



Adaptive graph diffusion networks: compact and expressive GNNs with large receptive fields

Chuxiong Sun¹ · Muhan Zhang² · Jie Hu¹ · Hongming Gu¹ · Jinpeng Chen³ · Mingchuan Yang¹

Accepted: 7 January 2025 / Published online: 25 January 2025
© The Author(s) 2025

Abstract

Graph neural networks (GNNs) are widely used in graph-based tasks, but deep GNNs often suffer from oversmoothing. Existing effective deep GNNs have various shortcomings including redundant complexity, oversimplified architecture, or predefined parameters. To address these issues, we propose adaptive graph diffusion networks (AGDNs), a class of compact and expressive GNNs that can effectively leverage deep neighborhood information. We introduce hopwise attention and hopwise convolution with positional embeddings for learning nodewise and channelwise hop weights, respectively, which overcomes oversmoothing and ensures a powerful ability to learn arbitrary filters in the spectral domain. Our experiments demonstrate that AGDNs can effectively learn various filters on images and exhibit superior performance on diverse and challenging open graph benchmark datasets for node classification and link prediction tasks while maintaining moderate complexity and fast running time.

Keywords Graph neural networks · Graph diffusion · Attention · Learning filters · Semisupervised node classification · Link prediction

1 Introduction

Graph neural networks (GNNs) have achieved great success in modeling diverse graph data types, such as citation networks (Kipf and Welling 2017; Vaswani et al. 2017; Klicpera et al. 2019a), social networks (Hamilton et al. 2017; Chen et al. 2018; Rossi et al. 2020), and biological graphs (Fout et al. 2017; Li et al. 2019, 2018, 2021). Fundamental tasks addressed by GNNs include semisupervised node classification (Defferrard et al. 2016; Kipf and Welling 2017; Hamilton et al. 2017; Vaswani et al. 2017; Li et al. 2019, 2020; He et al. 2021; Li et al. 2021; Chen et al. 2020; Yu et al. 2022) and link prediction (Kipf and Welling 2016; Wang et al. 2021).

In semisupervised node classification, GNNs predict the class of unlabeled nodes by using a limited set of labeled nodes and neighborhood information. Link prediction, on the

other hand, is about inferring potential edges in incomplete graphs, which can also form a self-supervised learning paradigm when done with masked edges. Despite its potential, self-supervised graph learning (Kipf and Welling 2016; Pan et al. 2018; Hou et al. 2022) often falls behind supervised methods when labels are available.

GNNs often incorporate successful techniques from other areas. For instance, attention mechanisms from transformer models (Vaswani et al. 2017) are widely employed. Graph attention networks (GATs) (Velickovic et al. 2018) utilize attention scores to weigh the importance of neighbors. However, GAT still suffers from the oversmoothing issue (Li et al. 2018), a significant challenge to deep GNNs. Techniques from computer vision, such as convolution mechanisms (LeCun et al. 1998), have also been applied to GNNs for adaptive multihop feature aggregation (Zhang et al. 2018; Chien et al. 2021).

The oversmoothing problem, a common issue with deep GNNs, arises when multiple layers are stacked to capture complex neighborhood information, leading to indistinguishable node features. This issue is often attributed to the nature of GNNs, which tend to amplify components with low eigenvalues associated with more uniform graph signals. Consequently, as illustrated in Fig. 1, high-frequency components are attenuated, yielding a more uniform output. Over iterations, the lowest-frequency component increasingly dominates the output, causing oversmoothing.

To alleviate the oversmoothing issue pervasive in deep GNNs, various solutions have been proposed. However, these techniques often bring increased complexity, simplified architecture, or rely heavily on predefined parameters. Residual GNNs (Li et al. 2018, 2019, 2020; Chen et al. 2020; Li et al. 2021) incorporate residual connections, preserving shallow information in each layer to counteract oversmoothing. However, this approach results in the layer count increasing linearly with the receptive field radius, demanding a more flexible methodology to reduce redundancy and runtime.

Decoupled GNNs (Klicpera et al. 2019a; Rossi et al. 2020; Liu et al. 2020; Chien et al. 2021) employ a two-part architecture including message-passing and feature transformation to capture multihop information and to mitigate oversmoothing. However, the simplicity of this structure can limit the capacity of the model and its compatibility with other methods (Chen et al. 2020; Wang 2021; Li et al. 2021).

