An Approximate Attention Filter For The Efficient Precomputation Of Scalable Graph Convolutions

SUBMITTED IN PARTIAL FULFILLMENT FOR THE DEGREE OF MASTER OF SCIENCE

Jonathan A. Harris, M.Sc.

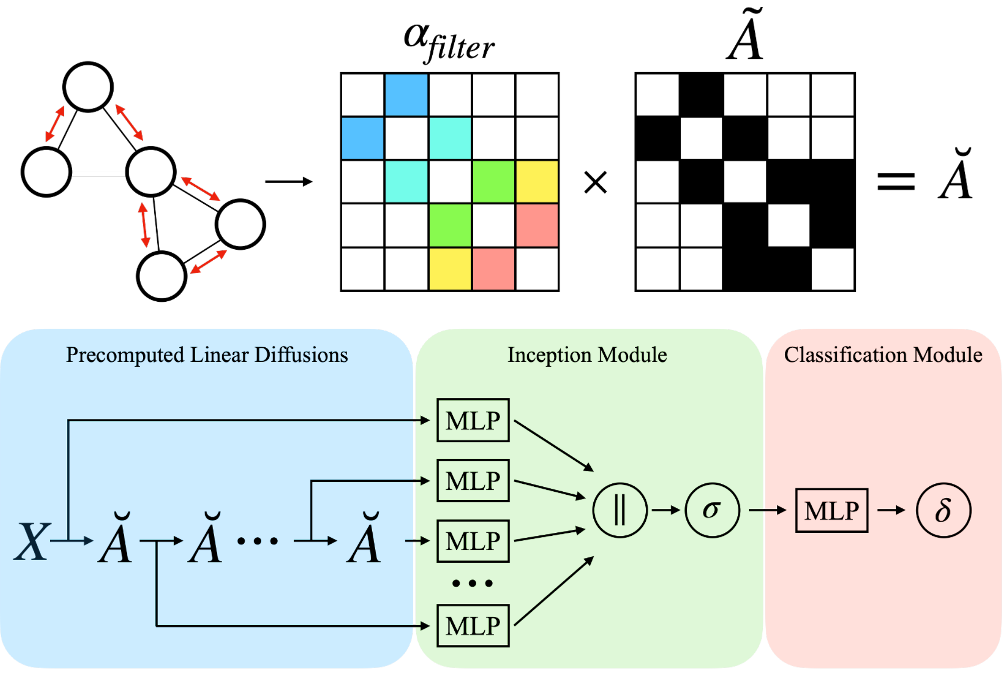
MASTER INFORMATION STUDIES

DATA SCIENCE

FACULTY OF SCIENCE

UNIVERSITY OF AMSTERDAM

Your Date of defence in the format 2022-06-30



Universiteit van Amsterdam

Amsterdam Data Science

SOCIAL DATABASE

An Approximate Attention Filter For The Efficient Precomputation Of Scalable Graph Convolutions

Jonathan A. Harris, M.Sc.

ABSTRACT

Graph neural networks are a popular framework for learning on relational data structures, yet recursive message-passing aggregations preclude their use at scale. Graph-sampling strategies reduce cost and improve speed by selecting subsets of neighbours during aggregation, however doing so introduce bias and statistical dependence between samples. Alternatively, newer full-graph decoupled architectures like Scalable Inception Graph Network (SIGN) improve scalability by *approximating* graph convolutional layers independently from parameter learning. As such, layers are replaced with linear diffusion operations of various filter sizes, which are efficiently pre-aggregated before training. However, diffusion is dependent on topology and does not consider node features, potentially limiting accuracy. The present study proposes enhancing diffusion in SIGN with a novel attention filter, *approximating* learned attention by characterizing feature-similarity between nodes. Experiments assessed the efficacy and efficiency of filters composed of learned graph attention weights, cosine similarity (CS), or a modified scaled dot product attention (DPA) on three benchmark datasets. Our results found that the addition of an approximated attention filter containing CS- or DPA-based weights outperformed the baseline SIGN model across all three datasets; more substantial accuracy gains were observed on the larger datasets. Furthermore, results demonstrate that the approximated filters improve accuracy while maintaining precomputation speeds within the same order of magnitude, without affecting training and inference speeds. Code can be found at: https://github.com/jah377/ApproximatedAttention.

1 INTRODUCTION

Graphs are a type of data structure that encode complex systems of objects and interactions between object pairs as nodes and edges [14]. Relational data structures exist across many domains, including healthcare, social sciences, and information retrieval [6]. The prevalence of graph-structured data and the introduction of Graph Convolutional Networks (GCNs) in 2016 [21] has propelled Graph Representational Learning (GRL) to one of the fasting growing fields in machine learning [6, 37]. GRL, and the broader class of Graph Neural Networks (GNNs), provide a framework to perform node-, edge-, or graph-related inference tasks like fake news detection, recommender systems, and discovering new pharmaceutical drugs based on molecular shape, respectively [6].

The relational structure intrinsic to social networks or e-commerce makes GNNs an attractive tool in industrial settings [37]. However, scalability presents a significant challenge that limits their use outside academia [12]. Several obstacles that inhibit applying GNNs at scale include the lack of web-scale data benchmarks, the inherent structure of data graphs, and the subsequent structure of GNN architectures [12, 16]. Initial GRL research was conducted on small datasets containing tens of thousands of nodes, with little focus on applications at scale due to the dearth of large-scale benchmarking data. In 2020, the introduction of the Open Graph Benchmark (OGB) graphs containing between 132 thousand and 244 million nodes aimed to improve data availability. Nevertheless, web-scale graphs like Facebook contain as many as 2.9 billion nodes [16, 29].

Beyond data availability, the structure of graphs (and therefore the structure of traditional GNN models) can make learning at scale cost-prohibitive [12]. Graphs are non-Euclidean, with irregular sparse connections between nodes. GNNs facilitate learning and downstream prediction tasks by traversing the edges and encoding node representation as low-dimensional vectors summarizing graph position and local topology [14].

In the case of GCNs, the model aggregate features of neighbouring nodes by combining linear diffusion operations and weight-learning at each convolutional layer; L layers are applied in sequence to convolve each node's neighbourhood L-hops away. Computation of loss must consider the diffusion of information along the edges of the graph, which can vary considerably at each node [12]. For example, web-scale graphs like Twitter contain approximately 10^8 nodes, and 10^{10} edges [12], yet on average less than four accounts separate any two random users [27].

Consequently, the number of neighbouring nodes necessary to generate a node embedding grows exponentially with each network layer, highlighting the "neighbourhood explosion" problem innate to GCNs. Moreover, full-batch training of GCNs requires the storage of the entire graph, all hidden embeddings for each layer, and network gradients on graphics processing unit (GPU) memory, limiting the use of GCNs at scale.

To address the aforementioned memory costs, graph sampling methods like GraphSAGE and ClusterGCN use subsampled graphs to approximate local connectivity [10, 13, 17, 36]. Sampling nodes within the larger graph permit mini-batch training of GNNs, during which only a fraction of the data is stored on GPU memory at any given time. Although these methods accelerate performance time and can scale to graphs with billions of edges, sampling may introduce variance/bias into GNN training or result in loss of information [9, 22]. Most significantly, speed gains from sampling are confined to training as downstream prediction tasks require the full graph [22].

Frasca et al. (2020) recently introduced an alternative approach to scalable GNNs that avoids graph sampling altogether [12]. The new architecture, Scalable Inception Graph Neural Network (SIGN), decouples neighbourhood aggregations and parameter learning to minimize computational complexity. SIGN achieves this by *approximating* graph convolutional layers, thus avoiding the expensive compounding of network layers and subsequent non-linear activations. The pre-aggregation of neighbourhood information transforms the input data into matrices, enabling mini-batch training once only possible with graph-sampling GNNs; the matrices feed into an efficient, graph-agnostic Multilayer Perceptron (MLP) network for weight-learning. Once considered state-of-the-art, accuracy improvements of newer models may make SIGN less desirable despite the superior speed performance [16].

