DIFFERENT INDUCTIVE BIASES CAN LEAD TO SIMILAR PERFORMANCE VIA DISTINCT ALGORITHMS

Batu El
Stanford University
batuel@stanford.edu

Deepro Choudhury
University of Oxford
deepro.choudhury20@imperial.ac.uk

Chaitanya Joshi
University of Cambridge
ckj24@cam.ac.uk

ABSTRACT

Graph Neural Networks (GNNs) have emerged as a useful class of neural networks with specific inductive biases that make them well-suited for processing graph-structured datasets. Despite the remarkable success of GNNs in several domains, the algorithms they learn from data are not well understood. Leveraging the mathematical equivalence between message passing in GNNs and variations of transformer attention, we investigate the information flow patterns inside graph neural networks with different inductive biases. We present a simple framework to combine the attention in multi-head and multi-layer models in a way that reflects the information flow within the network. In our experiments with 4 architectures across 7 node classification tasks, we (i) analyze the relationship between learned information flow patterns and the underlying graph structure and (ii) compare the information flow patterns learned by different GNN architectures. Firstly, we find that, in general, when the architecture does not impose the graph structure, the information flow patterns within the network do not reflect the graph structure. Secondly, we observe that on small heterophilous graphs, such as Texas, different GNN architectures achieve similar performance, yet a closer look at their information flow patterns suggests that they do so by implementing different algorithms. This observation underscores the necessity for more comprehensive evaluations that go beyond performance comparisons.[1]

1 INTRODUCTION

Transformers are attentional GNNs on fully-connected graphs (Rush, 2018; Joshi, 2020). Conversely, message passing in GNNs can be interpreted as a generic attention mechanism. In recent years, these architectures have achieved remarkable success in various domains (Brown et al., 2020; Jumper et al., 2021). However, despite their significant achievements and widespread deployment in real-world systems, the mechanisms through which they make decisions are still not well understood. In this paper, we investigate the information flow inside GNNs to develop insights about the processes learned by the model. To that end, we (i) analyze the relationship between learned information flow patterns and the underlying graph structure and (ii) compare the information flow patterns learned by different GNN architectures. The main theoretical contribution of our work is a framework to represent the information flow across the layers of a GNN in a single pseudo-adjacency matrix. In our experiments, we observe that the information flow pattern does not recover the underlying graph structure when the architecture does not explicitly bias the nodes to attend to their local neighbors. On small heterophilous graphs, such as Texas, different GNN architectures achieve similar performance, but they do so by learning different algorithms. Moreover, in certain dataset-model pairs, we observe pivotal nodes, certain nodes that seem to disproportionately influence the predictions for all the nodes in a graph regardless of the n-hop distance between the current node and the pivotal node.

[1] https://github.com/batu-el/understanding-inductive-biases-of-gnns

2 BACKGROUND

A graph, $G = (V, E)$, is a mathematical object that represents entities, V, and relationships, E, among the entities (Bronstein et al., 2021). A GNN trained for node classification learns a function f that maps each node $v_i \in V$ to a class y_i. Each v_i is represented with its associated feature vector, $x_i \in \mathbb{R}^d$, where d is the dimension of the input features. We consider tasks where the relationships between the nodes are binary. Therefore, E is fully characterized by an adjacency matrix A, where $a_{ij} = 1$ whenever node v_i directly relates to node v_j and $a_{ij} = 0$ otherwise. Hence, we describe these graphs with two matrices: $A \in \mathbb{R}^{n \times n}$ and $X \in \mathbb{R}^{n \times d}$, where n is the number of nodes in the graph. We denote the 1-hop neighbourhood of a node v_i as a set of nodes $\mathcal{N}^1(v_i) = \{v_j \mid A_{ij} = 1\}$.[2]

Unlike feed-forward networks that map input features x_i to classes y_i via a function $g(x_i) = y_i$ that is applied to each input individually, GNNs allow us to consider all node features in X and the relationships between them simultaneously while predicting the class of a node. Hence, a function f implemented by a GNN for a node classification task is a permutation-equivariant map from the tuple of matrices (X, A) to a vector y. In particular, $f(X, A) = y$ and $f(\mathbf{P}X, \mathbf{P}A\mathbf{P}^T) = \mathbf{P}y$ for any permutation matrix $\mathbf{P}$ (Bronstein et al., 2021).

2.1 ATTENTION AND MESSAGE PASSING

GNNs are often presented via the following message-passing formalization. This formula describes how a node's representation in layer ℓ, $h_i^\ell \in \mathbb{R}^{d_{\text{model}}}$, is updated in a GNN layer using the messages passed from the other nodes in the graph. The message passing operation maps h_i^ℓ to $h_i^{\ell+1}$ as follows:

$$h_i^{\ell+1} = \phi\left(h_i^\ell, \oplus\Big(\psi\left(h_i^\ell, h_j^\ell, e_{ij}\right)\Big)\right)$$

where ψ determines how much information is passed from h_i^ℓ to h_j^ℓ as a function of the representations of nodes i and j and the relationship between them, e_{ij}. The information node i chooses to collect from all nodes in the graph is combined using a permutation-invariant function $\oplus$, which is then combined with the representation of node i from layer ℓ via ϕ. In the datasets we use in our experiments, e_{ij} is a binary value equal to a_{ij} in the adjacency matrix.

Many GNN architectures, such as GCN (Kipf & Welling, 2017a) and GAT (Veličković et al., 2018; Brody et al., 2022) impose the restriction that whenever $a_{ij} = 0$, $\psi\left(h_i^\ell, h_j^\ell, e_{ij}\right)$ also equals 0. Other architectures, such as GraphFormer (Ying et al., 2021) and graph transformer (Dwivedi & Bresson, 2021), adopt a more flexible approach by allowing the nodes to collect information from outside of their neighborhood.

Based on this formulation, we define an attention matrix, $\mathbb{A} \in \mathbb{R}^{n \times n}$, as a function of H and A, where $H^\ell \in \mathbb{R}^{n \times d_{\text{model}}}$ contains the representation of a node from the graph in each of its rows and A is the adjacency matrix. A row of the attention matrix, $\mathbb{A}_i = \tilde{\psi}\left(h_i^\ell, H^\ell, A_i\right)$ is a vector that denotes how much node i attends to all the other nodes in the graph in layer ℓ. $\mathbb{A}_{ij}$ can be calculated by comparing h_i^ℓ and h_j^ℓ while considering the connection between the two nodes, A_{ij}. The rows of $\mathbb{A}$ can be normalized so that the sum of the values in $\mathbb{A}_i$ add up to 1 via the softmax function (Vaswani et al., 2017).

3 DATA

In our experiments, we use 7 node classification datasets with different levels of homophily, which quantifies the likelihood of a connection between the nodes that are *similar*. We use three homophily metrics (Node Homophily, Edge Homophily, and Adjusted Homophily) that quantify the likelihood of a connection between the nodes that *share the same class*. In Appendix A, we discuss the three approaches.[3] Table 1 presents the datasets we use in our experiments and relevant statistics. Cora and Citeseer (Yang et al., 2016) are homophilous citation networks, whose node features are

[2] More generally, the n-hop neighborhood of v_i is represented by $\mathcal{N}^n(v_i) = \{v_j \mid A_{ij}^n \neq 0\}$, where $A^n = \prod_{t=1}^n A$.