In contrast, diffusion-based GNNs (Defferrard et al. 2016; Klicpera et al. 2019b; Du et al. 2017; Wang et al. 2021; He et al. 2021; Bianchi et al. 2021) incorporate a graph diffusion step at each layer, which performs multiple message-passing steps and integrates intermediate results. The graph diffusion mentioned in this paper, directly modeled without probabilistic elements, differs from diffusion models in computer vision typically used for image noise reduction or smoothing. Despite having fewer layers, diffusion-based GNNs maintain deep receptive fields and offer better compatibility with many key GNN techniques (Chen et al. 2020; Wang 2021; Chien et al. 2022). These models may employ either explicit (Klicpera et al. 2019b) or implicit (Defferrard et al. 2016; Du et al. 2017; Wang et al. 2021; He et al. 2021) diffusion matrices, with the latter termed graph diffusion networks (GDNs). However, many GDNs utilize either predefined or naively learnable weights, hindering their ability to further capture various patterns within deep neighborhoods.

In this paper, we introduce adaptive graph diffusion networks (AGDNs), a compact and expressive class of GNNs with large receptive fields. We note that always acting as low-pass filters in the spectral domain can explain the oversmoothing of conventional GNNs. Thus, the novelty of AGDNs lies in the ability to learn arbitrary filters in the spectral domain,

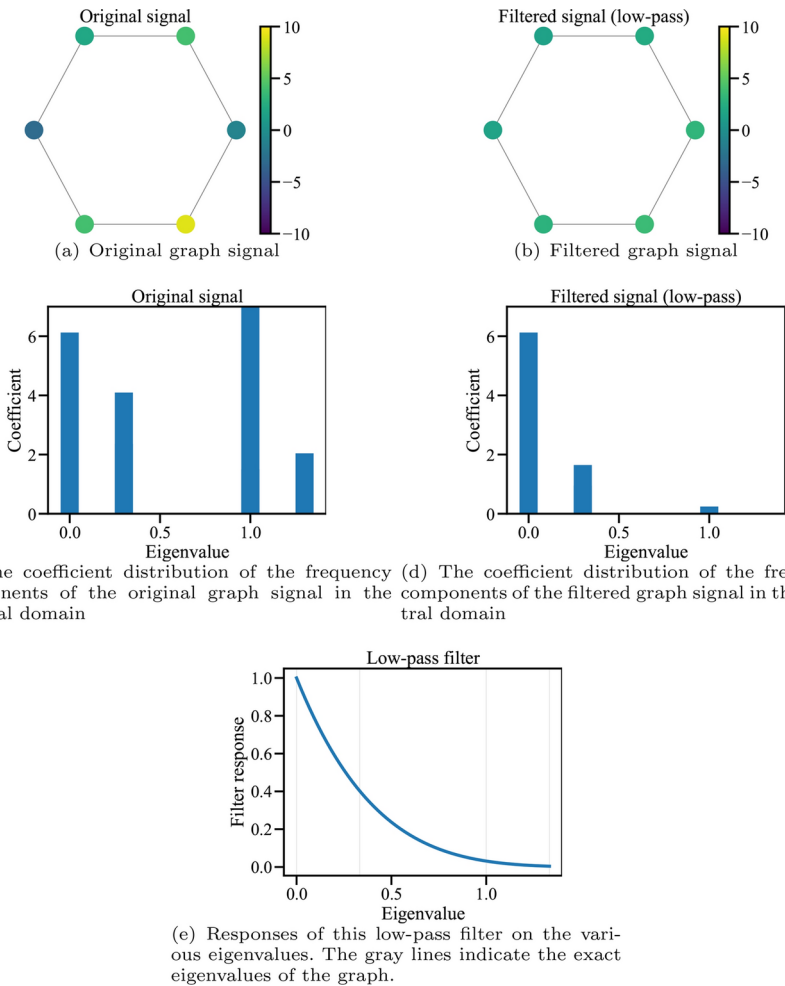


Fig. 1 Applying a low-pass filter ($g(\lambda) = (1 - 0.5\lambda)^5$) to a ring graph. Such a filter can be considered a deep 5-layer GNN. In the filtered graph signal, the ratio of high-frequency components becomes significantly lower