Attention mechanisms are used to improve model accuracy and are prevalent in natural language processing, computer vision, and graph learning [30, 31]. The mechanism, as found in Graph attention layers (GAT), improve accuracy by learning trainable attention weights between nodes that are then used to modulate the contribution of each node during neighbourhood aggregation. Here, improved accuracy occurs at a cost to training time. Including learnable attention parameters in the SIGN architecture would limit speed gains achieved during training and inference. Instead, Frasca et al. suggest it may be possible to implement attention in the precomputation stage by fixing attention-like weights to the linear graph aggregation operations [12]. To the best of the author's knowledge, this approach has not been explored in literature.

Therefore, the present study investigates the use of an attention filter during the precomputed aggregation, thus making the diffusion operator dependent on node features. Several methods to approximate attention parameters were explored, including learnable weights obtained from a trained GAT model, scaled dot product attention common to Transformer models, and Cosine Similarity. The goal of the proposed study is to answer the question: **To what extent does the inclusion of a precomputed approximated attention matrix impact performance of the SIGN approach?** In doing so, several sub-questions must be addressed, including: (1) What limits the scalability of GNNs? (answered above) (2) How do scalable full-graph approaches address these issues? (3) How can attention weights be approximated? Finally, (4) How does the inclusion of an approximated attention filter affect the accuracy and time complexity of SIGN?

2 RELATED LITERATURE

During the GNN encoding process, learning can take place in a *Transductive* setting, where the entire graph is known at the time of training, or an *Inductive* setting where training and testing are performed using different graphs [7]. A review of GNN architectures like GCNs and introduction of decoupled architectures are presented (Sections 2.1 and 2.2, respectively). Section 2.3 outlines existing attention mechanisms that may be implemented to compute the weights of the proposed approximate attention filter. Finally, supplemental appendices A and B provide a compendium of used notions and abbreviations.

2.1 Graph Neural Networks

Basic Notation. Let graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be an undirected input graph containing nodes $v_i \in \mathcal{V}$ and edges $e_{i,j} \in \mathcal{E}$. Let $N = |\mathcal{V}|$ be the number of nodes. Relational graph structure is captured by adjacency matrix $A \in \mathbb{R}^{N \times N}$, where $A_{i,j} = 0$ indicates that node v_i and v_j are not connected; otherwise $A_{i,j} = 1$ if nodes $(v_i, v_j) \in \mathcal{E}$. When self-loops are added to the graph, the resultant adjacency matrix is denoted as $\hat{A} = A + I$, where I represents an identity matrix. Node entities often include a feature vector stored in a feature matrix defined as $X \in \mathbb{R}^{N \times F}$, where F is the dimension of the node features and x_i indicates the feature vector of node v_i (the i-th row of X).

Convolution Operators. Graph convolutional networks introduced a first-order approximation of computationally expensive spectral GNN methods to propagate information from single-hop neighbours and self-loops; layers are stacked to obtain filters with larger receptive fields [21]. At each layer, vector representation $h_i^{(l+1)}$ is updated according to information aggregated from v_i's l-order neighborhood [14]. Node-level embeddings are then learned recursively through the layer-wise propagation as:

$$H^{(l+1)} = \sigma((\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}})H^{(l)}\Theta^{(l)}) \quad (1)$$

$$= \sigma(\tilde{A}H^{(l)}\Theta^{(l)}) \quad (2)$$

which can be generalized to the linear function:

$$f(X, A) = \tilde{A}X\Theta \tag{3}$$

where $H^{(l)}, H^{(l+1)} \in \mathbb{R}^{N\times F'}$ are the current and subsequent hidden representation of the node network of dimension F', $\sigma(\cdot)$ is a non-linear activation function such as rectified linear activation (ReLU), $\hat{D}$ is the normalized diagonal degree matrix of $\hat{A}$, and $\Theta^{(l)}$ is a trainable weight matrix of layer l. The symmetric normalized matrix ($\tilde{A}$) acts as a linear diffusion operator, where $\tilde{A}X$ in diffuses the "signal" (i.e. node features) over the graph [4]. Representations of one convolutional layer $H^{(l)}$ serve as input to the subsequent layer $H^{(l+1)}$. $H^{(0)} = X$ in the first layer whereas $Z \in \mathbb{R}^{N\times F}$ defines the final representation $H^{(L)}$. For downstream tasks, the model is trained in a fully-supervised manner where loss is defined using a softmax classification function and negative log-likelihood loss or cross-entropy [9, 14].

2.2 Scalable Full-Graph Approaches

The development of scalable full-graph models is based on several empirical observations. First, stacking L convolutional layers with element-wise non-linearity $\sigma(\cdot)$ increases the receptive field by aggregating each node's neighborhood L-hops away (Eq. 4); an additional layer with activation $\delta(\cdot)$ like softmax is used to classify nodes [21]. Furthermore, results by Wu et al. (2019) show that a single convolutional filter with a large receptive field ($\tilde{A}^L X$ in Eq. 5) *approximates* multiple layers with small filters [34]. The removal of all but the last non-linearity layer $\delta(\cdot)$ permits the decoupling of linear computations $\tilde{A}X$ and use of a simple MLP to determine the network weights ($\Theta = \Theta^{(1)}\Theta^{(2)}\cdots\Theta^{(L)}$). The simplified GCN (SCN) model, introduced by Wu et. al., preserves accuracy while reducing training time by orders of magnitude [34].

$$Y = \delta(\tilde{A}\cdots\sigma(\tilde{A}X\Theta^{(1)})\cdots\Theta^{(L)}) \tag{4}$$

$$Y = \delta(\tilde{A}^L X\Theta) \tag{5}$$

Whereas SCN employs a single diffusion operator $\tilde{A}$ of power L, the Scalable Inception Graph Network uses inception modules – as found in image classification architectures – to incorporate multiple linear diffusion operators of various sizes and powers [12]. For a given diffusion operator B, $A_L = B^L$ for $L = 1, \ldots, r$, where r represents the size of the convolutional filter; $r = 0$ corresponds to 1×1 convolution filter, which translates to a linear transformation of the features in each node without diffusing across nodes. The SIGN architecture has the form:

$$Z = \sigma([X\Theta_0 \mid A_1X\Theta_1 \mid \ldots \mid A_rX\Theta_r]) \tag{6}$$

$$Y = \delta(Z\Omega) \tag{7}$$

where $A_r \in \mathbb{R}^{N\times N}$ matrices denotes the linear diffusion operator of size r, A_rX is the precomputed matrix product, $\Theta_0, \ldots, \Theta_r \in \mathbb{R}^{F\times F'}$ and $\Omega \in \mathbb{R}^{F'(r+1)\times c}$ for c classes are learnable weight matrices of the Inception and Classification modules, respectively. $\sigma(\cdot)$ represents non-linear intermediary activation functions and $\delta(\cdot)$ represents the final activation function used for downstream tasks. Following the precomputation of A_rX for each $r \in L$, matrix products are passed through their own MLP (Inception module Θ). The concatenated tensor is passed through another MLP (classification module Ω) to obtain the final results $y_i \in Y$.

This precomputed data transformation facilitates the use of a simple MLP network and training with mini-batch gradient descent without the need for graph-sampling procedures. Compared to ClusterGCN and GraphSAINT, SIGN reduces training time by 92.5% and 65.5%, respectively. Since sampling only accelerates the time of training [22], SIGN improved inference time by 96.8% and 96.9%, respectively. While SIGN can scale to graphs over 100 million nodes, the sample-based models achieved higher test accuracy scores than SIGN (79.0% and 79.1% vs 77.6%). The difference in accuracy evinces that linear diffusion operations are not an exact replacement for convolutional feature propagation, trading off model expressiveness for simplicity. Still, some techniques exist to improve performance accuracy.

2.3 The Attention Mechanism

Attention is an essential concept in deep learning inspired by cognitive mechanisms enabling humans to focus on relevant information when presented with large amounts of data. In neural network layer aggregations, learned attention weights enhance the effect of some input while diminishing less relevant parts. Attention has been used to improve model accuracy for tasks such as text classification, machine translation, action recognition translation, and graph learning [25].

