM	OoM	OoM
GAT (c)	81.4%	67.9%	77.3%	20.6%	85.0%	73.2%	88.8%	91.6%	64.0 auc
GAT (d)	76.6%	61.5%	64.1%	33.6%	77.0%	71.0%	83.4%	OoM	69.2 auc
GATv2 (a)	84.0%	70.5%	78.3%	28.6%	87.4%	62.8%	91.4%	92.4%	67.7 auc
GATv2 (b)	82.5%	70.7%	78.0%	27.4%	86.2%	73.4%	91.4%	91.8%	67.9 auc
GATv2 (c)	82.9%	71.0%	77.8%	30.8%	86.6%	59.4%	90.8%	92.2%	69.5 auc
GATv2 (d)	84.0%	70.8%	78.4%	28.6%	85.8%	74.2%	90.4%	OoM	60.9 auc

For the GAT models, the original version performs best on almost all datasets. For the variants attending over different neighbourhoods, this is likely due to the expressivity benefits of self-attention

[7] Our first choice would have been to work with the large node classification tasks from OGB, but we did not have the compute available to do so.

and the lost structural information when attending over more than a node's immediate neighbours respectively; for the version with added node degree encodings this is somewhat surprising, and potentially indicates overfitting to the training set (likely not only due to the increased ease of learning relationships involving node degrees but also the higher number of weight parameters). Interestingly, the exceptions to this observation are on the CoraFull and `ogbg-molhiv` datasets, where the node-degree encoding version outperforms all others. While this could be due to a genuine benefit to learning attention heuristics based on node degrees on these datasets, there are many other factors that could be at play.[8]

For the GATv2 models, we make the following observations:

- In almost all cases removing biases appears to be bad. The major exception is on the Computers dataset, but this may be due to randomisation error.
- There was no consitent response to removing weight-sharing or adding a separate, learnable feature update matrix. This may well be due to the small size of the datasets as well as our inability to remove randomisation error.

A general observation is that the differences between the attention models we tried were smaller and more unpredictable than might be expected. While the results of this study *do* show that the dynamic attention of GATv2 is generally beneficial, we would hope to see this much more drastically, as well as more pronounced differences between our model variations, when training on larger, more modern graph datasets (such as the OGB node classification datasets) which we did not have the compute power to work with.

6 Conclusion and future work

One interesting direction of future work is to investigate problems that are inherently not solvable by either static or dynamic attention mechanisms. For example, a situation in which the edges that need to be paid attention on change according to the structure of the graph, instead of being fixed for the task (as the map ϕ in our proofs suggests) might be such problems. In those situations it might make sense to come up with a framework to better distinguish the representational capabilities of different attention layers, perhaps taking into account the the full ordering of the attention weights or the full subsequent representation of the nodes.

Furthermore, it might be useful to investigate more thoroughly what the significance of changing the adjacency matrix that the models operate on is. Theoretical results that can establish how various transformations might affect learning and expressiveness would help us design better tasks as well as better algorithms.

References

[ADH+21] Anurag Arnab, Mostafa Dehghani, Georg Heigold, Chen Sun, Mario Lučić, and Cordelia Schmid. Vivit: A video vision transformer. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6836–6846, 2021.

[BAY21] Shaked Brody, Uri Alon, and Eran Yahav. How attentive are graph attention networks? *arXiv preprint arXiv:2105.14491*, 2021.

[BG17] Aleksandar Bojchevski and Stephan Günnemann. Deep gaussian embedding of graphs: Unsupervised inductive learning via ranking. *arXiv preprint arXiv:1707.03815*, 2017.

[Cyb89] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[DB20] Vijay Prakash Dwivedi and Xavier Bresson. A generalization of transformer networks to graphs. *arXiv preprint arXiv:2012.09699*, 2020.

[8] The original GAT model is also only second-best on CiteSeer, but this is most likely due to randomisation error.

[DGV+18] Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. *arXiv preprint arXiv:1807.03819*, 2018.

[Fun89] Ken-Ichi Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural networks*, 2(3):183–192, 1989.

[GSR+20] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Message passing neural networks. In *Machine learning meets quantum physics*, pages 199–214. Springer, 2020.

[HDWS20] Ziniu Hu, Yuxiao Dong, Kuansan Wang, and Yizhou Sun. Heterogeneous graph transformer. In *Proceedings of The Web Conference 2020*, pages 2704–2710, 2020.

[HFZ+20] 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.

[Hor91] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

[HSW89] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[HVU+18] Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Noam Shazeer, Ian Simon, Curtis Hawthorne, Andrew M Dai, Matthew D Hoffman, Monica Dinculescu, and Douglas Eck. Music transformer. *arXiv preprint arXiv:1809.04281*, 2018.

[KO20] Dongkwan Kim and Alice Oh. How to find your friendly neighborhood: Graph attention design with self-supervision. In *International Conference on Learning Representations*, 2020.

[LLK+19] Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In *International Conference on Machine Learning*, pages 3744–3753. PMLR, 2019.

[NNP19] Dai Quoc Nguyen, Tu Dinh Nguyen, and Dinh Phung. Universal graph transformer self-attention networks. *arXiv preprint arXiv:1909.11855*, 2019.

[RBX+20] Yu Rong, Yatao Bian, Tingyang Xu, Weiyang Xie, Ying Wei, Wenbing Huang, and Junzhou Huang. Self-supervised graph transformer on large-scale molecular data. *Advances in Neural Information Processing Systems*, 33:12559–12571, 2020.

[SGT+08] 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.

[SMBG18] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*, 2018.

[SNB+08] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.

[VCC+17] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[VSP+17] 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.

[WTWS19] Xing Wang, Zhaopeng Tu, Longyue Wang, and Shuming Shi. Self-attention with structural position representations. *arXiv preprint arXiv:1909.00383*, 2019.

[YCS16] Zhilin Yang, William Cohen, and Ruslan Salakhudinov. Revisiting semi-supervised learning with graph embeddings. In *International conference on machine learning*, pages 40–48. PMLR, 2016.

[YJK+19] Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang, and Hyunwoo J Kim. Graph transformer networks. *Advances in neural information processing systems*, 32, 2019.

[ZL17] Marinka Zitnik and Jure Leskovec. Predicting multicellular function through multi-layer tissue networks. *Bioinformatics*, 33(14):i190–i198, 2017.

[ZZXS20] Jiawei Zhang, Haopeng Zhang, Congying Xia, and Li Sun. Graph-bert: Only attention is needed for learning graph representations. *arXiv preprint arXiv:2001.05140*, 2020.