hence overcoming the oversmoothing issue and further improving expressiveness. To ensure this ability, with well-limited parameters, we introduce graph diffusion with hopwise attention (HA) and hopwise convolution (HC) to adaptively gather multihop information. We also incorporate positional embeddings (PEs) to enhance this ability for HA. We validate this ability with experiments of learning various filters on images. We also evaluate AGDNs on diverse and challenging open graph benchmark (OGB) (Hu et al. 2020) datasets under semisupervised node classification and link prediction tasks, and our results show that AGDNs outperform state-of-the-art (SOTA) GNNs while maintaining moderate complexity and runtime. The main contributions of this paper are as follows:

- Introducing AGDNs, a compact and expressive class of GNNs with large receptive

fields.

- Addressing the oversmoothing problem with HA and HC with PEs.
- Validating the theoretical ability of AGDNs with experiments on learning various filters.
- Outperforming SOTA GNNs on diverse OGB datasets while maintaining moderate complexity and runtime.

2 Preliminaries

In this section, we summarize the important symbols in Table 1, providing concise definitions to ensure clarity and consistency throughout the manuscript. We consider an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with the node set $\mathcal{V}$ and the edge set $\mathcal{E}$. We denote the number of nodes by $N = |\mathcal{V}|$ and the number of edges by $E = |\mathcal{E}|$. The adjacency matrix, $\mathbf{A} \in \mathbb{R}^{N \times N}$, is assumed to be non-negative and symmetric. Given the symmetry of $\mathbf{A}$, the degree matrix $\mathbf{D}$, representing both in-degrees and out-degrees, is computed by summing the elements of each row or column of the adjacency matrix.

The normalized adjacency matrix, or transition matrix, is denoted by $\overline{\mathbf{A}} \in \mathbb{R}^{N \times N}$. The graph convolution is typically represented by left-multiplication with the normalized adjacency matrix rather than the standard adjacency matrix. Normalization is achieved by scaling the matrix elements through division by the node degrees, in the form of multiplication with the inverse degree matrix $\mathbf{D}^{-1}$. Various types of normalization are summarized in Table 2. This normalization facilitates comparisons across graphs with varying sizes and degree distributions. Additionally, it ensures that the largest absolute eigenvalue of $\overline{\mathbf{A}}$ is equal to 1, preventing numerical expansion during iterative graph convolutions.

We denote the raw node feature matrix by $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N]^T \in \mathbb{R}^{N \times d^{(0)}}$, where $d^{(0)}$ is the node feature dimension and $\mathbf{x}_i \in \mathbb{R}^{d^{(0)} \times 1}$ is the feature vector of node i .

Table 1 Symbol definitions

Symbols	Definitions	Symbols	Definitions
$\mathcal{E}$	Edge set	L	Number of GNN layers
$\mathcal{G}$	Undirected graph	$f^{(l)}(\cdot, \cdot)$	l -th GNN layer
$\mathcal{V}$	Node set	$g^{(l)}(\cdot)$	l -th polynomial filter
E	Number of edges	$\mathbf{X}$	Node feature matrix
N	Number of nodes	$\mathbf{x}_i$	The feature vector of node i
$\mathbf{A}$	Adjacency matrix	$\mathbf{H}^{(l)}$	Node representation matrix at l -th layer
$\overline{\mathbf{A}}$	Normalized adjacency matrix	$\mathbf{h}_i^{(l)}$	The representation vector of node i at l -th layer
$\mathbf{L}$	Laplacian matrix	$\mathbf{D}$	Degree matrix
$\overline{\mathbf{L}}$	Normalized Laplacian matrix	$\mathbf{W}^{(l)}$	linear transformation matrix at l -th layer
$\mathbf{\Lambda}$	Eigenvalue matrix of $\overline{\mathbf{L}}$	$\sigma(\cdot)$	Nonlinear activation function
$\mathbf{U}$	Eigenvector matrix of $\overline{\mathbf{L}}$	$\sigma_{leak}(\cdot)$	LeakyReLU activation function
$\mathbf{u}_i$	i -th eigenvector	$\tilde{\mathbf{X}}$	Spectral graph signal
$d^{(0)}$	Node feature dimension	$\alpha_{i,c}$	Coefficient of i -th frequency component
$\mathcal{N}_i$	Neighborhood set of node i	$d^{(l)}$	l -th layer's output dimension
$[\cdot]^T$	Transpose operation	$\odot$	Hadamard product