Transformer-Based Attention. The introduction of the Transformer architecture revolutionized language modelling due to the effectiveness of the self-attention [30]. Self-attention uses the scaled dot-product attention mechanism (DPA) to embed additional contextual information into input word embeddings by learning the relevance of other words to their meaning *within* a sentence. DPA is computed as follows:

$$Attention(Q, K, V) = softmax(\frac{Q \cdot K^T}{\sqrt{d_k}})V \tag{8}$$

where $Q, K, V \in \mathbb{R}^{N \times d_m}$ represent query, key, and value embedding matrices of N sequences with d_m-dimensional. Each embedding matrix is projected into d_K- or d_V-dimensional sub-spaces. Self-attention, also called "intra-attention", refers to the attention weight matrix $\alpha \in \mathbb{R}^{N \times N}$ produced by $softmax(\cdot)$ when Q and K are the same matrix representation; α maps contextual information of every possible pair onto V. As d_m increases, so to does the magnitude of the dot-product similarity scores ($Q \cdot K^T$). Therefore, a scaling factor $\sqrt{d_k}$ is included to maintain an acceptable variance, stabilizing the softmax function.

Multi-head self-attention (MHA) expands this concept by concurrently attending to information at different positions (or "heads") of the embedding. Instead of attending to d_m-dimensional embeddings, Q, K, and V matrices are linearly projected h-times with different trainable weights to compare attention heads. MHA is formalized as:

$$MultiHead(Q, K, V) = [head_1 \mid \ldots \mid head_h]W^O \tag{9}$$

$$head_i = softmax(\frac{QW_i^Q (KW_i^K)^T}{\sqrt{d_k}})VW_i^V \tag{10}$$

where $W_i^Q, W_i^K \in \mathbb{R}^{d_m \times d_K}$, $W_i^V \in \mathbb{R}^{d_m \times d_V}$, and $W_i^O \in \mathbb{R}^{kd_V \times d_m}$ are learned parameter matrices, and d_K and d_V are hidden dimensions of the projected sub-spaces.

Transformers are comprised of encoder-decoder structures, each containing several layers of multi-head self- attention and feed-forward network sub-layers [30], whose network and attention weights are learned during training. In the absence of encoder-decoder structures, direct implementation of DPA into SIGN is non-trivial and requires modification, discussed in Section 3.

Graph-Based Attention. Whereas Graph convolutional operators generate node-level embeddings dependent on topology, graph attention layers expand representation dependencies to include node features [31]. Each GAT layer implicitly assigns learned importance weights to different nodes within a neighbourhood during aggregation. Unlike self-attention found in Transformers that compares different positions *within* a sequence, graph-based attention compares sequences (node features) *between* nodes. The formalism of a GAT layer is defined as:

$$e_{ij}^{(l)} = \sigma(\vec{a}^{(l)T}(z_i^{(l)} \| z_j^{(l)})) \tag{11}$$

$$\alpha_{ij}^{(l)} = \frac{exp(e_{ij}^{(l)})}{\sum_{k \in N(i)} exp(e_{ik}^{(l)})} \tag{12}$$

$$h_i^{(l+1)} = ||_{k=1}^{K} \sigma(\sum_{j \in N_i} \alpha_{ij}^{(l)k} W^k z_j^{(l)}) \tag{13}$$

where z_i is the linear transformation, parameterized by weighted matrix $W \in \mathbb{R}^{F' \times F}$, of node v_i in layer l. The attention mechanism parameterized by a weight vector $\vec{a} \in \mathbb{R}^{2F'}$, and $\sigma(\cdot)$ represents the LeakyReLU activation function used to determine the importance scores between nodes v_i and v_j (Eq. 11). Equation 12 normalizes the attention score between two nodes relative to the other incoming edges in node v_i's neighborhood $\mathcal{N}(i)$ using the softmax function. Aggregation performed in the GAT layer scales the embedding of neighboring nodes by the learned attention weights per attention head k; MHA weights are averaged over all heads in the final layer, otherwise they are concatenated (Eq. 13).

Graph attention, and consideration of node features, improve the expressive power of GNNs compared to topological models like GCN, achieving top accuracy scores for node classification tasks [31]. However, there exists a trade-off between accuracy and computation. Compared to a single GCN layer, GAT increases the time complexity from $O(|\mathcal{V}|F^2)$ to $O(|\mathcal{V}|FF' + h|\mathcal{E}|F')$ [8, 31]. The GAT model must learn attention vector $\vec{a}$ between all neighbouring nodes and parameter matrix W at each layer l, for each head k, at each forward pass during training. Neighbourhood sampling techniques assuage memory complexity during training but introduce limitations associated with graph sampling [9, 13, 22].

3 APPROXIMATE ATTENTION FILTER

3.1 Motivation

Inspired by previous applications of attention mechanisms, the present study proposes using approximated attention during the pre-aggregated linear diffusion operations (AX) of SIGN. Attention weights are generated and used as a context-mapping filter during diffusion, thus approximating convolution-like attention (Section 3.2). In this framework, the attention filter aims to approximate layer-wise learnable attention weights and add feature-dependent information to the approximated convolutional neighbourhood aggregations. Figure 1 outlines the overall procedure where $\breve{A} = \alpha A$, where $\alpha \in \mathbb{R}^{N \times N}$ represents the matrix of attention weights that augments the diffusion operator A (Figure 1, top). The updated $\breve{A}$ is functionally identical to A, and thus linear diffusion and subsequent Inception modules remain unchanged. Linear diffusion is performed for varying filter sizes; each diffusion output is individually passed to an MLP and concatenated in the Inception module, and then passed to the Classification module for downstream node label prediction (Figure 1, bottom)

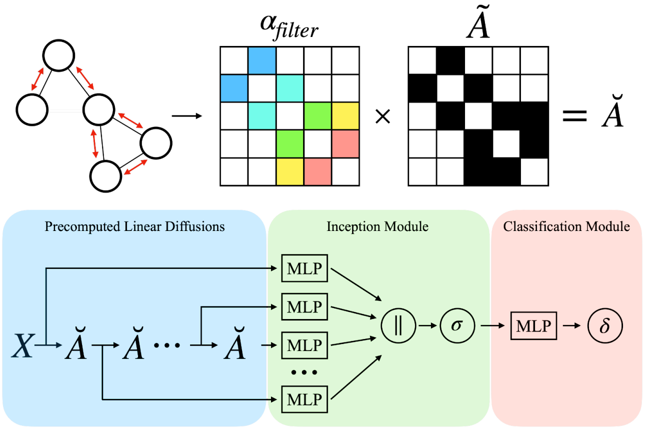


Figure 1: The proposed workflow. *Top*: Similarity weights are computed for edge-wise node pairs and multiplied element-wise by the adjacency matrix. Bottom: Linear diffusion operations are precomputed (blue), and individually fed into a MLP and concatenated (green). Output from non-linear activation in the Inception module is fed into the classification module for downstream node prediction tasks (red).

3.2 Proposed Attention Weight Methods

To address the second sub-question, three methods of approximating attention weights are proposed, including using learnable GAT weights, Cosine Similarity scores, and a modified implementation of DPA including single- and multi-head attention (SHA and MHA, respectively).

Graph Attention. The GAT layers attend to relevant neighbours during recursive aggregation by learning attention weights during training. To generate the feature-dependent linear diffusion operator $\breve{A}$, a GNN with GAT layers is trained over several epochs and attention weights α_{GAT} are extracted from the final layer. The neighbourhood sampling procedure, as found in GraphSAGE, can be used to facilitate mini-batch training necessary for large-scale datasets [13]. Once $\breve{A}$ is computed, feature diffusion is performed, and SIGN is implemented as initially described by Fransca et. al. [12]. In practice, the GNN and SIGN models are trained consecutively. Using learnable attention weights may confer more expressive power to SIGN. Still, computational complexity remains a limiting factor even if limited to the precomputed approximated aggregations.

Cosine Similarity. Explicit comparison scores may be used to avoid the need for network weight training and the cost of implicitly deriving similarities between node features. Cosine Similarity (CS) is a popular measure of similarity to compare vectors projected in a multi-dimensional space and is frequently used for language processing tasks [15].