[3] We calculate homophily metrics using the DGL implementation.

Metric	Cora	Citeseer	Chameleon	Squirrel	Cornell	Texas	Wisconsin
Node Homophily	82.5	70.6	10.4	8.9	10.6	6.5	17.2
Edge Homophily	81.0	73.6	23.5	22.4	13.1	10.8	19.6
Adjusted Homophily	77.1	67.1	3.3	0.7	-21.1	-25.9	-15.2
Number of Nodes	2708	3327	2277	5201	183	183	251
Number of Edges	10556	9104	36101	217073	298	325	515
Number of Classes	7	6	5	5	5	5	5

Table 1: Summary of Homophily, Network Structure, and Number of Classes in our Datasets.

bag-of-words representations of the abstracts of scientific papers and edges represent the citations between papers. The classes are research topics associated with each article. Similarly, Squirrel and Chameleon (Rozemberczki et al., 2019) are graphs where the nodes represent articles from the English Wikipedia, where the node features are the presence of certain nouns in the articles, and the edges represent (mutual) hyperlinks between articles. The five classes categorize articles into groups based on their average monthly traffic (Pei et al., 2020). The Cornell, Texas, and Wisconsin datasets are part of the WebKB dataset (Pei et al., 2020), which is a dataset of webpages from the computer science departments of universities. Nodes are webpages and edges are hyperlinks between them. The node features are the bag-of-words representation of the webpages, and the target is the category of the webpage.

4 MODELS

In our experiments, we focus on **attention-only** models. Our goal is not to train state-of-the-art models, but to understand simple GNNs that perform reasonably well by analyzing their attention patterns. In the simple one-layer & one-head case, all of our models have the following common high-level structure:

$$\mathrm{X} \longrightarrow H^{(0)} \longrightarrow H^{(1)} \longrightarrow \text{Logits}$$

We do not use MLP layers, and the only non-linearity is the ReLU activation applied prior to the residual connection that implements the update function ϕ. Hence, our layers have the following structure:

$$H^{(l+1)} = H^{(l)} + \text{ReLU}(\mathbb{A}V^{(l)}) \text{ , where } V^{(l)} = H^{(l)}W^{(l)}$$

During training, we use a dropout layer (Srivastava et al., 2014) before the residual connection. In all experiments, we set the dropout value to 0.5, which mitigates overfitting. We set $d_{\text{model}} = 128$, hence $H^{\ell} \in \mathbb{R}^{n \times 128}$. Appendix B describes the computation of $\mathbb{A}$ in our models.

4.1 INDUCTIVE BIASES

Our models only differ in how they calculate $\mathbb{A}$ in two different dimensions: parametrization and sparsity. Figure 1 visualizes these dimensions.

On the **sparse-dense axis**, our **sparse** $\mathbb{A}$ models restrict the attention to the neighborhood of the node, imposing the graph structure as an inductive bias of the architecture. This makes sparse models less flexible compared to the **dense** $\mathbb{A}$ models, which allow each node in the graph to attend to all the other nodes regardless of the presence of edges between them. On the **constant-learned axis**, our **constant** $\mathbb{A}$ models fix the attention to be inversely proportional to the square root of the degree of the node being attended to. On the other end of the spectrum, the **learned** $\mathbb{A}$ models learn *"how much to attend to other nodes"* during training. In between the two extremes, we have models with **learned but biased** $\mathbb{A}$, where $\mathbb{A}$ is learned, but biased to be inversely proportional to the n-hop distance between the nodes so that attention coefficients between closer nodes (in terms of n-hop distance) are amplified, and attention coefficients between farther nodes are suppressed.

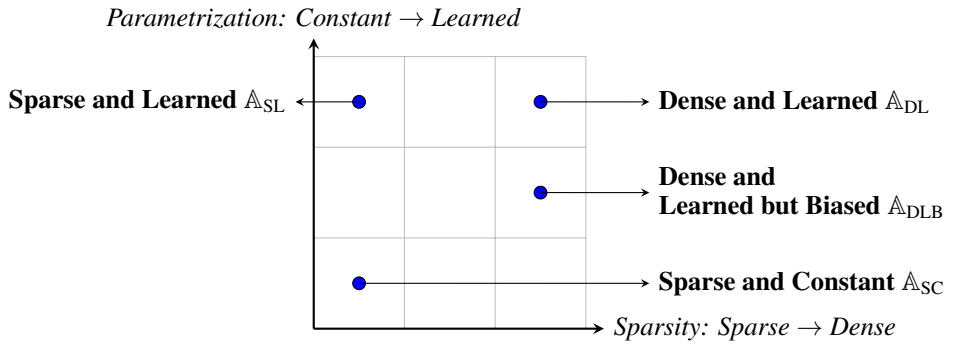

Figure 1: Models with different inductive biases.

In our experiments, we train models with 4 different attention configurations, which we describe below.

4.1.1 SPARSE AND CONSTANT (SC):

Every node attends only to its neighbors, and attention paid to every neighbor is inversely proportional to the square root of the degree of the neighbor. This recovers Graph Convolutional Networks (GCN) (Kipf & Welling, 2017b). We refer to the attention matrix here as $\mathbb{A}_{\text{SC}}$. [4]

4.1.2 SPARSE AND LEARNED (SL):

Like $\mathbb{A}_{\text{SC}}$, nodes attend only to their neighbors. However, unlike $\mathbb{A}_{\text{SC}}$, how much a node attends to its neighbors is not a set value that is a function of the adjacency matrix. Instead, attention is a learned function of the node features. This recovers the attention mechanism in Graph Attention Networks (GATs) (Veličković et al., 2018; Brody et al., 2022). We refer to the attention matrix here as $\mathbb{A}_{\text{SL}}$.

4.1.3 DENSE AND LEARNED BUT BIASED (DLB):

The attention mechanism in our dense models are not restricted to the neighborhood of the current focus of attention, so any node has the freedom to attend to any other node in the graph. However, $\mathbb{A}_{\text{DLB}}$ biases the attention coefficients with the inverse n-hop distance between the nodes, amplifying the attention coefficients between closer nodes and suppressing attention coefficients between farther nodes. In particular, we add the bias matrix to the learned attentions to obtain $\mathbb{A}_{\text{DLB}}$. The i^{th} row of the bias matrix contains values that are *inversely proportional* to the shortest path between the focus of attention, node i, and the nodes attended to by node i, scaling up the influence of nodes that are in proximity to node i in terms of n-hop neighborhood. This recovers the Graphormer (Ying et al., 2021).