Table 2 Transition matrices

Transition matrices	Symbols	Formulations
Row-normalized transition matrix	$\bar{\mathbf{A}}_{\text{row}}$	$\mathbf{D}^{-1} \mathbf{A}$
Column-normalized transition matrix	$\bar{\mathbf{A}}_{\text{col}}$	$\mathbf{A} \mathbf{D}^{-1}$
Symmetric normalized transition matrix	$\bar{\mathbf{A}}_{\text{sym}}$	$\mathbf{D}^{-1/2} \mathbf{A} \mathbf{D}^{-1/2}$
GCN transition matrix	$\bar{\mathbf{A}}_{\text{gcn}}$	$\hat{\mathbf{D}}^{-1/2} \hat{\mathbf{A}} \hat{\mathbf{D}}^{-1/2}$
GAT transition matrix	$\bar{\mathbf{A}}_{\text{gat}}$	$\mathbf{D}_{\text{gat.row}}^{-1} \mathbf{A}_{\text{gat}}$
Symmetric normalized GAT transition matrix	$\bar{\mathbf{A}}_{\text{gat.sym}}$	$\mathbf{D}_{\text{gat.row}}^{-\frac{1}{2}} \mathbf{A}_{\text{gat}} \mathbf{D}_{\text{gat.col}}^{-\frac{1}{2}}$

We denote the adjacency matrix with self-loops as $\hat{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ and the associated in(out)-degree matrix as $\hat{\mathbf{D}}$. We define the unnormalized GAT adjacency matrix that has the same positions of nonzero elements as $\mathbf{A}$ and for $(i, j) \in \mathcal{E}$, $(A_{\text{gat}})_{i,j} = \exp(\sigma_{\text{leak}}([\mathbf{W}^\top \mathbf{h}_i || \mathbf{W}^\top \mathbf{h}_j] \cdot \mathbf{a}))$. The diagonal entries of $\mathbf{D}_{\text{gat.row}}$ and $\mathbf{D}_{\text{gat.col}}$ are the row and column summations of $\mathbf{A}_{\text{gat}}$, respectively: $(\mathbf{D}_{\text{gat.row}})_{i,i} = \sum_{j=1}^N A_{\text{gat},i,j}$ and $(\mathbf{D}_{\text{gat.col}})_{i,i} = \sum_{j=1}^N A_{\text{gat},j,i}$.

We describe an L -layer common GNN model $f(\mathbf{X}, \mathbf{A})$ with a node feature matrix and adjacency matrix as input. This model stacks several layers $\mathbf{H}^{(l)} = g^{(l)}(\mathbf{H}^{(l-1)}, \mathbf{A}) \in \mathbb{R}^{N \times d^{(l)}}$ ($1 \leq l \leq L$) including nonlinear activations. We denote the intermediate node representation matrices by $\mathbf{H}^{(l)}$ ($0 \leq l \leq L$ and $\mathbf{H}^{(0)} = \mathbf{X}$). $\mathbf{H}^{(l)}$ can be represented with node representation vectors $\mathbf{H}^{(l)} = [\mathbf{h}_1^{(l)}, \mathbf{h}_2^{(l)}, \dots, \mathbf{h}_N^{(l)}]^\top$ with $\mathbf{h}_i^{(l)} \in \mathbb{R}^{N \times 1}$. The transition matrix is typically used in each layer and can be layerwise and learnable. For node classification, the softmax function is used at the output of $f(\mathbf{X}, \mathbf{A})$. We denote the nonlinear activation by σ and the LeakyReLU activation by σ_{leak} .