CS is defined as the division between the vector dot products and the product of the euclidean norms; a small value of ϵ is used to avoid division by zero (Eq. 14). Similarity scores range from $\{1, +1\}$ indicating that the node feature vectors are entirely dissimilar, orthogonal, or practically identical (CS=-1, 0, and +1, respectively).

$$sim(x, y) = \frac{x \cdot y}{max(\|x\|_2 \cdot \|y\|_2, \epsilon}) \tag{14}$$

In this approach, α_{CS} is an $N \times N$ matrix filter containing CS scores between all node pairs $(v_i, v_j) \in \mathcal{E}$

Scaled Dot-Product Attention. Lastly, a modified scaled dot-product attention mechanism generates the approximate attention filters α_{SHA} and α_{MHA}. Several distinctions from the previously described Transformer-based attention exist. First, unlike self-attention applied *within* a sentence embedding (Eq. 8), the modified DPA mechanism computes "inter-attention" *between* feature vectors of $(v_i, v_j) \in \mathcal{E}$, where Q and K correspond to representation vectors $x_i, x_j \in \mathbb{R}^{1\times d_K}$ respectively. Self-attention is avoided as Q and K are different inputs, and comparing individual node pairs instead of entire matrices avoids unnecessary comparisons where edges do not exist. Secondly, we are only interested in the attention matrix generated by the softmax function in Eq. 10; thus, the V input embeddings are ignored. Lastly, parameterized weights W_i^Q, W_i^K, and W_i^O are not trained due to the lack of encoder-decoder structures in SIGN traditionally used to learn attention weights in a Transformer. Therefore, linear projections are solely used to facilitate MHA, and α approximates trained attention weights as W_i^Q, W_i^K, and W_i^O are initialized but not learned. As before, α_{SHA} and α_{MHA} are $N \times N$ matrix filters containing DPA values between all node pairs $(v_i, v_j) \in \mathcal{E}$.

4 EXPERIMENTS

The current study evaluated the efficacy and efficiency of an approximate attention filter within the SIGN framework by completing node classification tasks on three benchmark datasets, including Pubmed, ogbn-arXiv, and ogbn-products [16, 35]. A robust investigation of model accuracy and time complexity was performed to answer sub-question three more completely. Predictive power of each model was assessed by the micro-averaged F1 score, previously reported in literature [12]. Node property predictions were performed on nodes unseen during training. However, all edge-wise node pairs were used to compute the approximated attention filter, making the experiment a supervised, transductive learning

Table 1: Theoretical time complexity of the base linear diffusion aggregation and proposed attention filters.

	Function	*Time Complexity*
SIGN-k	Linear Diffusion	$O(r\|\mathcal{E}\|F)$
α_{GAT}	Attention Filter	$O(hL\|\mathcal{V}\|FF' + hL\|\mathcal{E}\|F')$
α_{CS}	Attention Filter	$O(\|\mathcal{E}\|F)$
α_{SHA}	Attention Filter	$O(\|\mathcal{V}\|FF' + \|\mathcal{E}\|F'), \|F'\| = \|F\|$
α_{MHA}	Attention Filter	$O(\|\mathcal{V}\|FF' + \|\mathcal{E}\|F'), \|F'\| = h\|F\|$

scenario. A detailed description of the experimental setup, datasets, and hardware are presented in Sections 4.1- 4.3, respectively.

4.1 Experimental Setup

Pre-aggregated Linear Diffusion. A baseline normalized adjacency matrix diffusion operator $\tilde{A}$ was used across all experiments. Mentioned previously, SIGN can accommodate other operators such as PersonalizedPage Rank (PPR), heat and triangle-based diffusion; yet $\tilde{A}$ was used to simplify the evaluation of the attention filter [12]. Experimental models include SIGN (baseline) and SIGN with an attention filter computed by GAT, CS, and DPA-based attention including single- and multi-head attention (SHA and MHA, respectively). A summary of attention and diffusion procedures within the pre-aggregation step and their theoretical time complexities are presented in Table 1. Appendix C provides supplemental details on their estimations.

Graph attention weights require first training a GNN model with GAT layers and then extracting learned weights. For the non-learnable attention approximations, min-max normalization was applied row-wise to α_{CS}, α_{SHA} and α_{MHA} such that weights values ranged between zero and one. Once attention weights were obtained, the filter was fixed to the diffusion operator for precomputation of convolution-like r-hop aggregations (Figure 1).

SIGN & GAT Architectures. The same SIGN architecture was used across all experiments and closely followed the architecture first proposed by Frasca et al. [12]. The model included inception (Θ) and classification (Ω) modules consisting of multiple feed-forward layers as shown in equations 6 and 7, and illustrated at the bottom of Figure 1. The Parametric ReLU (PreLU) function was used during intermediate activations, while the logarithmic softmax was used as the final activation function. Batch normalization was used at each feed-forward layer to accelerate training [19]. The GNN architecture used to generate α_{GAT} contained at least two GAT layers with ReLU intermediate activations; the logarithmic softmax function was used for the final activation. Neighbourhood sampling was performed to facilitate mini-batch gradient descent with the Adam optimizer [13, 20].

Hyperparameter Optimization. Hyperparameter tuning was conducted separately for each experiment, per dataset, to determine the top-performing SIGN and GAT configurations, as determine by validation loss. For SIGN and GAT, tuning parameters were found by minimizing validation negative log-likelihood loss via mini-batch gradient descent with the Adam optimizer [20]. Experiments were run for a maximum of 300 epochs. Early stopping mechanics were applied with a patience of 20. A learning rate monitor reduced the learning rate by a factor of 0.1 if validation loss did not improve after five epochs [23]. Optimization sweeps were performed with a Bayesian search method to minimize validation loss [1]. Appendix D provides a list and value ranges of hyperparameters for SIGN and generation of α_{GAT}, α_{CS}, α_{SHA}, and α_{MHA} filters.

4.2 Data

Experiments were performed on three benchmark graphs available to the public [11, 16]. No data processing was performed on Pubmed and ogbn-products; self-loops and reverse edges were added to ogbn-arXiv [33], making all three graphs undirected. Prediction tasks for all three included multi-label single-class node classification. The graphs were selected due to large variations in graph structure, size, and connectivity. Descriptive statistics for each graphs were calculated (Table 2). Supplemental information about the datasets, prediction tasks, and data splitting protocols are presented in Appendix E.

Table 2: Summary of graph-level statistics.

	Pubmed	*arXiv*	*products*
Setting	Trans.	Trans.	Trans.
$\|\mathcal{V}\|$	19 717	169 343	2 449 049
$\|\mathcal{E}\|$	44 338	1 242 471	61 859 140
F	500	128	100
Classes	3	40	47
Directed	False	False	False
Avg. Deg.	5.5 ± 7.4	14.7 ± 68.6	51.5 ± 95.9
Deg. Range	[2 – 172]	[2 – 13 162]	[1 – 17 482]
Homophily	0.802	0.678	0.811
Train%	92.4%	53.7%	8.0%
Validation%	2.50%	17.6%	1.6%
Test%	5.10%	28.7%	90.4%

4.3 Hardware Configuration

This work was carried out on the LISA supercomputer, the Dutch national e-infrastructure with the support of SURF Cooperative, at the University of Amsterdam. All models were implemented using Python 3.9.7, Pytorch 1.11, and Pytorch Geometric 2.0.4 [11, 26] on a single high-performance cluster node with a GeForce 1080Ti-11GB GDDR5X GPU, and Intel(R) Xeon(R) Bronze 3104 CPUs 1.70GHz and 256GB of RAM. Experiment tracking and hyperparameter optimization were performed using Weights & Biases [3].

5 RESULTS

Node classification accuracy and time complexity metrics are presented in Sections 5.1 and 5.2. Ten independent runs over a range of fixed global seed values were performed. Models were trained over 300 epochs, and evaluation metrics were collected every tenth epoch. Metric averages and standard deviations were computed using data from the best epoch of each run, as defined by the highest validation F1 score [12]. Experiments using GAT and subsequent SIGN+GAT models were not conducted on the larger OGB datasets due to out-of-memory (OOM) errors. Supplemental tables containing optimized hyperparameter values, size of the model configurations, and complete summaries of accuracy are reported in Appendices F- H

5.1 Node Classification Accuracy

The experimental results suggest that using an approximated attention filter fixed to the diffusion operator incrementally improved SIGN accuracy; more substantial improvements were observed for larger datasets, as shown in Table 3. On the Pubmed dataset, SIGN+GAT was the single exception to the observed trend. The baseline SIGN model outperformed GAT by 4.9%, and subsequent inclusion of α_{GAT} reduced SIGN accuracy by 0.4%. Marginal accuracy improvements between 0.6% – 0.9% were achieved using CS- and DPA-based filters. Results on ogbn-arXiv found substantial accuracy improvements with the use of α_{MHA}, α_{SHA}, and α_{CS} attention filters, outperforming the baseline SIGN model by 2.5%, 5.0%, and 5.5%, respectively. Similarly, on the larger ogbn-products dataset, SIGN with an attention filter outperformed the baseline SIGN model by between 0.8% and 1.8%. While differences on Pubmed were negligible, comparisons between filters found that SIGN+SHA consistently outperformed SIGN+MHA (ogbn-arXiv: +2.0%; ogbn-products: +1.0%). SIGN+CS outperformed all models on ogbn-arXiv; equivalent accuracy was achieved with α_{CS} and α_{MHA} on ogbn-products.

