

## A PROOFS

### A.1 REINTERPRETATION OF ATTENTION BASED GNNs AS GRAPH CONVOLUTION MODELS

For our analysis, we assume the attention vector  $\mathbf{a}$  defined in (Veličković et al., 2018) is symmetric:  $\mathbf{a}[1 : N] = \mathbf{a}[N + 1 : 2N]$ . This results in a symmetric attention coefficient matrix and the analysis is simpler. Also note that the symmetric attention functions are used in practice as well (Thekumparampil et al., 2018). Our first result is on the equivalence of GAT and graph convolutions. As stated in equation 1, graph convolution with kernel  $\mathbf{g}$  can be defined using the eigenvectors and eigenvalues of the symmetric normalized graph Laplacian matrix  $\mathbf{L}_{\text{norm}}$ . Similarly we can define a convolution operation using an alternative of the normalized Laplacian, known as the random walk normalized Laplacian matrix, which is defined as:

$$\mathbf{L}_{\text{rw}} = \mathbf{D}^{-1}\mathbf{L} = \mathbf{I} - \mathbf{D}^{-1}\mathbf{A} = \mathbf{D}^{-1/2}\mathbf{L}_{\text{norm}}\mathbf{D}^{1/2}.$$

The convolution operation with  $\mathbf{L}_{\text{rw}}$  can then be defined as in equation 11

$$\mathbf{g} \hat{\star} \mathbf{x} = \mathbf{D}^{-1/2} \mathbf{U} \mathbf{g}(\Lambda) \mathbf{U}^T \mathbf{D}^{-1/2} \mathbf{x}. \quad (11)$$

**Proposition 1** *Each layer in the GAT model defines a new, **layer-dependent graph adjacency matrix**  $\Gamma^{(l)}$  and a corresponding degree matrix  $\Gamma_{\mathbf{D}}^{(l)}$ . Each layer then computes a first order approximation of the convolution operator  $\hat{\star}$  defined in equation 11 using  $\Gamma^{(l)}$  and  $\Gamma_{\mathbf{D}}^{(l)}$ :*

$$\mathbf{H}^{(l+1)} = \sigma(\Gamma_{\mathbf{D}}^{(l)-1} \Gamma^{(l)} \mathbf{H}^{(l)} \mathbf{W}^{(l)}) \approx \sigma(\mathbf{g}_{\Gamma^{(l)}, \Gamma_{\mathbf{D}}^{(l)}}^{(l)} \hat{\star}(\mathbf{H}^{(l)}))$$

where  $\sigma$  is the non-linearity and  $\mathbf{g}_{\Gamma^{(l)}, \Gamma_{\mathbf{D}}^{(l)}}^{(l)}$  is the convolutional kernel characterized by  $\Gamma^{(l)}, \Gamma_{\mathbf{D}}^{(l)}$ .

Proposition 1 establishes a direct equivalence between the graph convolution and the GAT model. This interpretation also shows that the GAT model applies Laplacian smoothing to node based features (Taubin, 1995; Li et al., 2018). Such a connection between spectral operations and attention based graph neural networks provides directions for theoretical analysis of attention GNNs.

### A.2 GAT MODEL IS EQUIVALENT TO LAYER-WISE CONVOLUTION

Consider a single layer of the GAT model. Let  $\mathbf{H} \in \mathbb{R}^{N \times D}$  be the input feature matrix to a single GAT layer, let  $\mathbf{W} \in \mathbb{R}^{D \times F}$  denote the weight matrix, let  $\mathbf{C} \in \mathbb{R}^{D \times 2}$  be such that  $\mathbf{C} = [\mathbf{a}(1 : N) \mathbf{a}(N + 1 : 2N)]$ , where  $\mathbf{a} \in \mathbb{R}^{2N}$  denotes the attention coefficient vector as defines in (Veličković et al., 2018). Let  $\mathbf{A} \in \mathbb{R}^{N \times N}$  be the graph adjacency matrix and let  $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}_N$ . For a given graph, if  $\mathbf{D}$  represents the degree matrix, then  $\mathbf{D}^{-1}\mathbf{A}$  is simply the state transition matrix of a random walker on the graph.