4.1.4 DENSE AND LEARNED (DL):

Finally, our most flexible model allows each node to attend to every other node in the graph. The attention coefficients are learned in an unbiased manner, in a way that is analogous to Transformer attention (Vaswani et al., 2017). This recovers the Graph Transformer (Yun et al., 2020).[5]

[4] $\mathbb{A}_{\text{SC}} = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$, where $\tilde{A} = A + I$ is the adjacency matrix with added self-connections and $\tilde{D}$ is a diagonal degree matrix containing the row-sums of $\tilde{A}$, i.e. $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$.

[5] In the previous three models, the attention mechanism was the *only* avenue through which the model uses the graph structure. Hence, in the absence of attention bias, a dense transformer learns function that maps the node features X to the labels y without considering the graph structure. To inform our model about the graph structure, we provide the model with a 16-dimensional Laplacian eigenvector positional encoding (Dwivedi & Bresson, 2021) associated with each node (which the model can choose to use). The model uses a linear map to map this to a $d_{\text{model}} = 128$ dimensional vector before adding it to the representation of the associated node in $H^{(0)}$.

5 AGGREGATING ATTENTION ACROSS HEADS AND LAYERS

We experiment with 1 and 2 layer models with 1 or 2 attention heads. A key challenge in analyzing attention in multi-head and multi-layer models is coming up with a good way to aggregate attention matrices learned in different heads and across layers. In this section, we present our approach.

5.1 AGGREGATING ATTENTION ACROSS HEADS

We begin by checking whether the attention matrices in different heads learn similar patterns. Figure 4 demonstrates the correlation between the attention values learned by the two heads in 1-layer 2-head models. Overall, we observe that the two heads in SL models are learning notably similar values when trained on datasets with high homophily levels. The attention values learned from more heterophilous datasets by the SL models (row 1) and DLB models (row 2) also demonstrate positive correlation. This pattern is also present but weaker in the DL models (row 3). We observe that the attention patterns in different heads are often positively correlated and, more importantly, are *not negatively correlated* in our models. This allows us to combine the attention matrices across heads by averaging them, as small attention values learned by one head are unlikely to offset large attention values learned by the other head: $\mathbb{A}_{\text{Agg.}} = \frac{1}{N_H} \sum_{i=1}^{N_H} \mathbb{A}_{Hi}$, where N_H is the number of heads and $\mathbb{A}_{Hi}$ is the attention matrix for the i-th head. Although we find this approach to be straightforward and often effective, we acknowledge that it is a rough approximation not without its limitations.

5.2 AGGREGATING ATTENTION ACROSS LAYERS

To develop intuition about how to combine attention values across layers, we plot the attention values learned in the first and second layers of 2-layer 1-head transformer models. We observe that the attention matrices in different layers still demonstrate a positive correlation; however, the learned attention values are more spread out around the $x = y$ line, indicating that attention across layers is less similar compared to attention across heads.

Combining attention across layers is more nuanced than it first appears. We propose multiplying the attention matrices from successive layers to capture how information flows through the network. Specifically, if node i attends to node j in the second layer, then i indirectly attends to all the nodes that j attended to in the first layer. Formally, we define $\mathbb{A}_{\text{Agg.}} = \mathbb{A}_{L2}\mathbb{A}_{L1}$. In this formulation, the i-th row of $\mathbb{A}_{\text{Agg.}}$ is a linear combination of the rows of $\mathbb{A}_{L1}$, weighted by the attention coefficients in the i-th row of $\mathbb{A}_{L2}$. Consequently, if j attends to k in the first layer, and i attends to j in the second layer, this matrix multiplication captures the flow of information from k to j to i.

When combining attentions in multi-layer multi-head models, we first aggregate across heads by averaging, then matrix multiply across layers to obtain a *single aggregate attention matrix*. We note that, alternatively, one can aggregate for each unique sequence of heads across all layers, before averaging these values to obtain $\mathbb{A}_{\text{Agg.}}$.

6 RESULTS

6.1 MODEL PERFORMANCE

Table 3 demonstrates the performance metrics from our experiments. As expected, we observe that the models that restrict attention to the neighbors achieve higher accuracies in homophilous graphs, such as Core and Citeseer. Removing this constraint seems to improve model performance in smaller heterophilous graphs, such as Cornell, Texas, and Wisconsin. Moreover, biasing attention towards the neighbors without restricting it achieves higher accuracies in graphs with moderate homophily levels, such as Chameleon and Squirrel. Overall, we do not observe significant changes in model performance as we increase the number of heads and layers.

6.2 ATTENTION PATTERNS

Figure 2 shows the distribution of attention weights between neighbors and non-neighbors for SL, DLB, and DL models. In 1-layer SL nodes can only attend to their neighborhood, whereas in 2-

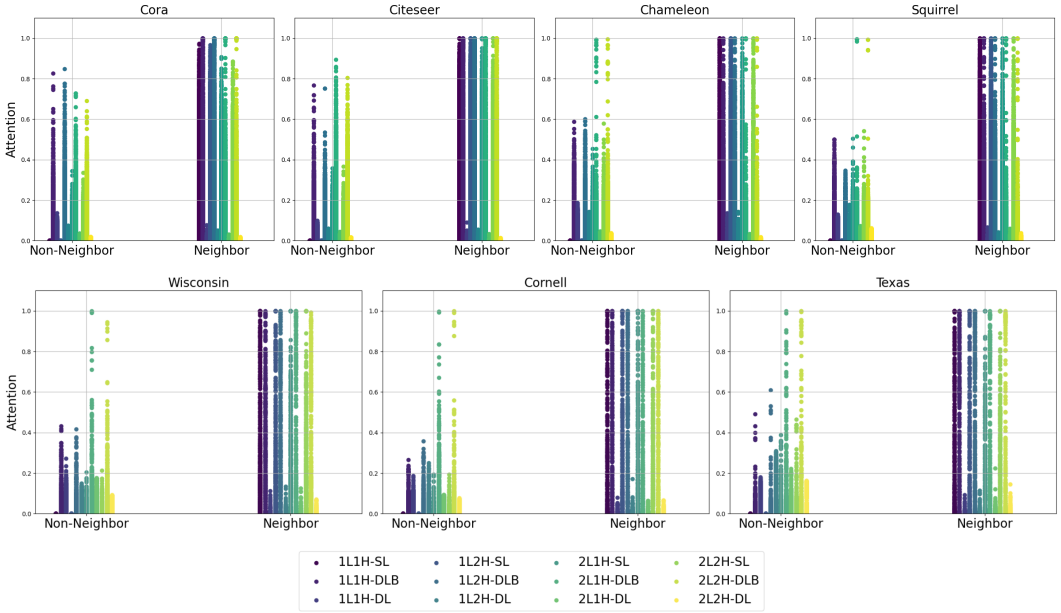

Figure 2: Attention in SL, DLB, and DL models. We show the attention patterns in 1-layer 1-head (1L1H), 1-layer 2-head (1L2H), 2-layer 1-head (2L1H), and 2-layer 2-head (2L2H) models. For each model, we calculate the *aggregated attention* as described in Section 5. We then plot attention paid to neighbours and non-neighbours in two separate buckets. If a node is attending to itself, we classify that as inside the neighbourhood.

layer, attention to non-neighbors emerging due to the information propagation within the network. We observe that our attention aggregation scheme captures this phenomenon. We also note that DLB models focuses most of their attention to the neighborhood; however, in the absence of bias, DL models distribute attention roughly equally between neighbors and non-neighbors, which can be observed in Table 4 that reports the ratio of average attention to non-neighbors and average attention to neighbors (see row 3 of Table 4). Similarly, in Figure 6, instead of grouping the attention values into neighbor and non-neighbor buckets, we show how attention is distributed across the n-hop neighborhood of the nodes for our 1 layer 1 head models. Notably, the attention in DLB models (row 2) mostly focuses on nodes that are closer in terms of n-hop neighborhood, while attention in DL (row 3) is relatively flat across the n-hop neighborhood.