5.2 Runtime

Table 4 contains average preprocessing run times that account for filter weight computation and diffusion operations. Compared to the baseline SIGN model, the addition of an attention filter increased precomputation time proportional to the time required to generate filter weights. Generating α_{GAT} was cost-prohibitive due to the time needed to train a GAT model, increasing precomputation time from 0.12 to 357 seconds. Performance time to compute α_{MHA} was consistently greater than α_{CS} and α_{SHA} as the number of attention heads expanded the number of weight calculations. On the larger datasets, SIGN+CS exhibited increased performance times compared to SIGN+SHA despite the shorter theoretical complexity, suggesting inefficiencies in the CS implementation.

Table 3: Comparison of Micro-averaged F1 *test* Scores (%) (average ± standard deviation). Bold indicates top performing configuration.

	Pubmed	*arXiv*	*products*
GAT	84.6 ± 0.3	*OOM*	*OOM*
SIGN	89.5 ± 0.6	54.7 ± 0.4	75.5 ± 0.5
SIGN+GAT	89.1 ± 0.1	*OOM*	*OOM*
SIGN+CS	90.1 ± 0.4	**59.6 ± 0.2**	76.3 ± 0.3
SIGN+SHA	90.1 ± 0.5	59.1 ± 0.4	**77.3 ± 0.3**
SIGN+MHA	**90.4 ± 0.4**	57.1 ± 0.3	76.3 ± 0.2

Table 4: Prepossessing time (in seconds) (average ± standard deviation). Bold indicates fastest performing configuration.

	Pubmed	*arXiv*	*products*
SIGN	**0.11 ± 0.00**	**0.86 ± 0.01**	**109.76 ± 0.53**
SIGN+GAT	356.66 ± 5.66	*OOM*	*OOM*
SIGN+CS	0.32 ± 0.00	2.51 ± 0.02	299.74 ± 2.45
SIGN+SHA	0.54 ± 0.01	1.90 ± 0.03	214.39 ± 3.09
SIGN+MHA	2.16 ± 0.01	6.03 ± 0.10	559.47 ± 12.6

6 DISCUSSION

The Scalable Inception Graph Network provides a sampling-free model that efficiently scales to graphs containing hundreds of millions of nodes [12]. When first introduced, SIGN achieved state-of-the-art performance times while also outperforming existing GNN models, making the architecture a baseline benchmark model for large-scale graphs.

More recently, newer models are favoured over SIGN due to improved accuracy, despite no reports of their execution speed [16]. However, the accuracy of SIGN may be improved by enhancing the topology-based diffusion operations with feature-dependent information. Therefore, we propose fixing a novel approximated attention filter characterizing feature-similarity to the diffusion operator during preprocessing. The proposed approach is a natural extension of SIGN and other decoupled GNN architectures, potentially improving accuracy without affecting training and inference times.

Experiments were conducted to assess the efficacy and efficiency of several attention filters composed of learned graph attention weights, cosine similarity, or an untrained dot product attention mechanism. Our results found that adding an attention filter containing CS or DPA similarity scores conferred improved predictive power of SIGN at a modest cost to precomputation time. Graph attention (α_{GAT}) was the lone exception to the observed trend, reducing the accuracy of SIGN by 0.4% while increasing preprocessing time by 3000 fold. The utility of the α_{GAT} filter may be limited by the expressivity of the GAT model, as the baseline SIGN model outperformed GAT by 4.9%. Furthermore, implementation of graph attention requires training the GAT and SIGN models consecutively. Graph attention may be impractical as the GAT model could not scale to larger experimental graphs.

The extent of accuracy gains imparted by the attention filter varied between the three distinct datasets. Incremental gains were observed on the Pubmed dataset ($0.6\% - 0.9\%$), but were within the standard deviation of the baseline SIGN model. Alternatively, attention filters were more efficacious on ogbn-products and ogbn-arXiv, improving accuracy by $0.8\% - 1.8\%$ and $2.5\% - 5.5\%$, respectively.

As previously mentioned, the intuition of the filter is to attune diffusion based on the similarity between node features. GNNs implicitly assume homophily in graph data, where connections are more likely to occur between nodes of the same class or similar features. An investigation by Hussain et al. (2021) found that increasing the homophily of a graph reliably improved model accuracy, regardless of the GNN method or dataset [18]. In the present study, Pubmed and ogbn-products exhibited high homophily (both $> 80\%$); therefore, there may be less to gain from the additional similarity bias of the attention filter. The most significant accuracy gains were observed on ogbn-arXiv, the graph with the lowest homophily used in the present study (67.8%).

In contrast to learned attention weights, three alternative methods were explored to compute the explicit similarity between node features and included cosine similarity and single- and multi-head dot product attention. CS and DPA scores are inherently related as the former compares vectors by the cosine angle between them, whereas the latter considers both the angle and magnitude [28, 30]. Implementation of DPA differed from Vaswani et al. as parameterized weight matrices were not trained due to SIGN's lack of encoder-decoder structures. The experiment results found that SIGN+SHA outperformed SIGN+MHA by $1\% - 2\%$, potentially due to the increased number of initialized, untrained weights. SIGN+CS and SIGN+SHA were the top-performing experimental configurations for ogbn-arXiv and ogbn-products (59.6% and 77.3%, respectively).

The primary advantage of SIGN is performance speed and scalability, and should be considered when modifying the architecture [12]. Analysis of preprocessing times found that inclusion of learned parameters, such as α_{GAT}, is cost-prohibitive and hinders speed gains of SIGN. Similarly, multi-head attention requires additional computational time as scores are calculated per head, making it less advantageous compared to α_{SHA} or α_{CS}. *In toto*, results found that moderate accuracy gains are achieved using CS- and DPA-based approximated attention filters without prohibitively increasing performance time compared to the baseline SIGN configuration.

Limitations: While the present study successfully demonstrated the efficacy and efficiency of attention filters within the SIGN framework, several limitations exist. First, the reported accuracy gains are limited to a single diffusion operator and may be less effective as multiple operators are shown to improve model expressiveness [12]. Additionally, the reported runtimes reflect execution time on a single GPU node (11GB) with 90GB of CPU RAM. Frasca et al. note that SIGN is easily parallelized with distributed computing [12]. Likewise, CS and DPA scores may be computed in parallel as loops are used to compare edge-wise node pairs in batches.

Lastly, while implementation of the baseline SIGN model achieved comparable accuracy results to Frasca et. al. on ogbn-products (75.5% vs. 76.5%) [12], the same implementation on ogbn-arXiv resulted in significantly worse accuracy compared to literature despite identical data transformation and hyperparameters (54.7% vs. 72.0%) [16, 33]. An extensive investigation found that the Deep Graph Library (DGL) diffuses node features differently than the built-in PyTorch Geometric `transform.SIGN` class [11]. This discrepancy must be addressed to more accurately determine the efficacy of an approximated attention filter in the SIGN framework.

Future Work: Despite promising results, further investigation is required. Future work should examine whether accuracy gains are achieved when multiple operators are used. While tuning optimization results indicated that min-max row-wise normalization was preferable, alternative α_{filter} normalization schemes should be investigated. Furthermore, additional experimentation on other graphs is required to confirm that the attention filter is more advantageous for learning on graphs with low homophily.