We can further define a matrix  $\mathbf{Q} \in \mathbb{R}^{N \times 2}$  as

$$\mathbf{Q} = \mathbf{H} \mathbf{W} \mathbf{C}. \quad (12)$$

Further, let us define  $e_{ij}$  (as in (Veličković et al., 2018)) as

$$e_{ij} = \mathbf{a}^T [\mathbf{W} \mathbf{h}_i || \mathbf{W} \mathbf{h}_j] \quad (13)$$

where  $\mathbf{h}_i$  is the  $i$ th row of  $\mathbf{H}$ . Then, we can see that the vector  $\mathbf{e}_i$  where  $e_i(j) = e_{ij}$  is given as

$$\mathbf{e}_i = [\tilde{\mathbf{A}}(:, i) \quad \text{diag}(\tilde{\mathbf{A}}(:, i))] \begin{bmatrix} \mathbf{Q}(i, :) & \mathbf{0}_{1 \times 2} \\ \mathbf{0}_{N \times 2} & \mathbf{Q} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}. \quad (14)$$

Hence, we can express the matrix of attention coefficients, before the softmax operation,  $\Gamma$  as

$$\Gamma = f(\hat{Q}^\top \hat{A}) \quad (15)$$

where

$$\hat{Q} = \begin{bmatrix} Q(1,:) & & & & \\ & Q(2,:) & & & \\ & & \ddots & & \\ & & & Q(N,:) & \\ & & & & Q \end{bmatrix}, \quad \hat{A} = \begin{bmatrix} 1 & 0 & \cdots \\ 0 & 0 & \cdots \\ 0 & 1 & \cdots \\ \mathbf{0}_{N-1} & \mathbf{0}_{N-1} & \cdots \\ 0 & 0 & \cdots \\ 1 & 0 & \cdots \\ 0 & 0 & \cdots \\ 0 & 1 & \cdots \\ \vdots & \vdots & \cdots \end{bmatrix}, \quad \tilde{A} = \begin{bmatrix} \tilde{A} \\ \text{diag}(\tilde{A}(:, 1)) \\ \vdots \\ \text{diag}(\tilde{A}(:, N)) \end{bmatrix} \quad (16)$$

and  $f = \exp(\text{LeakyReLU}(\cdot))$ . Further, let  $\Gamma_D = \text{diag}(\Gamma \mathbf{1}_N)$ .

The GAT layer update can be expressed as,

$$\hat{H} = \sigma \left( \text{diag} \left( f(\hat{Q}^\top \hat{A}) \mathbf{1}_N \right)^{-1} f(\hat{Q}^\top \hat{A}) H W \right) \quad (17)$$

$$\hat{H} = \sigma \left( \Gamma_D^{-1} \Gamma H W \right) \quad (18)$$

Given a new graph  $G_\Gamma(\mathcal{E}, \mathcal{V})$  with  $\Gamma$  being the adjacency matrix. Let  $\Gamma_D$  be the corresponding degree matrix. Then, the random walk normalized Laplacian is defined as

$$L_{\text{rw}} = I - \Gamma_D^{-1} \Gamma \quad (19)$$

Note that although  $L_{\text{rw}}$  is asymmetric, it is similar to the symmetric normalized Laplacian matrix:

$$L_{\text{norm}} = I - \Gamma_D^{-1/2} \Gamma \Gamma_D^{-1/2} = \Gamma_D^{1/2} L_{\text{rw}} \Gamma_D^{-1/2}. \quad (20)$$

Hence,  $L_{\text{rw}}$  has real eigenvalues and match with those of  $L_{\text{norm}}$ . The corresponding eigenvectors of the two matrices are also related: If  $v$  is an eigenvector of  $L_{\text{norm}}$ , then  $\Gamma_D^{-1/2} v$  is an eigenvector of  $L_{\text{rw}}$ . Using this, we can define a new convolution operator as

$$g_\theta \hat{\star} h = \Gamma_D^{-1/2} U_{\text{sym}} g_\theta(\Lambda_{\text{sym}}) U_{\text{sym}}^T \Gamma_D^{1/2} h \quad (21)$$

Then, using the Chebychev polynomial approximation similar to (Kipf & Welling 2016), we can show that for a given feature vector  $h$ , we can get a first order approximation to the operation  $g_\theta \hat{\star} h$  as

$$g_\theta \hat{\star} h \approx \Gamma_D^{-1} \Gamma h. \quad (22)$$

For multiple output features, the new graph convolution operation has a first order approximation as

$$g_\theta \hat{\star} H \approx \Gamma_D^{-1} \Gamma H W \quad (23)$$

which matches exactly with equation 18. This shows that the model defined in (Veličković et al. 2018) is similar to a GCN model, but defined layer-wise.

### A.3 SPECTRAL SPARSIFICATION PRESERVES GRAPHS CONVOLUTIONAL FEATURES

In this section, we show that the features learnt by graph convolution based neural networks are preserved when spectral sparsification techniques are applied to the original data graph. We use the following notation:  $L_{\text{norm}}$  is as defined in equation 20 and denotes the symmetric normalized Laplacian matrix of a graph,  $L_{\text{norm},s}$  denotes the symmetric normalized Laplacian matrix of the corresponding spectrally sparsified graph with a parameter of  $\epsilon$ . Similarly, we use  $L_{\text{rw}}$  to denote the random walk normalized Laplacian matrix of a graph and  $L_{\text{rw},s}$  to denote the corresponding random walk normalized Laplacian matrix of the spectrally sparsified graph.