6.3 Recovering the Graph Structure

	1L1H		1L2H		2L1H		2L2H	
	DLB	DL	DLB	DL	DLB	DL	DLB	DL
Cora	46.75	0.13	61.58	0.38	36.58	0.21	37.94	0.32
Citeseer	28.24	0.22	41.51	0.69	33.70	0.23	39.02	0.13
Chameleon	51.01	0.76	58.63	0.72	28.32	1.24	33.98	0.39
Squirrel	84.11	0.07	86.31	0.07	49.22	0.43	50.10	0.51
Cornell	57.44	0.21	61.38	0.85	43.51	1.25	48.95	1.04
Texas	58.37	0.61	58.31	1.22	44.83	3.69	47.14	2.95
Wisconsin	58.06	1.48	58.78	0.95	41.94	1.06	41.12	1.47

Table 2: F1-scores between the adjacency matrix and the pseudo-adjacency matrix constructed by thresholding attention as described in Appendix F for DLB and DL models. We run this analysis on 1 layer 1 head (1L1H), 1 layer 2 head (1L2H), 2 layer 1 head (2L1H), and 2 layer 2 head (1L1H) models. To calculate the F1-scores, we flatten both of these matrices and treat comparison as a binary classification task.

Next we check whether the learned attention in our dense models matches the underlying graph structure. We construct pseudo-adjacency matrices by thresholding attentions as described in Appendix F. In Table 2, we check if the entries in this pseudo-adjacency matrix match those in the adjacency matrix using element-wise $F1$-scores as our similarity metric.[6] We observe that for DL model, $F1$-scores are always near 0, indicating that the model does not recover the graph structure in its attention pattern. However, the pseudo-adjacency matrices of the DLB models achieve over 50% F1-scores. We observe that $F1$-scores are higher for 2-head models, possibly because averaging across heads removes some of the randomness in the learned attentions.

6.4 A Closer Look at Pseudo-Adjacency Matrices

Finally, we inspect the pseudo-adjacency matrices constructed by thresholding the attention to compare the information flow patterns inside different networks. In Figure 3, we make two key observations:

1. Diagonals in Pseudo-Adjacency: We observe that the pseudo-adjacency matrix learned by DLB models often contains high values on the diagonal (Figure 3), meaning that the nodes mostly attend to themselves. This observation *may suggest* that the network solves the classification task mainly using node features, aggregating little information from the representations of other nodes in the graph. We note that this self-attention pattern is more visible in two-layer models.

2. Verticals in Pseudo-Adjacency: We observe vertical lines in the pseudo-adjacency matrices learned by the DL model. These vertical lines indicate that all nodes focus their attention on certain *reference nodes* in the graph. Based on this observation, we hypothesize that to solve the classification task, the DL model implements an algorithm that compares the given node with reference nodes in order to determine the predicted label of the given node. We observe that this is the common pattern learned by the DL model across all datasets. There seem to be fewer but more pronounced reference nodes in two-layer models and large heterophilous graphs, such as Chameleon and Squirrel.

3. Similar Performance & Different Algorithm: Remarkably, DLB and DL models achieve ***similar accuracies*** on heterophilous tasks, as we have demonstrated in Table 3. However, our investigation of their attention patterns suggests that they solve the task at hand by implementing ***completely different algorithms***.

7 Related Work

Our work lies at the intersection of GNN explainability and Transformer interpretability.

GNN Explainability. GNNExplainer proposed by Ying et al. (2019) was one of the earliest attempts to develop a framework for explaining the predictions of GNNs. Ying et al. (2019) proposed to find a subgraph of a node's computational graph that influences the predictions the most. Since then, many works have expanded this idea, developing concept-based methods Magister et al. (2021), counterfactual explanations Lucic et al. (2022) and generative explanations Yuan et al. (2020). However, these methods do not focus on analyzing the information flow within the network, which is the main focus of our work.

Transformer Interpretability. Transformer interpretability and, in particular, mechanistic interpretability, aim to predict the behaviour of the neural networks by reverse-engineering model computation into human-understandable components (Olah, 2022; Meng et al., 2022; Geiger et al., 2021; Geva et al., 2021). Notably, previous works have demonstrated that studying the underlying mechanisms of a model can allows us to better predict out-of-distribution behaviour Mu & Andreas (2021), fix model errors Hernandez et al. (2022); Vig et al. (2020), and understand emergent behaviour Barak et al. (2023); Wei et al. (2022). It encompasses understanding features learned by models Olah et al. (2017); Elhage et al. (2022), mathematical frameworks for understanding architectures Elhage et al. (2021) and efforts to find circuits in models Nanda et al. (2023); Cammarata et al. (2020); Chughtai et al. (2023); Wang et al. (2023).

[6] Sparsity of adjacency matrices lead to significant class imbalance: the majority of the values in both the adjacency and the pseudo-adjacency matrices are 0s. Therefore, accruacy is not an informative metric in this context. We observe around 99% accuracy for both models.