7 CONCLUSION

The present study proposed a novel approximated attention filter to improve the predictive power of the scalable SIGN model by attending to node features during the precomputed linear diffusion aggregations. The attention filter is suitable for any GNN model that decouples graph convolutions and the scalable classifier. Experimental results on Pubmed, ogbn-arXiv and ogbn-product benchmark datasets demonstrate accuracy gains with the use of CS- or DPA-generated attention filters without increasing precomputation time by orders of magnitude. The use of graph attention weights is limited by the performance and time complexity of the underlying model and did not improve performance. Furthermore, the results indicate that larger graphs and graphs with less homophily benefit more from applying the approximated attention filter. Lastly, the SIGN model accommodates multiple types of diffusion operators; thus, future work should investigate the efficacy of attention filters in more complex SIGN configurations.

Acknowledgements

I sincerely thank Dr. Paul Groth, PhD., and Jeroen den Boef, MSc. for their invaluable advice and guidance throughout this investigation. Additionally, I would like to thank Social-Database for the opportunity and the Data Science team – Joran Cornelisse, MSc., Chiel Heldoorn, MSc., and Dr. Alexandra Rusu, PhD. – for their relevant (and irrelevant) conversations.

REFERENCES

[1] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. 2011. Algorithms for hyper-parameter optimization. *Advances in neural information processing systems* 24 (2011).

[2] K. Bhatia, K. Dahiya, H. Jain, P. Kar, A. Mittal, Y. Prabhu, and M. Varma. 2016. The extreme classification repository: Multi-label datasets and code. http://manikvarma.org/downloads/XC/XMLRepository.html

[3] Lukas Biewald. 2020. Experiment Tracking with Weights and Biases. https://www.wandb.com/ Software available from wandb.com.

[4] Myriam Bontonou, Carlos Lassance, Jean-Charles Vialatte, and Vincent Gripon. 2019. A unified deep learning formalism for processing graph signals. *arXiv preprint arXiv:1905.00496* (2019).

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

[6] Michael M Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. 2021. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *arXiv preprint arXiv:2104.13478* (2021).

[7] Ines Chami, Sami Abu-El-Haija, Bryan Perozzi, Christopher Ré, and Kevin Murphy. 2020. Machine learning on graphs: A model and comprehensive taxonomy. *arXiv preprint arXiv:2005.03675* (2020).

[8] Ming Chen, Zhewei Wei, Bolin Ding, Yaliang Li, Ye Yuan, Xiaoyong Du, and Ji-Rong Wen. 2020. Scalable graph neural networks via bidirectional propagation. *Advances in neural information processing systems* 33 (2020), 14556–14566.

[9] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. 2019. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 257–266.

[10] Jeroen B den Boef, Joran Cornelisse, and Paul Groth. 2021. GraphPOPE: Retaining Structural Graph Information Using Position-aware Node Embeddings. (2021).

[11] Matthias Fey and Jan E. Lenssen. 2019. Fast Graph Representation Learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*.

[12] Fabrizio Frasca, Emanuele Rossi, Davide Eynard, Ben Chamberlain, Michael Bronstein, and Federico Monti. 2020. Sign: Scalable inception graph neural networks. *arXiv preprint arXiv:2004.11198* (2020).

[13] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. *Advances in neural information processing systems* 30 (2017).

[14] William L Hamilton. 2020. Graph representation learning. *Synthesis Lectures on Artifical Intelligence and Machine Learning* 14, 3 (2020), 1–159.

[15] Jiawei Han, Jian Pei, and Micheline Kamber. 2011. *Data mining: concepts and techniques*. Elsevier.

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

[17] Wenbing Huang, Tong Zhang, Yu Rong, and Junzhou Huang. 2018. Adaptive sampling towards fast graph representation learning. *Advances in neural information processing systems* 31 (2018).

[18] Hussain Hussain, Tomislav Duricic, Elisabeth Lex, Denis Helic, and Roman Kern. 2021. The interplay between communities and homophily in semi-supervised classification using graph neural networks. *Applied Network Science* 6, 1 (2021), 1–26.

[19] Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*. PMLR, 448–456.

[20] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[21] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[22] Xin Liu, Mingyu Yan, Lei Deng, Guoqi Li, Xiaochun Ye, and Dongrui Fan. 2021. Sampling methods for efficient training of graph convolutional networks: A survey. *IEEE/CAA Journal of Automatica Sinica* 9, 2 (2021), 205–234.

[23] Ilya Loshchilov and Frank Hutter. 2018. Fixing weight decay regularization in adam. (2018).

[24] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems* 26 (2013).

[25] Zhaoyang Niu, Guoqiang Zhong, and Hui Yu. 2021. A review on the attention mechanism of deep learning. *Neurocomputing* 452 (2021), 48–62.

[26] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 8024–8035. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[27] Christo Petrov. 2022. Twitter Statistics 2022. https://techjury.net/blog/twitter-statistics/

[28] Grigori Sidorov, Alexander Gelbukh, Helena Gómez-Adorno, and David Pinto. 2014. Soft similarity and soft cosine measure: Similarity of features in vector space model. *Computación y Sistemas* 18, 3 (2014), 491–504.

[29] Statista. 2022. Most popular social networks worldwide as of January 2022, ranked by number of monthly active users. https://www.statista.com/statistics/272014/global-social-networks-ranked-by-number-of-users/

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

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

[32] Kuansan Wang, Zhihong Shen, Chiyuan Huang, Chieh-Han Wu, Yuxiao Dong, and Anshul Kanakia. 2020. Microsoft academic graph: When experts are not enough. *Quantitative Science Studies* 1, 1 (2020), 396–413.

[33] Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, Chao Ma, Lingfan Yu, Yu Gai, Tianjun Xiao, Tong He, George Karypis, Jinyang Li, and Zheng Zhang. 2019. Deep Graph Library: A Graph-Centric, Highly-Performant Package for Graph Neural Networks. *arXiv preprint arXiv:1909.01315* (2019).

[34] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. 2019. Simplifying graph convolutional networks. In *International conference on machine learning*. PMLR, 6861–6871.

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

[36] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. 2019. Graphsaint: Graph sampling based inductive learning method. *arXiv preprint arXiv:1907.04931* (2019).

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

Appendices

Appendix A COMMONLY USED NOTATIONS

Table 5: Summary of mathematical notations used throughout the study

Notation	*Definition*
$\|\cdot\|$	The length of a set.
$\|\|\cdot\|\|_2$	L_2 normalization.
$[A\|B]$	The concatenation of matrices A and B.
$A \cdot B$	The dot product of matrices A and B.
B^T	The transpose of matrix B.
ϵ	Small error used to prevent dividing by zero.
$\mathcal{G}$	A graph.
$\mathcal{V}$	A set of nodes (vertices) in a graph.
v	A node $v \in V$.
N	The number of nodes in $\mathcal{V}$.
F	The dimension of a node feature vector.
F'	The dimension of a *hidden* node feature vector.
$X \in \mathbb{R}^{\mathbb{N}\times\mathbb{F}}$	The feature matrix of a graph.
x_i	The feature vector of node i.
$y_i \in Y$	The class label of node i in set Y.
$\mathcal{E}$	A set of pair-wise edges in a graph.
e_{ij}	An edge $e_{ij} \in \mathcal{E}$.
$\mathcal{N}_i$	The neighbors of node i.
$A \in \mathbb{R}^{\mathbb{N}\times\mathbb{N}}$	The graph adjacency matrix.
I	An identity matrix where symmetric graph adjacency matrix. $\hat{A} = A + I$.
$\hat{A}$	A graph adjacency matrix with self-loops. $\hat{A} = A + (A_{ii} = 1)$.
$\hat{D}$	The normalized degree matrix of $\hat{A}$. $\hat{D}_{ii} = \sum_{j=1}^{N} \hat{A}_{ij}$.
$\tilde{A}$	The symmetric normalized adjacency matrix. $\tilde{A} = (\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}})$.
AX	The linear diffusion of node feature matrix X along A.
$l \in L$	The l-th layer in a set of layers within a neural network.
$H^{(l)}$	Hidden graph embedding matrix at layer l.
$h_i^{(l)}$	The hidden embedding of node i in $H^{(l)}$.
A_r	The r-th power of diffusion operator B^L for $L = 1, \dots, r$
$\alpha \in \mathbb{R}^{\mathbb{N}\times\mathbb{N}}$	A linear matrix of attention weights.
$\breve{A}$	The element-wise matrix multiplication of α and $\tilde{A}$.
Q, K, V	The query, key, and value embedding matrices in DPA.
d_m	The embedding dimensions of Q, K, V.
d_k	The hidden embedding dimensions of Q, K following linear transformation.
k	The number of attention heads in MHA.
$\sigma(\cdot)$	The intermediary ReLU, PreLU, or LeakyReLU activation functions.
$\delta(\cdot)$	The logarithmic softmax function for classification tasks.
$Z \in \mathbb{R}^{N\times F}$	The final graph representation $H^{(L)}$.
$W, \Theta, \Omega, \alpha$	Learnable model parameters.