**Spectral sparsification and the GCN model** Consider the graph convolution network architecture proposed in (Kipf & Welling 2016). We assume that the non-linearity  $\sigma(\cdot)$  used in Lipshitz continuous

with a Lipschitz constant  $\ell_\sigma$ . For a single neural network layer, let the input features be  $\mathbf{H} \in \mathbb{R}^{N \times D}$ , let the weight matrix be  $\mathbf{W} \in \mathbb{R}^{D \times F}$ , where  $F$  is the number of output features. Then, the new set of features computed by the GCN model is

$$\widehat{\mathbf{H}} = \sigma((\mathbf{L}_{\text{norm}} - \mathbf{I})\mathbf{H}\mathbf{W}) \quad (24)$$

and the corresponding set of features computed by the GCN model using a spectrally sparsified graph are given as

$$\widehat{\mathbf{H}}' = \sigma((\mathbf{L}_{\text{norm},s} - \mathbf{I})\mathbf{H}\mathbf{W}) \quad (25)$$

### Proof of Theorem 1

**Proof** We first characterize the spectral norm error between the corresponding graph Laplacians  $\mathbf{L}_{\text{norm}}$  and  $\mathbf{L}_{\text{norm},s}$  and then use the bound to prove Theorem 1. We use  $\mathbf{D}$  and  $\mathbf{D}_s$  to denote the degree matrices of the full and the spectrally sparsified graphs.

Since both  $\mathbf{L}_{\text{norm}}$  and  $\mathbf{L}_{\text{norm},s}$  are symmetric and positive semidefinite, we have,

$$\begin{aligned} \|\mathbf{L}_{\text{norm}} - \mathbf{L}_{\text{norm},s}\| &= \max(|\lambda_{\max}(\mathbf{L}_{\text{norm}} - \mathbf{L}_{\text{norm},s})|, |\lambda_{\min}(\mathbf{L}_{\text{norm}} - \mathbf{L}_{\text{norm},s})|) \\ &= \sup_{\mathbf{x} \in \mathbb{R}^N: \|\mathbf{x}\|=1} |\mathbf{x}^\top (\mathbf{L}_{\text{norm}} - \mathbf{L}_{\text{norm},s}) \mathbf{x}| \end{aligned}$$

We then have

$$|\mathbf{x}^\top (\mathbf{L}_{\text{norm}} - \mathbf{L}_{\text{norm},s}) \mathbf{x}| = |\mathbf{x}^\top \mathbf{L}_{\text{norm}} \mathbf{x} - \mathbf{x}^\top \mathbf{D}^{-1/2} \mathbf{D}_s^{1/2} \mathbf{L}_{\text{norm},s} \mathbf{D}_s^{1/2} \mathbf{D}^{-1/2} \mathbf{x}| \quad (26)$$

$$\begin{aligned} &+ |\mathbf{x}^\top \mathbf{D}^{-1/2} \mathbf{D}_s^{1/2} \mathbf{L}_{\text{norm},s} \mathbf{D}_s^{1/2} \mathbf{D}^{-1/2} \mathbf{x} - \mathbf{x}^\top \mathbf{L}_{\text{norm},s} \mathbf{x}| \\ &\leq |\mathbf{x}^\top \mathbf{L}_{\text{norm}} \mathbf{x} - \mathbf{x}^\top \mathbf{D}^{-1/2} \mathbf{D}_s^{1/2} \mathbf{L}_{\text{norm},s} \mathbf{D}_s^{1/2} \mathbf{D}^{-1/2} \mathbf{x}| \\ &+ |\mathbf{x}^\top \mathbf{D}^{-1/2} \mathbf{D}_s^{-1/2} \mathbf{L}_{\text{norm},s} \mathbf{D}^{-1/2} \mathbf{D}_s^{1/2} \mathbf{x} - \mathbf{x}^\top \mathbf{L}_{\text{norm},s} \mathbf{x}| \end{aligned} \quad (27)$$

Taking supremum on both sides, we get

$$\begin{aligned} \sup_{\mathbf{x} \in \mathbb{R}^N: \|\mathbf{x}\|=1} |\mathbf{x}^\top (\mathbf{L}_{\text{norm}} - \mathbf{L}_{\text{norm},s}) \mathbf{x}| &\leq \epsilon \sup_{\mathbf{x} \in \mathbb{R}^N: \|\mathbf{x}\|=1} |\mathbf{x}^\top \mathbf{L}_{\text{norm}} \mathbf{x}| + \\ &\left\| \mathbf{D}^{-1/2} \mathbf{D}_s^{-1/2} \mathbf{L}_{\text{norm},s} \mathbf{D}^{-1/2} \mathbf{D}_s^{1/2} - \mathbf{L}_{\text{norm},s} \right\| \\ &\leq \epsilon \|\mathbf{L}_{\text{norm}}\| + \end{aligned} \quad (28)$$