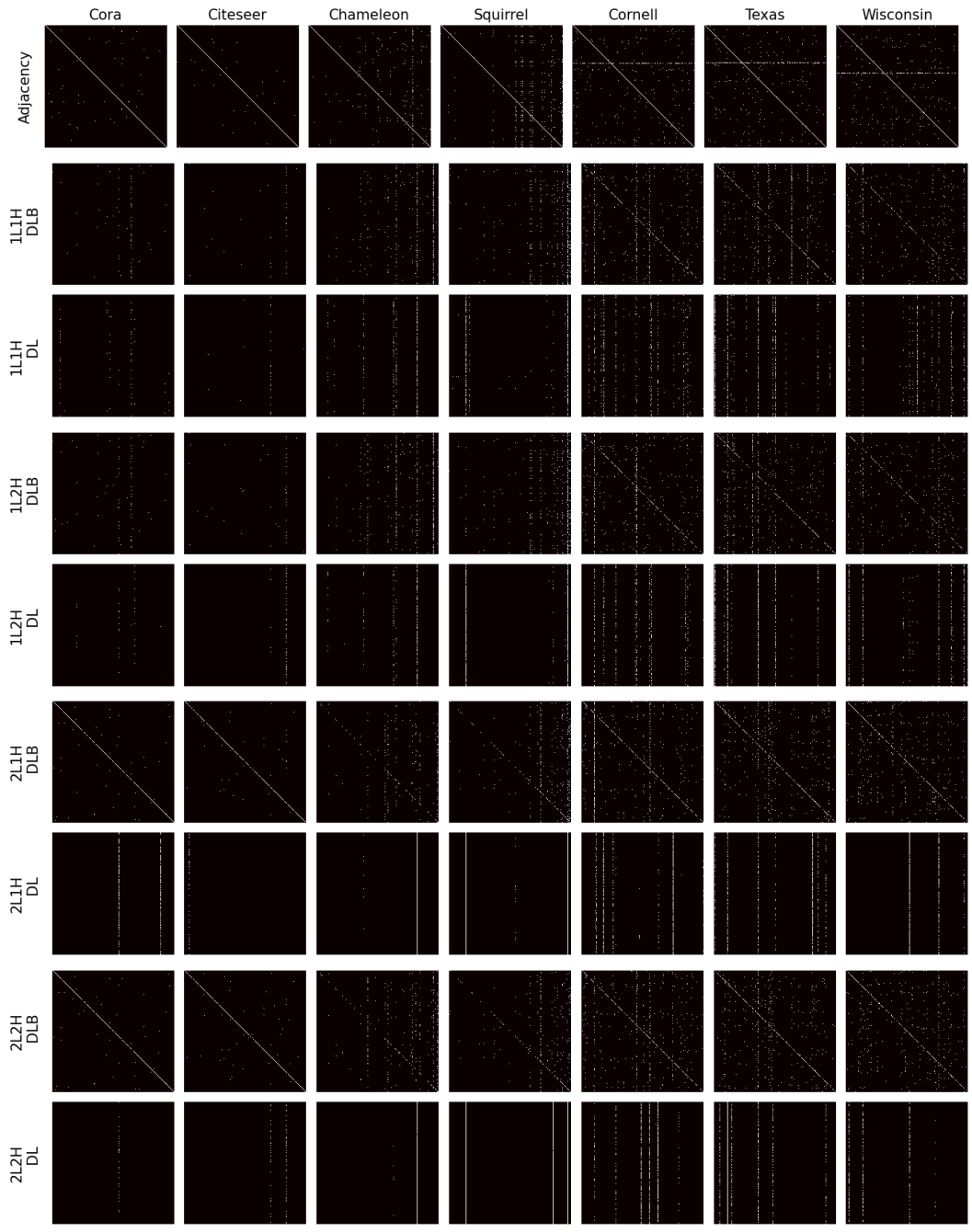

Figure 3: Heatmaps for the adjacency matrix (row 1) and the pseudo-adjacency matrices (rows 2-9) constructed by thresholding attention as described in Appendix F.

8 DISCUSSION

In this paper, we developed a framework to analyze the attention patterns learned by GNNs trained on node classification tasks. Our framework formulates message passing as a generic attention mechanism and offers a mathematically principled method to aggregate attention across heads and across layers in a way that captures how information flows within the network. Previous works on Transformer circuits have focused on **discrete interactions** between abstract features extracted from the hidden representations via sparse auto-encoders (Bricken et al., 2023) or by averaging across inputs (Meng et al., 2022). This approach reduces the network to a series of *if-then* rules that need to be identified among exponentially many possibilities (Wang et al., 2023). In contrast, our approach aims to capture the **continuous interactions** between abstract features by observing the information flow within the network.

Using this framework, we analyzed whether the learned attention patterns reflect the underlying graph structure. In section 6.4, we demonstrated that networks with different inductive biases can achieve similar performance on a task while implementing distinct algorithms. This observation prompts a critical question: how can we determine which of these algorithms is correct? Consequently, it underscores the necessity for more comprehensive and holistic evaluations that go beyond performance comparisons.

One of the limitations of our approach for combining attention across layers is the oversight of residual connections within the network, which ensure the nodes have access to their own representations from the previous layer even when they do not attend to themselves. Our current framework to combine attention matrices across layers does not consider this channel of information flow. We note that this limitation may be resolved by slightly modifying our methodology. A more significant challenge is posed by the non-linear activation functions that mediate the information flow within a network in a more nuanced way.

Future work can combine the analysis of the information flow with an understanding of the abstract concepts encoded in the hidden representations to build a better understanding of the algorithms implemented by the model.

REFERENCES

Sami Abu-El-Haija, Bryan Perozzi, Amol Kapoor, Hrayr Harutyunyan, Nazanin Alipourfard, Kristina Lerman, Greg Ver Steeg, and Aram Galstyan. Mixhop: Higher-order graph convolution architectures via sparsified neighborhood mixing. In *International Conference on Machine Learning (ICML)*, 2019.

Boaz Barak, Benjamin L. Edelman, Surbhi Goel, Sham Kakade, Eran Malach, and Cyril Zhang. Hidden progress in deep learning: Sgd learns parities near the computational limit, 2023.

Trenton Bricken, Adly Templeton, Joshua Batson, Brian Chen, Adam Jermyn, Tom Conerly, Nick Turner, Cem Anil, Carson Denison, Amanda Askell, Robert Lasenby, Yifan Wu, Shauna Kravec, Nicholas Schiefer, Tim Maxwell, Nicholas Joseph, Zac Hatfield-Dodds, Alex Tamkin, Karina Nguyen, Brayden McLean, Josiah E Burke, Tristan Hume, Shan Carter, Tom Henighan, and Christopher Olah. Towards monosemanticity: Decomposing language models with dictionary learning. *Transformer Circuits Thread*, 2023. https://transformer-circuits.pub/2023/monosemantic-features/index.html.

Shaked Brody, Uri Alon, and Eran Yahav. How attentive are graph attention networks?, 2022.

Michael M. Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges, 2021.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020. URL `https://arxiv.org/abs/2005.14165`.

Nick Cammarata, Shan Carter, Gabriel Goh, Chris Olah, Michael Petrov, Ludwig Schubert, Chelsea Voss, Ben Egan, and Swee Kiat Lim. Thread: Circuits. *Distill*, 2020. doi: 10.23915/distill.00024. https://distill.pub/2020/circuits.

Bilal Chughtai, Lawrence Chan, and Neel Nanda. A toy model of universality: Reverse engineering how networks learn group operations, 2023.

Vijay Prakash Dwivedi and Xavier Bresson. A generalization of transformer networks to graphs, 2021.

Nelson Elhage, Neel Nanda, Catherine Olsson, Tom Henighan, Nicholas Joseph, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Nova DasSarma, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. A mathematical framework for transformer circuits. *Transformer Circuits Thread*, 2021. https://transformer-circuits.pub/2021/framework/index.html.

Nelson Elhage, Tristan Hume, Catherine Olsson, Nicholas Schiefer, Tom Henighan, Shauna Kravec, Zac Hatfield-Dodds, Robert Lasenby, Dawn Drain, Carol Chen, Roger Grosse, Sam McCandlish, Jared Kaplan, Dario Amodei, Martin Wattenberg, and Christopher Olah. Toy models of superposition, 2022.