Appendix B LIST OF ABBREVIATIONS

Table 6: Technical abbreviations used in current manuscript.

Acronym	*Definition*
CS	Cosine Similarity
DPA	Scaled Dot-Product Attention mechanism
GAT	Graph Attention Network
GCN	Graph Convolutional Network
GNN	Graph Neural Network
GPU	Graphical Processing Unit memory
GRL	Graph Representational Learning
MHA	Multi-head Attention
MLP	Multilayer Perceptron
OGB	Open Graph Benchmark dataset
OOM	Out-of-memory error
PreLU	Parametric Rectified Linear Unit function
ReLU	Rectified Linear Unit function
SHA	Single-head Attention
SIGN	Scalable Inception Graph Network

Appendix C TIME COMPLEXITY FORMULATION

C.1 SIGN

The theoretical time complexity of the precomputed linear diffusion operations is $O(r|\mathcal{E}|F)$, as noted by the authors [12]. The time complexity of a forward pass in SIGN is $O(rL_{ff}NF^2)$ where L_{ff} is the number of feed-forward layers in the MLP.

C.2 Graph Attention Network

As noted by Velivckovic et al. (2017), the time complexity of a single GAT head at any given layer $O(|\mathcal{V}|FF' + |\mathcal{E}|F')$ where the first calculation is the linear transformation of the vectors and the second is the calculation of the similarity score [31]. Calculating the transformation and similarity scores are affected by the number of convolutional layers, whereas increasing the number of attention heads requires computing similarity scores per head. Therefore, if both layers and the number of attention heads are considered, the expected overall time complexity of the GAT layer is $O(L|\mathcal{V}|FF' + hL|\mathcal{E}|F')$.

C.3 Cosine Similarity

The time complexity of computing the cosine similarity of two F-dimensional vectors is $O(F)$ [28]. To generate α_{CS}, the similarity score is computed for every $(v_i, v_j) \in \mathcal{E}$, increasing the overall time complexity to $O(|\mathcal{E}|F)$

C.4 Dot-Product Attention

First, a linear transformation is performed for every $(v_i, v_j) \in \mathcal{E}$ in Q, K such that $f : \mathbb{R}^{N\times F} \to \mathbb{R}^{N\times F'}$, taking $O(|\mathcal{V}|FF')$ [5]. Next, the dot-product $QW_i^Q \cdot (KW_i^K)^T$ is computed pairwise for $\mathcal{E}$, taking an additional $O(h|\mathcal{E}|F')$. In the present study, the hidden dimension F' resulting from the linear transformation is determined by F. This is not the case for the GAT layer where hidden dimension F' is determined from hyperparameter optimization.

Appendix D HYPERPARAMETER OPTIMIZATION

Architectural and optimization hyperparameters of the GAT and SIGN models, as well as range of considered parameter values across all datasets are presented in Tables 7 and 8. For SIGN and GAT, tuning parameters were found by minimizing validation negative log-likelihood loss via mini-batch gradient descent with the Adam optimizer [20]. Experiments were run for a maximum of 300 epochs. Early stopping mechanics were applied with a patience of 20. A learning rate monitor reduced the learning rate by a factor of 0.1 if validation loss did not improve after five epochs [23]. Optimization sweeps were performed with a Bayesian search method to minimize validation loss [1].

Sweeps of GAT with Neighborhood Sampling were performed on Pubmed, and out-of-memory errors prevented testing on larger datasets. After tuned parameters were identified, a GAT model was trained, attention weights were extracted, and the approximated attention filter was implemented during the precomputation stage of SIGN. Cosine Similarity did not require parameter optimization. Lastly, the number of attention heads was optimized for MHA.

Table 7: Mini-batch GAT hyperparameter search space for *Pubmed* dataset. Neighbors Sampled, −1, indicates that all node v_i's neighbors were sampled. $*$ and † represent $base_2$ and $base_{10}$ distribution increments. Out-of-memory errors prevented sweeps of *ogbn* datasets

	Pubmed
Batch Size	$\{8, \ldots, 1024\}^*$
Learning Rate	$\{1e^{-6}, \ldots, 1e^{-1}\}^\dagger$
Weight Decay	$\{1e^{-6}, \ldots, 1e^{-1}\}^\dagger$
Layers	$\{2, \ldots, 4\}$
Hidden Units	$\{64, \ldots, 512\}^*$
Node Dropout	$\{0.0, \ldots, 0.8\}^\dagger$
Input Attn. Heads	$\{1, \ldots, 3\}$
Output Attn. Heads	$\{1, \ldots, 3\}$
Neighbors Sampled	$[-1, 10, 20, 50, 100, 150, 300, 500]$

Table 8: SIGN hyperparameter search space. Θ and Ω represent Inception and Classification modules. $*$ and $\dagger$ indicate $base_2$ and $base_{10}$ distribution increments. $\star$ and $\diamond$ correspond to CS- and DPA-specific hyperparameters.

	Pubmed	*ogbn*
k-hops	$\{0, \ldots, 5\}$	$\{0, \ldots, 5\}$
Batch Size	$\{64, \ldots, 1024\}^*$	$\{2048, \ldots, 16384\}^*$
Learning Rate	$\{1e^{-6}, \ldots, 1e^{-1}\}^\dagger$	$\{1e^{-6}, \ldots, 1e^{-1}\}^\dagger$
Weight Decay	$\{1e^{-6}, \ldots, 1e^{-1}\}^\dagger$	$\{1e^{-6}, \ldots, 1e^{-1}\}^\dagger$
Θ Layers	$\{1, \ldots, 3\}$	$\{1, \ldots, 3\}$
Θ Hidden Units	$\{64, \ldots, 512\}^*$	$\{128, \ldots, 1024\}^*$
Ω Layers	$\{1, \ldots, 3\}$	$\{1, \ldots, 3\}$
Ω Hidden Units	$\{64, \ldots, 512\}^*$	$\{128, \ldots, 1024\}^*$
Feature Dropout	$\{0.0, \ldots, 0.8\}^\dagger$	$\{0.0, \ldots, 0.8\}^\dagger$
Node Dropout	$\{0.0, \ldots, 0.8\}^\dagger$	$\{0.0, \ldots, 0.8\}^\dagger$
Attention Heads	$\{1, \ldots, 5\}^\diamond$	$\{1, \ldots, 5\}^\diamond$
Row-Normalization	$[True, False]^{\star,\diamond}$	$[True, False]^{\star,\diamond}$

Appendix E DATA BENCHMARKS

E.1 Pubmed

The undirected graph represents the manuscript citation network indexed by Yang et al. (2016) [35]. Nodes represent scientific publications from the PubMed database, and edges indicate one paper citing another. The network includes 19, 717 papers and contains labels specifying the type of diabetes addressed in the publication. Nodes are annotated by term frequency-inverse document frequency weighted word frequencies of the publication's top 500 unique words. The dataset is split into training, validation, and test sets using a conventional random split scheme.

E.2 ogbn-arXiv

While also representing a manuscript citation network, this dataset is a directed graph of Computer Science (CSci) arXiv papers [16, 32]. Papers contain a label corresponding to the $n = 40$ subject areas of the CSci arXiv papers, such as csci.AI, csci.LG, and csci.OS; labels were manually added by the paper's authors and arXiv moderators. Each paper includes a 128-dimensional feature vector obtained by averaging the *WORD2VEC* embeddings of the title, and abstract [24]. All papers include the publication year used to split the data. Unlike the Pubmed dataset, ogbn-arXiv is split based on the publication dates of the papers. This splitting is considered more realistic, as a model could be trained on existing papers and used to predict subject areas of more recent papers. Models were trained on papers published until 2017, validated on papers between 2017-2018, and finally tested on papers published since 2019.