2.1 Preparation for spectral analysis

The Laplacian matrix, defined as $\mathbf{L} = \mathbf{D} - \mathbf{A}$, is fundamental in spectral analysis, capturing graph properties through its eigenvalues and eigenvectors. Moreover, the normalized version of the Laplacian matrix $\bar{\mathbf{L}}$ is more widely used but is not directly involved in graph convolutions, where repeated multiplications could lead to numerical expansion. Thus, the eigenvalues of the normalized Laplacian matrix do not need to be restricted to the $[-1, 1]$ range, as with the normalized adjacency matrix. Instead, they are confined to $[0, 2]$ (Chung 1996; Von Luxburg 2007), which still allows for effective comparisons across graphs with different sizes and degree distributions (Chung 1996). However, its computation follows that of the normalized adjacency matrix. The multiple normalization approaches, as outlined in Table 2, all result in the same form of normalized Laplacian matrix with the normalized adjacency matrix: $\bar{\mathbf{L}} = \mathbf{I} - \bar{\mathbf{A}}$, where the degree matrix $\mathbf{D}$ becomes the identity matrix. The associated eigendecomposition is $\bar{\mathbf{L}} = \mathbf{U} \mathbf{\Lambda} \mathbf{U}^\top$. Here, $\mathbf{U} = [\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_N]$ is the orthonormal eigenvector matrix and $\mathbf{\Lambda} = \text{diag}(\lambda_1, \lambda_2, \dots, \lambda_N)$ is the eigenvalue matrix. $\mathbf{u}_i$ is the i -th eigenvector of $\bar{\mathbf{L}}$, and $\lambda_i \in [0, 2]$ is the i -th eigenvalue of $\bar{\mathbf{L}}$. A graph signal $\mathbf{X} \in \mathbb{R}^{N \times d}$ can be transformed into the spectral domain by using $\tilde{\mathbf{X}} = \mathbf{U}^\top \mathbf{X}$ and recovered by using $\mathbf{X} = \mathbf{U} \tilde{\mathbf{X}}$. Diffusion-based GNNs and some decoupled GNNs can be viewed as polynomial filters in the spectral domain. For example, $f(\mathbf{X}; \bar{\mathbf{L}}) = \sum_{k=0}^K \theta_k \bar{\mathbf{L}}^k \mathbf{X}$ can be represented as $\mathbf{U} g(\mathbf{\Lambda}) \tilde{\mathbf{X}}$, where $g(\mathbf{\Lambda}) = \sum_{k=0}^K \theta_k \mathbf{\Lambda}^k$ and the coefficients θ_k can be predefined or learnable. Alternatively, another form of $g(\mathbf{\Lambda})$ is more typically used:

$g(\Lambda) = \sum_{k=0}^K \theta_k (\mathbf{I} - \Lambda)^k$, which matches the form of graph diffusion. The c -th channel has the graph signal $\mathbf{X}_{:,c}$ that can be decomposed into frequency components by using $\mathbf{X}_{:,c} = \sum_{i=1}^N \alpha_{i,c} \mathbf{u}_i$ with coefficients $\alpha_{i,c}$. In fact, for orthonormal bases $\mathbf{u}_i$, $\tilde{X}_{i,c} = \alpha_{i,c}$.

3 Related works

3.1 Attention mechanisms

The attention mechanism in transformers (Vaswani et al. 2017) computes the attention scores as the dot product of query and key vectors derived from input embeddings:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax} \left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d}} \right) \mathbf{V}, \quad (1)$$

where $\mathbf{K}^T$ denotes the transpose of the key matrix $\mathbf{K}$. In graph attention networks (GATs) (Velickovic et al. 2018), the attention score between a node i and its neighbor j is computed by performing a dot product between the concatenation of their transformed feature vectors and a query vector $\mathbf{a}$ and then by performing normalization among neighbors via softmax:

$$\bar{A}_{ij} = \frac{\exp(\sigma_{\text{leak}}([\mathbf{W}\mathbf{h}_i || \mathbf{W}\mathbf{h}_j] \cdot \mathbf{a}))}{\sum_{j \in \mathcal{N}_i} \exp(\sigma_{\text{leak}}([\mathbf{W}\mathbf{h}_i || \mathbf{W}\mathbf{h}_j] \cdot \mathbf{a}))}, \quad (2)$$

where $\cdot$ represents the dot product and the LeakyReLU activation σ_{leak} is applied to introduce nonlinearity in the attention mechanism.

Both mechanisms assign different weights to inputs based on their relevance in a given context. GAT is more flexible than is GCN, but in some cases, GAT acts like GCN (NT and Maehara 2019). GAT still suffers from the oversmoothing issue. Moreover, GAT utilizes only a breadthwise attention mechanism. In this paper, we propose HA as the depthwise attention mechanism that can weigh the importance of different hops and overcome the oversmoothing issue.