Atticus Geiger, Hanson Lu, Thomas F Icard, and Christopher Potts. Causal abstractions of neural networks. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, 2021. URL `https://openreview.net/forum?id=RmuXDtjDhG`.

Mor Geva, Roei Schuster, Jonathan Berant, and Omer Levy. Transformer feed-forward layers are key-value memories. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (eds.), *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 5484–5495, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.446. URL `https://aclanthology.org/2021.emnlp-main.446`.

Evan Hernandez, Sarah Schwettmann, David Bau, Teona Bagashvili, Antonio Torralba, and Jacob Andreas. Natural language descriptions of deep visual features, 2022.

Chaitanya Joshi. Transformers are graph neural networks. *The Gradient*, 2020.

John Jumper, Ritchie Evans, Alexander Pritzel, and et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596:583–589, 2021. doi: 10.1038/s41586-021-03819-2.

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

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2017b. URL https://arxiv.org/abs/1609.02907.

Boris Knyazev, Graham W. Taylor, and Mohamed R. Amer. Understanding attention and generalization in graph neural networks, 2019.

Ana Lucic, Maartje ter Hoeve, Gabriele Tolomei, Maarten de Rijke, and Fabrizio Silvestri. Cf-gnnexplainer: Counterfactual explanations for graph neural networks, 2022.

Lucie Charlotte Magister, Dmitry Kazhdan, Vikash Singh, and Pietro Liò. Gcexplainer: Human-in-the-loop concept-based explanations for graph neural networks, 2021.

Kevin Meng, David Bau, Alex J Andonian, and Yonatan Belinkov. Locating and editing factual associations in GPT. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/pdf?id=-h6WAS6eE4.

Jesse Mu and Jacob Andreas. Compositional explanations of neurons, 2021.

Neel Nanda, Lawrence Chan, Tom Lieberum, Jess Smith, and Jacob Steinhardt. Progress measures for grokking via mechanistic interpretability. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/pdf?id=9XFSbDPmdW.

M. E. J. Newman. Mixing patterns in networks. *Phys. Rev. E*, 67:026126, Feb 2003. doi: 10.1103/PhysRevE.67.026126. URL https://link.aps.org/doi/10.1103/PhysRevE.67.026126.

Chris Olah. Mechanistic interpretability, variables, and the importance of interpretable bases. https://www.transformer-circuits.pub/2022/mech-interp-essay, Jun 2022. Accessed: 2024-03-10.

Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. Feature visualization. *Distill*, 2017. doi: 10.23915/distill.00007. https://distill.pub/2017/feature-visualization.

Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. Geom-gcn: Geometric graph convolutional networks. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/pdf?id=S1e2agrFvS.

Oleg Platonov, Denis Kuznedelev, Artem Babenko, and Liudmila Prokhorenkova. Characterizing graph datasets for node classification: Homophily-heterophily dichotomy and beyond. In *The Second Learning on Graphs Conference*, 2023. URL https://openreview.net/pdf?id=D4GLZkTphJ.

Benedek Rozemberczki, Carl Allen, and Rik Sarkar. Multi-scale attributed node embedding. *CoRR*, abs/1909.13021, 2019. URL http://arxiv.org/abs/1909.13021.

Alexander Rush. The annotated transformer. In Eunjeong L. Park, Masato Hagiwara, Dmitrijs Milajevs, and Liling Tan (eds.), *Proceedings of Workshop for NLP Open Source Software (NLP-OSS)*, pp. 52–60, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/W18-2509. URL https://aclanthology.org/W18-2509.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014. URL http://jmlr.org/papers/v15/srivastava14a.html.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. In *ICLR*, 2018.

Jesse Vig, Sebastian Gehrmann, Yonatan Belinkov, Sharon Qian, Daniel Nevo, Yaron Singer, and Stuart Shieber. Investigating gender bias in language models using causal mediation analysis. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 12388–12401. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/92650b2e92217715fe312e6fa7b90d82-Paper.pdf.

Kevin Ro Wang, Alexandre Variengien, Arthur Conmy, Buck Shlegeris, and Jacob Steinhardt. Interpretability in the wild: a circuit for indirect object identification in GPT-2 small. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/pdf?id=NpsVSN6o4ul.

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. Emergent abilities of large language models, 2022.

Zhilin Yang, William W. Cohen, and Ruslan Salakhutdinov. Revisiting semi-supervised learning with graph embeddings. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pp. 40–48. JMLR.org, 2016.

Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. Do transformers really perform badly for graph representation? In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/pdf?id=OeWooOxFwDa.

Rex Ying, Dylan Bourgeois, Jiaxuan You, Marinka Zitnik, and Jure Leskovec. Gnnexplainer: Generating explanations for graph neural networks, 2019.

Hao Yuan, Jiliang Tang, Xia Hu, and Shuiwang Ji. Xgnn: Towards model-level explanations of graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '20. ACM, August 2020. doi: 10.1145/3394486.3403085. URL http://dx.doi.org/10.1145/3394486.3403085.

Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang, and Hyunwoo J. Kim. Graph transformer networks, 2020.

Jiong Zhu, Yujun Yan, Lingxiao Zhao, Mark Heimann, Leman Akoglu, and Danai Koutra. Beyond homophily in graph neural networks: Current limitations and effective designs. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 7793–7804. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/58ae23d878a47004366189884c2f8440-Paper.pdf.

A DATA

Homophily quantifies the likelihood of a connection between the nodes that share the same class.

Node Homophily is the average fraction of a node's neighbors sharing the same class (Pei et al., 2020).

$$h_{\text{node}} = \frac{1}{|V|} \sum_{v \in V} \frac{|\{u \in N(v) : y_u = y_v\}|}{d(v)},$$

where V the set of nodes, $N(v)$ the neighbors of v, y_v the class of v, $d(v)$ the degree (number of neighbors) of v.

Edge Homophily is the fraction of edges connecting nodes of the same class (Zhu et al., 2020; Abu-El-Haija et al., 2019).

$$h_{\text{edge}} = \frac{|\{\{u, v\} \in E : y_u = y_v\}|}{|E|},$$

where E is the set of edges. When the classes are imbalanced, this can be misleading.

Adjusted Homophily corrects for the class imbalances (Newman, 2003; Platonov et al., 2023).

$$h_{\text{adj}} = h_{\text{edge}} - \frac{\sum_{k=1}^{K} \bar{p}(k)^2}{1 - \sum_{k=1}^{K} \bar{p}(k)^2},$$

where $\bar{p}(k)$ is the fraction of nodes in class k, and K is the total number of classes. By subtracting the expected homophily $\left(\sum_{k=1}^{K} \bar{p}(k)^2\right)$ from edge homophily h_{edge} and normalizing, this formula calculates how much the observed homophily exceeds or falls short of the expected homophily based on class proportions. This operation aims to *adjust* edge homophily in the presence of class imbalance.