$$\begin{aligned} &\left\| \mathbf{D}^{-1/2} \mathbf{D}_s^{-1/2} \mathbf{L}_{\text{norm},s} \mathbf{D}^{-1/2} \mathbf{D}_s^{1/2} - \mathbf{L}_{\text{norm},s} \right\| \\ &\leq \epsilon \|\mathbf{L}_{\text{norm}}\| + 3\epsilon \|\mathbf{L}_{\text{norm}}\| \\ &\leq 4\epsilon \|\mathbf{L}_{\text{norm}}\| \end{aligned} \quad (29)$$

$$\leq 8\epsilon \quad (30)$$

where we assume that  $3\epsilon^2 + \epsilon^3 < \epsilon$ , which holds true for small  $\epsilon$  and we also have  $\|\mathbf{L}_{\text{norm}}\| \leq 2$ .

We then have the final result as below. Let  $\ell_\sigma$  be the Lipschitz constant of the non-linearity  $\sigma$ .

$$\left\| \widehat{\mathbf{H}} - \widehat{\mathbf{H}}' \right\|_F \leq \ell_\sigma 4\epsilon \|(\mathbf{L}_{\text{norm}} - \mathbf{L}_{\text{norm},s})\mathbf{H}\mathbf{W}\|_F \quad (31)$$

$$4\epsilon \|\mathbf{L}_{\text{norm}} - \mathbf{L}_{\text{norm},s}\| \|\mathbf{H}\mathbf{W}\|_F \quad (32)$$

where, we use  $\ell_\sigma = 1$  for ReLU or ELU non-linearity, and use inequality

$$\|\mathbf{AB}\|_F \leq \|\mathbf{A}\| \|\mathbf{B}\|_F$$

for any two matrices  $\mathbf{A}$  and  $\mathbf{B}$ . ■

Theorem 1 shows that if two GCN models that use the full and spectrally sparsified graphs have the same initialization  $\mathbf{W}$ , then the corresponding feature updates are close in a Frobenius norm sense.

Although we have not explored the dynamics or training, we strongly believe that similar bounds can be obtained on the gradients of the network parameters and in turn on the gradient descent updates.

**Spectral sparsification and the GAT model** We can now consider the graph attention network model proposed in (Veličković et al., 2018). With  $H$  and  $W$  as defined in the previous section, the feature update equations for the GAT model using the full and the spectrally sparsified graphs are given as

$$\begin{aligned}\widehat{H} &= \sigma((L_{\text{rw}} - I)HW) \\ \widehat{H}' &= \sigma((L_{\text{rw},s} - I)HW)\end{aligned}\tag{33}$$

Due to Equations equation 20 we can rewrite the above equations as

$$\widehat{H} = \sigma(D^{-1/2}(L_{\text{norm}} - I)D^{1/2}HW)\tag{34}$$

$$\widehat{H}' = \sigma(D_s^{-1/2}(L_{\text{norm},s} - I)D_s^{-1/2}HW)\tag{35}$$

As before, we first bound the error  $\|L_{\text{rw}} - L_{\text{rw},s}\|$  and then use it to bound the error  $\widehat{H} - \widehat{H}'$ .

#### Proof of Theorem 2

Theorem 2 shows that if a layer-wise spectral sparsification of the graph is used to reduce the number of edges, then the feature updates computed by the sparse model are also preserved. Note that this requires sparsifying the graph in each layer separately with the weights in the adjacency matrix given by  $\Gamma$ . In the next section, we show that this expensive procedure of layer-wise sparsification can be repalced by a one-time spectral sparsification procedure for the binary node classification problem.

We use the following Lemma to establish Theorem 2

**Lemma 1** Let  $A \in \mathbb{R}^{N \times N}$  be any matrix and  $D \in \mathbb{R}^{N \times N}$  be a diagonal matrix with positive diagonal entries. Then, we have

$$\|A - D^{-1}AD\| \leq \|A\| (\|I - D^{-1}\| + \|D^{-1}\| \|I - D\|)\tag{36}$$

**Proof** We have

$$\begin{aligned}\|A - D^{-1}AD\| &= \|A - D^{-1}A + D^{-1}A - D^{-1}AD\| \\ &= \|A(I - D^{-1}) + D^{-1}A(I - D)\| \\ &\leq \|A\| \|I - D^{-1}\| + \|I - D\| \|D^{-1}\| \|A\| \\ &\leq \|A\| (\|I - D^{-1}\| + \|D^{-1}\| \|I - D\|)\end{aligned}\tag{37}$$