3.2 Diffusion-based GNNs

Let us suppose that $\mathcal{G}$ is an undirected graph with a normalized adjacency matrix (transition matrix) $\bar{\mathbf{A}}$ and node feature matrix $\mathbf{X}$. Graph diffusion (Klicpera et al. 2019a, b; Page et al. 1999; Kondor and Lafferty 2002) can be described as:

$$f(\mathbf{X}, \bar{\mathbf{A}}) = \sum_{k=0}^K \theta_k \bar{\mathbf{A}}^k \mathbf{X}, \quad (3)$$

with l_1 -normalized nonnegative weights θ_k and maximum diffusion depth K .

Diffusion-based GNNs enhance receptive fields by intensifying message-passing steps and by using fewer layers. The graph diffusion convolution (GDC) (Klicpera et al. 2019b) method computes the explicit diffusion matrix during preprocessing and the diffused feature

during training and inference. However, GDC faces limitations in scalability, flexibility, and link prediction performance. Graph diffusion networks (GDNs) address these limitations by using implicit, memory-efficient graph diffusion. They iteratively calculate multihop features and aggregate them during training and inference, resulting in smaller memory requirements.

Spectral GNNs, which include diffusion-based GNNs and some decoupled GNNs (Defferrard et al. 2016; Klicpera et al. 2019a; Chen et al. 2020), employ graph diffusion as polynomial filters in the spectral domain. With preparation in Subsection 2.1, we can give the form of graph diffusion in the spectral domain:

$$f(\mathbf{X}, \overline{\mathbf{A}}) = \mathbf{U}g(\mathbf{\Lambda})\widetilde{\mathbf{X}}, \quad (4)$$

$$g(\mathbf{\Lambda}) = \sum_{k=0}^K \theta_k (\mathbf{I} - \mathbf{\Lambda})^k, \quad (5)$$

Thus, graph diffusion can be interpreted as a polynomial filter in the spectral domain. For comparison, ignoring feature transformations, multiple GCN layers are a special case of graph diffusion when only the K -order coefficient θ_K is not zero and equal to 1. Due to the approximation power of polynomials, with a sufficiently large order K and suitable coefficients, graph diffusion can approximate any filters in the spectral domain. However, many existing GDNs, such as the topology adaptive graph convolutional network (TAGCN) (Du et al. 2017) and multihop attention graph neural network (MAGNA) (Wang et al. 2021), utilize predefined coefficients. ChebNet (Defferrard et al. 2016), ARMA (Bianchi et al. 2021) and BernNet (He et al. 2021) learn naive coefficients with complicated polynomial bases or forms. Our proposed AGDNs offer more flexible coefficients and can learn arbitrary filters even with the simplest polynomial bases, enhancing both node classification and link prediction performance.

3.3 Other deep GNNs

Residual GNNs such as the jumping-knowledge network (JKNet) (Xu et al. 2018), GCNII (Chen et al. 2020), DeepGCN (Li et al. 2019), DeeperGCN (Li et al. 2020), and RevGNNs (Li et al. 2021) preserve shallow information by using residual connections. DGCNN (Zhang et al. 2018) utilizes learnable convolution kernels to combine intermediate representations. Despite their excellent performance, these GNNs often involve large-scale architectures, leading to extended runtimes. Alternatively, our compact AGDNs with large receptive fields offer competitive performance with fewer parameters and less runtime. Residual connections can also be incorporated into AGDNs.

Decoupled GNNs such as DCNN (Defferrard et al. 2016), SGC (Wu et al. 2019), SIGN (Rossi et al. 2020), APPNP (Klicpera et al. 2019a), GPR-GNN (Chien et al. 2021), and DAGNN (Liu et al. 2020) achieve deep receptive fields with lower complexity by separating the message-passing and feature transformation steps. Despite addressing oversmoothing issues, their decoupled architecture limits model capacity and poses compatibility issues with certain GNN techniques. In contrast, AGDNs manage complexity effectively while

maintaining the stacked multilayer architecture characteristic of GNNs, thereby avoiding these limitations.

4 Proposed methods

In this section, we first introduce the framework of AGDNs by proposing a novel symmetric graph attention network (GAT) (Vaswani et al. 2017) transition matrix. Then, we propose two scalable mechanisms with positional embeddings for combining multihop information.