B ATTENTION CACLUCATION

In Transformers, each node in the graph, represented by a row of H^ℓ at layer ℓ, has two separate roles in the attention calculation (Vaswani et al., 2017; Rush, 2018). Firstly, H_i^ℓ acts as a query when it is the current focus of attention, i.e. when it is attending to other nodes. Secondly, it acts as a *key* when it is being compared to the current focus of attention. Following the previous implementations, our models learn two independent linear maps, $W^{K(l)}$ and $W^{Q(l)}$ to compute $Q^{(l)} = H^{(l)}W^{Q(l)}$ and $K^{(l)} = H^{(l)}W^{K(l)}$ corresponding to these two roles in layer l.

When calculating how much to attend to another node, the mechanism compares the node that is the current focus of attention (query) with the rest of the nodes in the graph (keys) to compute a relevance score that determines how much information the query node is going to borrow from the key node.[7] Then, how much node i attends to node j is calculated as a function of the dot product between the i^{th} row of $Q^{(l)}$ and the j^{th} row of $K^{(l)}$:

$$\mathbb{A}_i = \text{softmax}\left(\frac{Q_i^{(l)} {K^{(l)}}^T}{\sqrt{d_h}}\right),$$

where d_h is the number of columns in $Q^{(l)}$.[8]

[7]For instance, in GAT (Veličković et al., 2018), the key and query matrices are constrained to be the same and the attention is limited only to the neighbors of the current focus of attention.

[8]The square root of d_h is used to normalize attention vector to avoid numerical issues due to the exponentiation of large numbers in the softmax that follows.

C PLOTS FOR ATTENTION ACROSS HEADS AND LAYERS

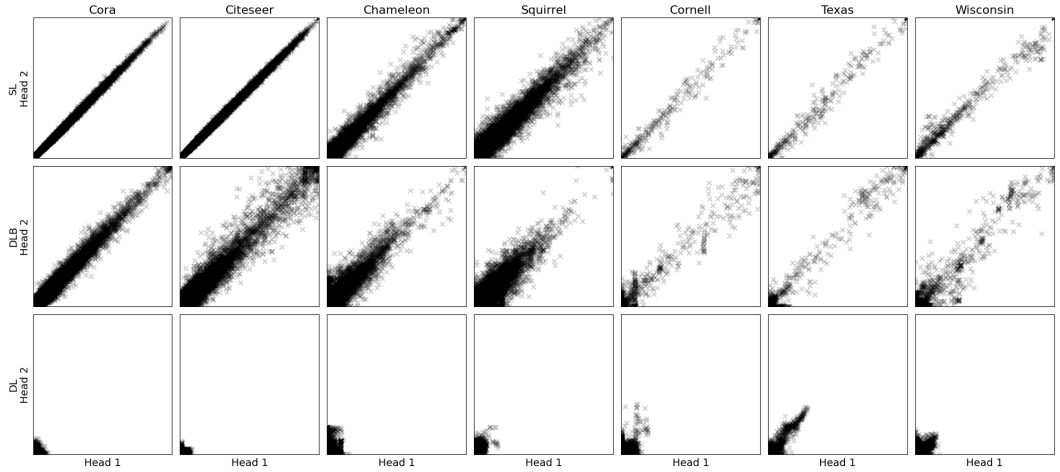

Figure 4: Comparison of the attention values learned by different heads in the models with 1-layer 2-head transformer models: SL (row 1), DLB (row 2), and DL (row 3). We couple the attention values as points $(x, y) = (\mathbb{A}_{ij}^{H1}, \mathbb{A}_{ij}^{H2})$ and place them as points in the coordinate plane. Hence, the values that lie on $x = y$ indicate that the attention patterns match exactly. Any deviation from the $x = y$ line indicates the heads are learning different attention values for the corresponding entry.

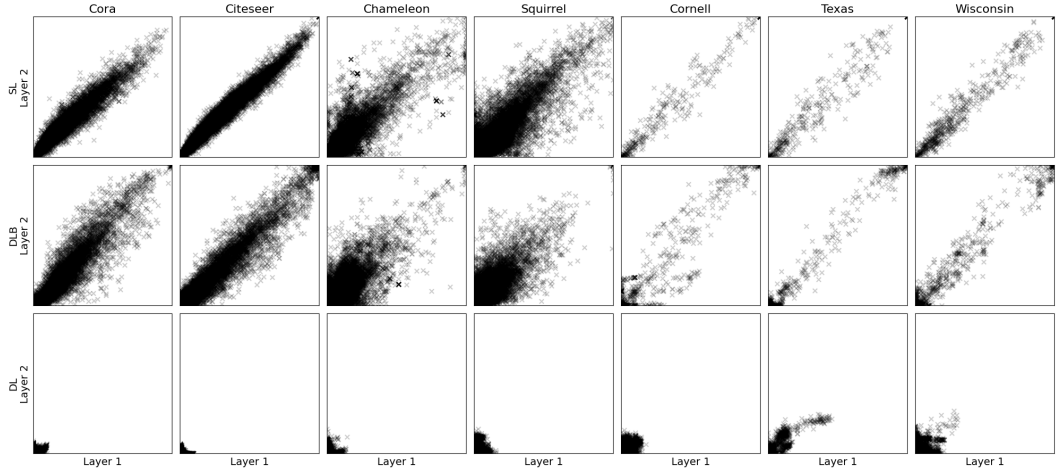

Figure 5: Comparison of the attention values learned by different layers in the models with 2-layer 1-head architectures: SL (row 1), DLB (row 2), and DL (row 3). We couple the attention values as points $(x, y) = (\mathbb{A}_{ij}^{H1}, \mathbb{A}_{ij}^{H2})$ and place them as points in the coordinate plane. Hence, the values that lie on $x = y$ indicate that the attention patterns match exactly. Any deviation from the $x = y$ line indicates the heads are learning different attention values for the corresponding entry.