■

#### Proof of Theorem 2:

$$\begin{aligned}\|L_{\text{rw}} - L_{\text{rw},s}\| &= \|D^{-1/2}L_{\text{norm}}D^{1/2} - D_s^{-1/2}L_{\text{norm},s}D_s^{1/2}\| \\ &= \|D^{-1/2}L_{\text{norm}}D^{1/2} - D^{-1/2}L_{\text{norm},s}D^{1/2} + \\ &\quad D^{-1/2}L_{\text{norm},s}D^{1/2} - D_s^{-1/2}L_{\text{norm},s}D_s^{1/2}\| \\ &\leq \|L_{\text{norm}} - L_{\text{norm},s}\| + \|L_{\text{norm},s} - D^{1/2}D_s^{-1/2}L_{\text{norm},s}D_s^{1/2}D^{-1/2}\|\end{aligned}$$

Then, using Lemma 1 we get

$$\begin{aligned}\|L_{\text{rw}} - L_{\text{rw},s}\| &\leq \|L_{\text{norm}} - L_{\text{norm},s}\| + \epsilon(1 + \epsilon) \|L_{\text{norm},s}\| \\ &\leq (5\epsilon + \epsilon^2) \|L_{\text{norm}}\| \\ &\leq 6\epsilon \|L_{\text{norm}}\| \\ &\leq 12\epsilon\end{aligned}$$

Further, from the feature update equations equation 33 and since  $\sigma$  is Lipschitz continuous with Lipschitz constant  $\ell_\sigma$ , we have the final result as in the proof of Theorem 1

#### A.4 APPROXIMATION OF WEIGHT MATRICES

Theorems 1 and 2 provide an upper bound on the feature updates obtained using the full and sparsified graphs under both GCN and GAT. A stronger notion of information preservation after sparsification is obtained by studying the weight matrices to see if the graph structure is retained after the sparsification. To this end, we would like to study the error  $\|\mathbf{W} - \mathbf{W}_s\|$ , where  $\mathbf{W}$  and  $\mathbf{W}_s$  are weight matrices of the neural network, learned using the full and the sparsified graph respectively in any given layer. Such a result shows whether the graph structure retained is sufficient for the GAT model to learn strong features.

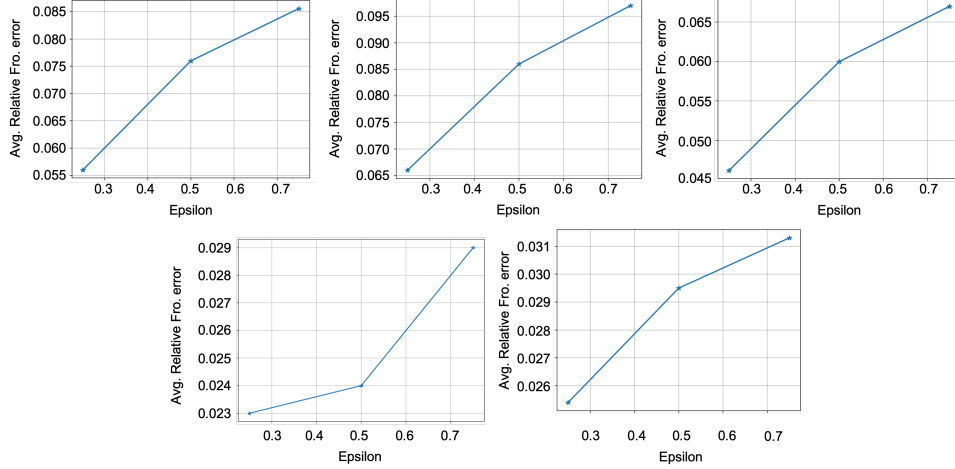


Figure 2: Relative Error between the learned weights without and with sparsification on Coauthor-Phy, Github Social and Coauthor-CS (top) and Amazon-Computer and Amazon-Photos (bottom) datasets. We can see that the error is proportional to the  $\epsilon$  parameter. Such a comparison was not possible on the Reddit dataset, since the model cannot be run on the full graph.

To lend support to this claim, we studied the difference between the weight matrices learned with and without spectral sparsification. We used three different datasets (Coauthor-Phy, Github Social and Coauthor-CS). In each case, we used three different values of  $\epsilon$  (0.25, 0.5, 0.75). At each parameter setting, we performed 5 independent trials and averaged the relative Frobenius errors between the weight matrices and the attention function  $a$  of all attention heads. We report the results in Fig. 2. It is clear that the error between the learned matrices is proportional to the value of  $\epsilon$  itself. This shows that the training process is highly stable with respect to spectral sparsification of the input graph.