E.3 ogbn-products

The undirected graph represents individual products purchased on Amazon.com and whether the items were purchased together [2, 16]. Each product includes a 100-d feature vector generated from a dimensionality-reduced bag-of-words of the product description and a single target label corresponding to a top-level product category ($n = 47$)[9]. The data is split by the "sales ranking" (e.g. popularity) into training, validation, and test sets. Hu et al. (2020) sort the products by sales and use the top 8% for training, the following top 2% for validation, and the rest for testing [16]. The authors suggest that this splitting procedure more closely matches a real-world scenario where manual labelling is conducted on the most critical nodes and then automated using machine learning to categorize the less popular items.

Appendix F HYPERPARAMETER OPTIMIZATION

Top hyperparameter configurations for GAT, SIGN, SIGN+CS, SIGN+SHA, and SIGN+MHA are reported across all benchmark data in Tables 9 and 14. Mentioned previously, OOM errors prevented training GAT on ogb-arXiv and ogb-products datasets.

Table 9: *GAT*: Best configuration of *Pubmed* benchmark datasets.

	Pubmed
Batch Size	1024
Learning Rate	$1e^{-2}$
Weight Decay	$1e^{-3}$
Layers	2
Hidden Units	8
Node Dropout	0.6
Input Attn. Heads	8
Output Attn. Heads	8
Neighbors Sampled	150

Table 10: *SIGN*: Best configuration of benchmark datasets. Θ and Ω represent Inception and Classification modules.

	Pubmed	*arXiv*	*products*
r-hops	2	2	3
Batch Size	256	2048	4096
Learning Rate	$1e^{-3}$	$1e^{-3}$	$1e^{-2}$
Weight Decay	$1e^{-7}$	$1e^{-3}$	$1e^{-5}$
Θ Layers	2	1	3
Θ Hidden Units	512	128	1024
Ω Layers	3	3	3
Ω Hidden Units	512	512	1024
Feature Dropout	0.3	0.1	0.1
Node Dropout	0.3	0.3	0.4

Table 11: *SIGN-GAT*: Best configuration of *Pubmed* benchmark datasets.

	Pubmed
r-hops	5
Batch Size	64
Learning Rate	$1e^{-3}$
Weight Decay	$1e^{-6}$
Θ Layers	2
Θ Hidden Units	64
Ω Layers	3
Ω Hidden Units	512
Feature Dropout	0.0
Node Dropout	0.4

Table 12: *SIGN+CS*: Best configuration of benchmark datasets. Θ and Ω represent Inception and Classification modules. $*$ indicates parameter used to compute scaled CS.

	Pubmed	*arXiv*	*products*
r-hops	3	2	4
Batch Size	512	2048	16384
Learning Rate	$1e^{-3}$	$1e^{-3}$	$1e^{-2}$
Weight Decay	$1e^{-7}$	$1e^{-7}$	$1e^{-5}$
Θ Layers	3	3	2
Θ Hidden Units	256	1024	1024
Ω Layers	2	3	2
Ω Hidden Units	512	1024	512
Feature Dropout	0.3	0.0	0.1
Node Dropout	0.2	0.6	0.3
Row-Normalization*	$True$	$True$	$True$

Table 13: *SIGN+SHA*: Best configuration of benchmark datasets. Θ and Ω represent Inception and Classification modules. $*$ indicates parameter used to compute scaled DPA.

	Pubmed	*arXiv*	*products*
r-hops	5	2	4
Batch Size	64	16384	8192
Learning Rate	$1e^{-3}$	$1e^{-3}$	$1e^{-3}$
Weight Decay	$1e^{-3}$	$1e^{-4}$	$1e^{-7}$
Θ Layers	2	3	2
Θ Hidden Units	64	1024	1024
Ω Layers	3	3	3
Ω Hidden Units	512	1024	1024
Feature Dropout	0.0	0.0	0.2
Node Dropout	0.4	0.3	0.5
Attention Heads*	1	1	1
Row-Normalization*	$True$	$True$	$True$

Table 14: *SIGN+MHA*: Best configuration of benchmark datasets. Θ and Ω represent Inception and Classification modules. $*$ indicates parameter used to compute scaled DPA.

	Pubmed	*arXiv*	*products*
r-hops	4	2	4
Batch Size	256	4096	8192
Learning Rate	$1e^{-4}$	$1e^{-3}$	$1e^{-3}$
Weight Decay	$1e^{-7}$	$1e^{-6}$	$1e^{-3}$
Θ Layers	3	1	3
Θ Hidden Units	512	512	1024
Ω Layers	2	3	3
Ω Hidden Units	256	512	1024
Feature Dropout	0.2	0.2	0.2
Node Dropout	0.5	0.2	0.5
Attention Heads*	3	5	4
Row-Normalization*	$True$	$True$	$True$

Appendix G MODEL SIZES OF BEST HYPERPARAMETER CONFIGURATION

Table 15 contains the total number of *trainable* model parameters of the top-performing configuration as determined by hyperparameter optimization. Due to OOM errors, GAT and SIGN+GAT experiments were not performed in ogbn-arXiv and ogbn-products. The total number reported for SIGN+GAT includes parameters in both the SIGN and GAT models.

Table 15: Total Number of Learnable Parameters. $*$ includes number of GAT parameters.

	Pubmed	*arXiv*	*products*
GAT	33,779	OMM	OOM
SIGN	2,613,768	531,885	14,124,085
SIGN+GAT	715,262*	OMM	OOM
SIGN+CS	1,570,825	10,947,629	8,422,454
SIGN+SHA	681,483	10,947,629	11,069,494
SIGN+MHA	4,576,266	1,270,317	17,379,382

Appendix H COMPLETE MICRO-AVERAGED F1 RESULTS

Micro-average F1 accuracy results for training, validation, and test splits for each model, per dataset are found in Tables 16- 18. Ten independent runs over a range of fixed global seed values were performed; models were trained over 300 epochs, and evaluation metrics were collected every tenth epoch. Metric averages and standard deviations were computed using data from the best epoch of each run, as defined by the highest validation F1 score [12].

Table 16: *Pubmed*: Micro-averaged F1 Training, Validation and Test Scores (%) (average ± standard deviation).

	Training	*Validation*	*Testing*
GAT	85.1 ± 0.1	88.6 ± 0.2	84.6 ± 0.3
SIGN	98.2 ± 1.5	92.7 ± 0.5	89.5 ± 0.6
SIGN+GAT	99.5 ± 0.9	89.2 ± 0.4	89.1 ± 0.1
SIGN+CS	97.5 ± 1.8	91.8 ± 0.5	90.1 ± 0.4
SIGN+SHA	98.8 ± 1.4	92.1 ± 0.4	90.1 ± 0.5
SIGN+MHA	98.0 ± 1.8	93.3 ± 0.2	90.4 ± 0.4

Table 17: *ogbn-arXiv*: Micro-averaged F1 Training, Validation and Test Scores (%) (average ± standard deviation).

	Training	*Validation*	*Testing*
SIGN	69.7 ± 0.5	61.0 ± 0.1	54.7 ± 0.4
SIGN+CS	80.8 ± 2.1	64.7 ± 0.1	59.6 ± 0.2
SIGN+SHA	77.9 ± 5.3	64.0 ± 0.3	59.1 ± 0.4
SIGN+MHA	74.5 ± 2.5	63.0 ± 0.1	57.1 ± 0.3

Table 18: *obgn-products*: Micro-averaged F1 Training, Validation and Test Scores (%) (average ± standard deviation).

	Training	*Validation*	*Testing*
SIGN	95.1 ± 0.1	92.5 ± 0.0	75.5 ± 0.5
SIGN+CS	97.0 ± 0.2	91.7 ± 0.0	76.3 ± 0.3
SIGN+SHA	98.6 ± 0.2	91.2 ± 0.1	77.3 ± 0.3
SIGN+MHA	98.5 ± 0.3	91.6 ± 0.0	76.3 ± 0.2