D MODEL PERFORMANCE

Model	Cora	Citeseer	Chameleon	Squirrel	Cornell	Texas	Wisconsin
1L1H							
SC	**0.85** (0.01)	**0.75** (0.02)	0.45 (0.02)	0.31 (0.01)	0.61 (0.05)	0.71 (0.06)	0.76 (0.05)
SL	**0.85** (0.01)	0.74 (0.02)	**0.57** (0.02)	**0.40** (0.01)	0.64 (0.03)	**0.78** (0.06)	0.79 (0.05)
DLB	**0.85** (0.01)	**0.75** (0.01)	**0.57** (0.02)	**0.40** (0.02)	0.65 (0.05)	0.77 (0.07)	0.80 (0.05)
DL	0.69 (0.02)	0.69 (0.02)	0.49 (0.02)	0.35 (0.01)	**0.69** (0.03)	0.75 (0.10)	**0.81** (0.04)
1L2H							
SL	**0.85** (0.01)	**0.74** (0.01)	**0.58** (0.02)	0.40 (0.01)	0.66 (0.07)	**0.78** (0.08)	0.78 (0.03)
DLB	0.84 (0.01)	**0.74** (0.02)	**0.58** (0.02)	**0.41** (0.01)	0.67 (0.05)	**0.78** (0.07)	0.78 (0.03)
DL	0.69 (0.02)	0.69 (0.01)	0.50 (0.02)	0.36 (0.01)	**0.71** (0.06)	**0.78** (0.04)	**0.80** (0.03)
2L1H							
SC	0.86 (0.01)	**0.75** (0.01)	0.43 (0.03)	0.32 (0.02)	0.60 (0.06)	0.66 (0.07)	0.71 (0.03)
SL	**0.87** (0.01)	0.74 (0.01)	0.60 (0.02)	**0.43** (0.01)	0.64 (0.05)	0.77 (0.07)	0.77 (0.03)
DLB	**0.87** (0.01)	**0.75** (0.01)	**0.61** (0.01)	**0.43** (0.02)	0.63 (0.09)	0.73 (0.08)	0.76 (0.05)
DL	0.69 (0.02)	0.69 (0.01)	0.50 (0.02)	0.36 (0.01)	**0.72** (0.06)	**0.79** (0.07)	**0.80** (0.03)
2L2H							
SL	**0.87** (0.01)	**0.74** (0.01)	0.61 (0.02)	**0.44** (0.01)	0.61 (0.07)	**0.77** (0.07)	0.75 (0.02)
DLB	0.86 (0.01)	**0.74** (0.01)	**0.63** (0.02)	**0.44** (0.01)	0.62 (0.09)	0.75 (0.07)	0.78 (0.04)
DL	0.69 (0.02)	0.69 (0.01)	0.50 (0.02)	0.36 (0.01)	**0.70** (0.06)	**0.77** (0.08)	**0.81** (0.03)

Table 3: Accuracies of SC, SL, DLB, and DL. We train our models on 10 random splits of the datasets. We use 40% of the nodes for training, 30% for validation, and 30% for test. Reported results are averaged across 10 runs and standard deviations are reported in brackets. For SC with constant attention, we only train 1 head models since the model does not have the flexibility to learn different attention patterns in two heads.

E ATTENTION PATTERNS

	Model	Cora	Citeseer	Chameleon	Squirrel	Cornell	Texas	Wisconsin
1L1H	SL	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	DLB	0.0	0.0	0.01	0.0	0.01	0.0	0.0
	DT	1.13	0.68	1.03	1.16	1.28	1.21	0.86
1L2H	SL	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	DLB	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	DL	0.79	0.55	0.97	1.14	0.86	1.16	0.91
2L1H	SL	0.00	0.00	0.00	0.01	0.00	0.00	0.00
	DLB	0.00	0.00	0.02	0.02	0.01	0.01	0.01
	DL	0.96	0.77	0.94	1.25	1.00	0.75	0.98
2L2H	SL	0.00	0.00	0.00	0.01	0.00	0.00	0.00
	DLB	0.00	0.00	0.02	0.03	0.01	0.01	0.01
	DL	0.89	0.84	1.14	1.12	0.91	0.77	0.88

Table 4: Ratios of the average attention paid to non-neighbours and average attention paid to neighbours in SL, DLB, and DL models. If a node is attending to itself, we classify that as inside the neighborhood. Values close to 0 indicate that the attention is focused on the neighborhood of the nodes. Values close to 1 indicate the attention is on average equally distributed between neighbors and non-neighbors. Values greater than 1 indicate that on average attention is focused on non-neighbors.

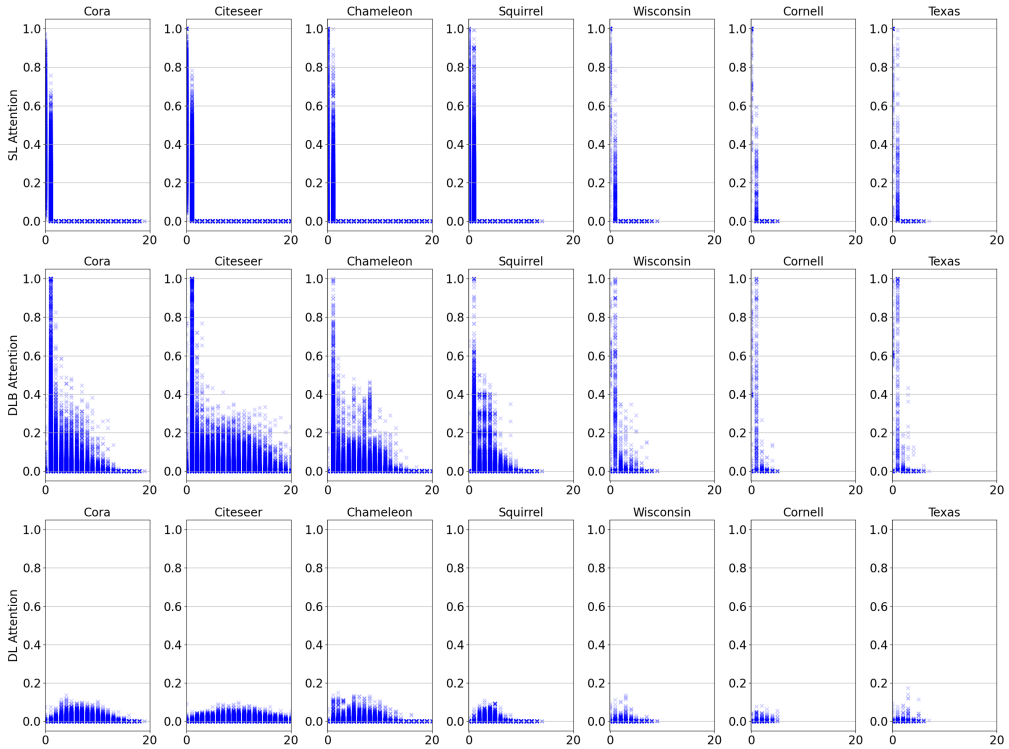

Figure 6: Distributions of the attention values across the n-hop neighborhoods for 1-layer 1-head models: SL (row 1), DLB (row 2), and DL (row 3). Each marker $\times$ shows the attention a node pays (y-axis) to a neighbor in its x-hop neighborhood (x-axis). For a particular node i, if i is paying 0.5 attention to j, and j is in the 3-hop neighborhood of i, that is demonstrated with the point $(x, y) = (3, 0.5)$. If a node is attending to itself, we consider that to be the 0-hop neighborhood.

F PSEUDO-ADJACENCY MATRIX

To check whether we can recover the original graph structure encoded in the data adjacency matrix from the aggregate attention matrix, we threshold the values in the attention matrix to construct a "pseudo-adjacency" by setting all the values that are greater than the threshold to 1 and everything else to 0. Thresholding attention has been previously suggested by Knyazev et al. (2019, Section 2.1). In our experiments, we select the threshold such that the number of recovered edges in the pseudo-adjacency matrix is similar to the total number of edges in the adjacency matrix. This allows us to roughly balance precision and recall by avoiding too sparse or too dense pseudo-adjacency matrices, preventing the inflation of false positives and false negatives. To find the threshold, we grid search all threshold values between 0 and 1 with 0.001 increments.