## B DETAILS OF EXPERIMENTS

### B.1 DESCRIPTION OF DATASETS

**Transductive learning tasks:** Amazon Computers and Amazon Photo are segments of the Amazon co-purchase graph (McAuley et al., 2015), where nodes represent goods, edges indicate that two goods are frequently bought together, node features are bag-of-words encoded product reviews, and class labels are given by the product category. Coauthor CS and Coauthor Physics are co-authorship graphs based on the Microsoft Academic Graph from the KDD Cup 2016 challenge 3. Here, nodes are authors, that are connected by an edge if they co-authored a paper; node features represent paper keywords for each author’s papers, and class labels indicate most active fields of study for each author. For the Reddit dataset, we predict which community different Reddit posts belong to based on the interactions between the posts. The Github social dataset consists of Github users as nodes and the task is that of classifying the users as web or machine learning developers (binary classification). For all the above datasets, the task is that of node classification. Additionally, we have experiments on citation graphs: Cora, Citeseer and Pubmed. In these datasets, the nodes represent authors, edges represent mutual citations and the task is to categorize the authors into their fields of study.

Table 6: Comparison of FastGAT with GAT (Veličković et al. [2018]).

Metric	Method	Cora	Citeseer	PubMed
F1-micro	GAT	0.72 $\pm$ 0.007	0.685 $\pm$ 0.004	0.735 $\pm$ 0.003
	FastGAT-0.5	0.713 $\pm$ 0.003	0.685 $\pm$ 0.007	0.73 $\pm$ 0.004
	FastGAT-0.9	0.63 $\pm$ 0.02	0.65 $\pm$ 0.002	0.722 $\pm$ 0.019
GPU Time (s)	GAT	0.294	0.281	0.686
	FastGAT-0.5	0.22	0.236	0.54
	FastGAT-0.9	0.175	0.196	0.456
CPU Time (s)	GAT	0.51	0.52	2.71
	FastGAT-0.5	0.49	0.51	2.01
	FastGAT-0.9	0.39	0.41	1.42
% Edges redu.	FastGAT-0.5	15.4%	10.7%	20%
	FastGAT-0.9	48.7%	38%	50%

**Inductive learning tasks:** We use the Protein=Protein interaction dataset (Zitnik & Leskovec [2017]) where the graphs correspond to different human tissues. The dataset contains 20 graphs for training, 2 for validation and 2 for testing. Critically, testing graphs remain completely unobserved during training. To construct the graphs, we used the preprocessed data provided by (Hamilton et al. [2017a]). The average number of nodes per graph is 2372. Each node has 50 features that are composed of positional gene sets, motif gene sets and immunological signatures. There are 121 labels for each node set from gene ontology, collected from the Molecular Signatures Database (Subramanian et al., 2005), and a node can possess several labels simultaneously.

**Evaluation setup.** For the Reddit dataset, we use training, validation and test data split of 65%, 10% and 25%, as specified in the DGLGraph library. For the other datasets, the split is 10%, 20% and 70%. The same split is for evaluating the original GAT model. For training and evaluation, we closely follow the setup used in (Veličković et al. [2018]). We first use the spectral sparsification algorithm to obtain a sparse graph and then use a two-layer GAT model for training and inference. The first layer consists of  $K = 8$  attention heads, computing 8 output features each, after which we apply the exponential linear unit (ELU). The second layer consists of a single attention head that computes  $C$  features (where  $C$  is the number of classes), followed by a softmax activation. We use the same architecture while comparing with the GAT and the sparseGAT models. We train all the models using a transductive approach wherein we use the features of all the nodes to learn the node embeddings. For the inductive learning task, we follow the evaluation method used in (Veličković et al. [2018]). We apply a three-layer GAT model. The first two layers consist of  $K = 4$  attention heads computing 256 features (for a total of 1024 features), followed by an ELU nonlinearity. The final layer is used for (multi-label) classification:  $K = 6$  attention heads computing 121 features each, that are averaged and followed by a logistic sigmoid activation.

**Implementation details.** For each dataset, we compute the effective resistances of the edges using the Laplacians library written in Julia by Spielman (Lap). The rest of the algorithm is implemented in PyTorch. We use the code for the GAT provided in (Veličković et al. [2018]). We train our models on an Ubuntu 16.04 with 128GB memory and a Tesla P100 GPU (with 16GB memory). We use Adam optimizer with a learning rate of 0.001. We use the hyperparameters recommended in (Veličković et al. [2018]) for all of our experiments that use the GAT model. For FastGCN, we use the baseline parameters recommended in (Chen et al. [2018]).

**Computing effective resistances.** For all datasets, computing the effective resistances is a one-time pre-processing task. We use the algorithm proposed in (Spielman & Srivastava [2011]), which takes about  $O(M \log N)$  time to compute the effective resistances of all the edges in the graph. We compute the resistance values and store them as metadata. While performing training and inference, we load the resistance values and then sample from the distribution described in Section 3.2.

## B.2 EXPERIMENTAL RESULTS ON SMALLER DATASETS

We report the experimental results on the smaller datasets Cora, Citeseer and Pubmed in Table 6. Since the number of edges are small compared the larger datasets, the adjacency matrices for these graphs are already considerably sparse. Hence, sparsification does not result in a large reduction in

the number of edges. However, the trend is still similar to that was observed on large datasets, since the accuracy performance does not drop, while training and inference time is lower than that for the model using the full graph.