4.1 Adaptive graph diffusion networks

For the following discussion, we omit the condition that the last L -th layer does not include a nonlinear activation. A common L -layer GNN model follows stacking multilayer architecture: $f(\mathbf{X}, \mathbf{A}) = f^{(L)}(\dots f^{(2)}(f^{(1)}(\mathbf{X}, \mathbf{A}), \mathbf{A}), \mathbf{A}), \mathbf{A})$.

For an AGDN model, the l -th layer, which outputs the l -th node representation matrix $\mathbf{H}^{(l)}$, can be described as follows:

$$\mathbf{H}^{(l)} = f^{(l)}(\mathbf{H}^{(l-1)}, \mathbf{A}) = \sigma \left(\sum_{k=0}^K \Theta_{:,k,:}^{(l)} \odot (\bar{\mathbf{A}}^{(l),k} \mathbf{H}^{(l-1)} \mathbf{W}^{(l)}) \right), \quad (6)$$

where $\odot$ refers to the elementwise multiplication (Hadamard product). We also illustrate the overall architecture of an AGDN model in Fig. 2. To enhance the flexibility and expressiveness of GDNs, we introduce variable weights for different nodes or feature channels. These weights are represented by a 3-dimensional weighting tensor $\Theta^{(l)} \in \mathbb{R}^{N \times (K+1) \times d^{(l)}}$.

Instead of directly multiplying the representation matrix with a scalar $\theta_k^{(l)}$, which is the

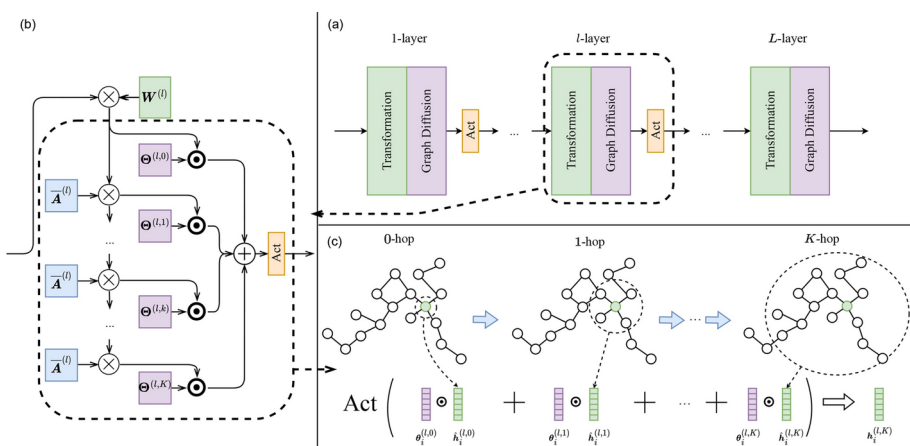


Fig. 2 Model architecture of an AGDN model. **a** We preserve the stacking architecture of common GNNs in AGDNs. **b** In each AGDN layer, we perform feature transformation and adaptive graph diffusion. **c** We utilize generalized weights for adaptive graph diffusion. The operator $\otimes$ represents matrix multiplication, the $\odot$ operator represents elementwise multiplication, and the $\oplus$ operator represents summation

associated element of the weighting vector $\theta^{(l)} \in \mathbb{R}^{K+1}$, we perform an elementwise multiplication between the representation matrix and a matrix $\Theta_{:,k,:}^{(l)} \in \mathbb{R}^{N \times d^{(l)}}$, which is the associated slice of the weighting tensor $\Theta^{(l)}$. A larger element $\Theta_{i,k,c}^{(l)}$ indicates that the k -th hop representation has a greater contribution to the output representation for node i and the c -th channel. GDNs are a special case of AGDNs when all elements in $\Theta_{:,k,:}^{(l)}$ are the same. GNNs following the MPNN framework are a special case of AGDNs if for all layer l , all elements in $\Theta_{:,k,:}^{(l)}$ are 1 if $k = 1$ and 0 if $k \neq 1$.