## Q2. FASTGAT HAS THE SAME RATE OF LEARNING AS GAT MODELS

Our next goal is to study if FastGAT needs more epochs to achieve the same level of accuracy as that of using full graphs. Figures 3 4 show consistent per epoch learning rate for multiple datasets. The accuracy achieved while training with sparsified graphs matches well with that obtained using the full graph on all the datasets, showing that spectral sparsification does not affect learning in attention GNNs.

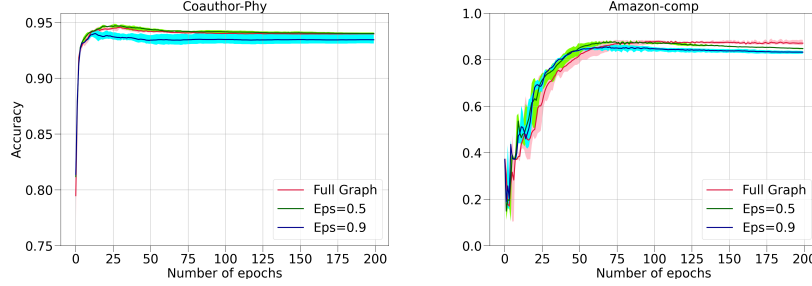


Figure 3: Accuracy Vs number of training epochs for both full and sparsified graphs. The accuracy attained matched almost exactly for every epoch even when a sparsified graph is used. For each dataset, the plots were computed by averaging over 5 independent trials.

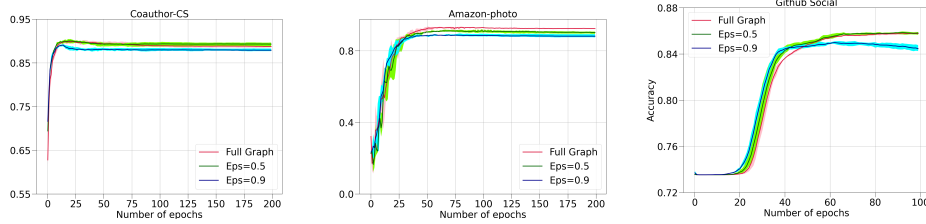


Figure 4: Accuracy Vs number of epochs for full and sparsified graphs, for the Coauthor-CS, Amazon-Photos and Github Social datasets

### B.3 FAST ALGORITHM TO COMPUTE EFFECTIVE RESISTANCES

In this section, we briefly describe the algorithm to quickly compute the effective resistances of a graph  $G(\mathcal{E}, \mathcal{V})$ . We use the algorithm presented in (Spielman & Srivastava 2011) (Section 4) and describe it here for the sake of completion.

For any graph  $G(\mathcal{E}, \mathcal{V})$ , let  $\mathbf{Y} \in \mathbb{R}^{M \times M}$  be such that  $\mathbf{Y}(e, e) = w_e$  and  $\mathbf{B} \in \mathbb{R}^{M \times N}$  be such that

$$\mathbf{B}(e, v) = \begin{cases} 1, & \text{if } v \text{ is } e\text{'s head} \\ -1, & \text{if } v \text{ is } e\text{'s tail.} \\ 0, & \text{otherwise} \end{cases} \quad (38)$$

Then, it can be shown that

$$R_{uv} = \left\| \mathbf{Y}^{1/2} \mathbf{B} \mathbf{L}^\dagger (\chi_u - \chi_v) \right\|^2 \quad (39)$$

Note that the  $R(uv)$ 's are just pair-wise distances between the columns of the  $M \times N$  matrix  $\mathbf{Y}^{1/2} \mathbf{B} \mathbf{L}^\dagger$ . The Johnson-Lindenstrauss Lemma can then be applied to approximately compute these distances. If  $\mathbf{R}$  is a  $t \times M$  random matrix chosen from a suitable distribution such as the Bernoulli distribution or the Gaussian random distribution, then if  $t = O(N/\tau^2)$ , then we have

$$(1-\tau) \left\| \mathbf{Y}^{1/2} \mathbf{B} \mathbf{L}^\dagger (\chi_u - \chi_v) \right\|^2 \leq \left\| \mathbf{R} \mathbf{Y}^{1/2} \mathbf{B} \mathbf{L}^\dagger (\chi_u - \chi_v) \right\|^2 \leq (1+\tau) \left\| \mathbf{Y}^{1/2} \mathbf{B} \mathbf{L}^\dagger (\chi_u - \chi_v) \right\|^2. \quad (40)$$

Finally, the effective resistances are computed by using a fast Laplacian linear system solver (Spielman & Teng [2011]) applied to the rows of the matrix  $\mathbf{R}\mathbf{Y}^{1/2}\mathbf{B}$ . Each application of the fast solver takes  $O(M \log(1/\delta))$  time where  $\delta$  denotes the failure probability and can be set to a constant. The fast solver needs to be applied to  $O(\log N)$  rows of the matrix  $\mathbf{R}\mathbf{Y}^{1/2}\mathbf{B}$ . Hence, the overall complexity of the algorithm is  $O(M \log N)$ .

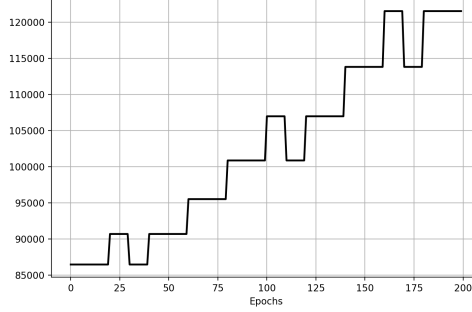


Figure 5: Number of edges selected by the adaptive algorithm for the Coauthor-Phy dataset. The number of edges at a constant epsilon of 0.5 was 163334.

#### B.4 ADAPTIVE SPARSIFICATION ALGORITHM

In the previous sections, we showed that for a suitable value of the tolerance parameter  $\epsilon$  (such as 0.5, 0.9), the accuracy is equivalent to that of using the full graph. However, the level of sparsification needed to maintain the classification performance might be different for different datasets. This raises a very natural question of how to design the  $\epsilon$  parameter for different datasets. In this subsection, we seek to address this question.

We provide here an algorithm that sweeps through various values of  $\epsilon$  and achieves state of the art results on any given dataset. In our experience, we find that using  $\epsilon = 0.5$  produces test accuracies that are as good as that of using the full graph. Hence, we set 0.5 as the minimum value of  $\epsilon$  that our algorithm chooses. It iteratively chooses a denser or a sparser graph based on the current validation error of the algorithm. We provide a block diagram of the algorithm in Figure 6. In Figure 7, we show the training accuracy Vs the epochs for our algorithm and compare it with that of a model using a constant  $\epsilon$  of 0.5. From the figure, it is evident that our adaptive algorithm is successful in achieving the same learning rate as that of a model with constant  $\epsilon$ . Hence this algorithm is suitable to be deployed as is on other real world datasets.

In Figure 5, we show the the number of edges resulting edges in the graph after each instance of the algorithm choosing to sparsify or make the graph more dense. Since denser graphs do offer more

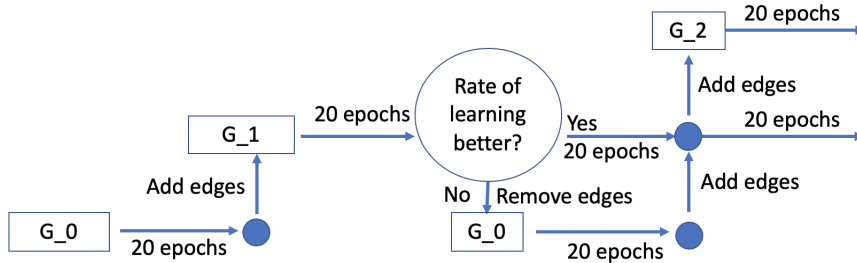


Figure 6: Adaptive algorithm to tune epsilon parameter (or the number of edges). We start with a sparse graph and iteratively build denser graphs as we progress through the epochs. In the “Add edges” step, we add a fixed number ( $0.003M$ ) of edges to the graph. In the “Rate of learning better?” step, we compare the slopes of the training accuracy curve with the previous slope over 20 epochs.



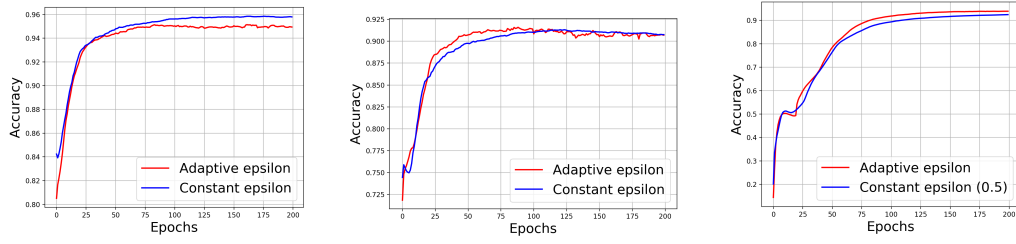


Figure 7: Simulation results for the adaptive algorithm on the coauthor-Physics, coauthor-CS and Reddit datasets.

information, it is natural that the algorithm chooses denser graphs over time in general. But it is also interesting to see that there are instances where the algorithm chooses a sparser graph. We show the accompanying time per epoch as well in Figure. where we can see that it is much smaller than that of using a constant, low  $\epsilon$  parameter.