We omit the computation of transition matrix $\bar{A}^{(l)}$ in Eq. 6. The superscript (l) is necessary if we utilize the GAT transition matrix and its variant. We summarize the common transition matrices in Table 2. The transition matrix used in graph diffusion is the normalized adjacency matrix, which is also used in graph convolution. Since the symmetric GCN (Kipf and Welling 2017) transition matrix has been shown to be more effective on specific datasets, we propose a symmetric variant of the GAT (Vaswani et al. 2017) transition matrix to improve performance on these datasets, denoted as $\bar{A}_{\text{gat.sym}}$ in Table 2.

4.2 Learning the weighting tensor

The critical issue is to obtain suitable weights. To simplify the discussion, we reformulate the l -th AGDN layer, previously described in Eq. 6, from both channel and node viewpoint.

First, associated with the right-hand side of $\odot$ in Eq. 6, we iteratively compute the multihop node representation vectors for node i at the l -th layer as follows:

$$\hat{h}_{i,c}^{(l,k)} = \begin{cases} h_i^{(l-1,0)} \mathbf{W}^{(l)}, & \text{if } k = 0; \\ \bar{A}^{(l)} \hat{h}_j^{(l,k-1)}, & \text{if } 0 < k \leq K, \end{cases} \quad (7)$$

where the initial 0-th hop representation vector, $h_i^{(l,0)}$, is generated by applying the l -th layer's linear transformation matrix, $\mathbf{W}^{(l)}$.

Then, associated with the elementwise multiplication and summation in Eq. 6, we aggregate the multihop node representations $\hat{h}_i^{(l,k)}$ with nodewise and channelwise weights $\Theta_{i,k,c}^{(l)}$ from the weighting tensor $\Theta^{(l)}$:

$$h_{i,c}^{(l)} = \sigma \left(\sum_{k=0}^K \Theta_{i,k,c}^{(l)} \hat{h}_{i,c}^{(l,k)} \right), \quad (8)$$

where $h_{i,c}^{(l)}$, associated with node i and c -th channel, corresponds to the (i, c) entry of the node representation matrix at the l -th layer $\mathbf{H}^{(l)}$ defined in Eq. 6.

Learning an adaptive weighting tensor directly can be challenging, as it leads to a significant increase in the number of parameters. To address this issue, we introduce hopwise convolution (HC) and hopwise attention (HA) as methods to obtain an adaptive weighting tensor with a reasonable number of parameters. By applying different weights, we obtain three

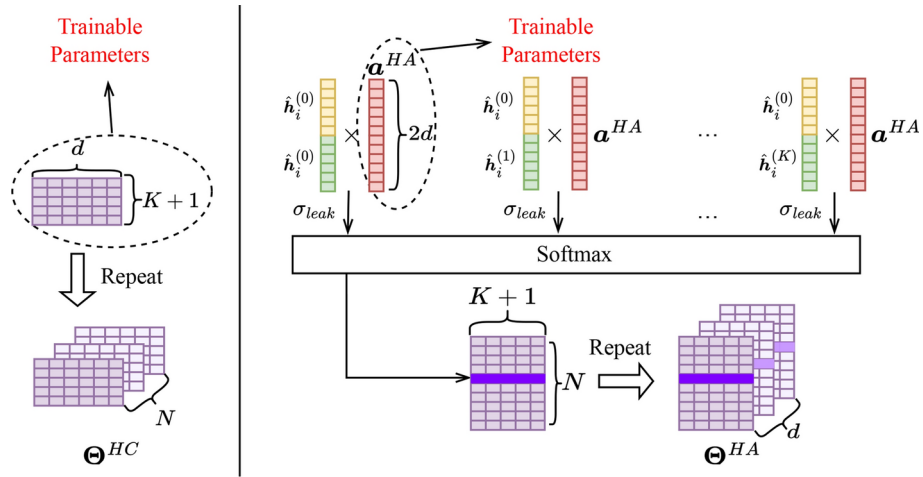


Fig. 3 HA and HC. We omit the superscript (l). HC (Left): The weighting tensor of hopwise convolution Θ is directly derived from $(K + 1) \times d$ kernel matrix Θ^{HC} by repeating N times along the first dimension. HA (Right): The hopwise attention is parameterized by $\mathbf{a}^{\text{HA}}$ with a LeakyReLU activation function σ_{leak} and normalized along hops with the softmax function. The associated weighting tensor Θ can be derived from the $N \times (K + 1)$ matrix Θ^{HA} by repeating d times along the third dimension

Table 3 Complexity comparison

Complexity	GNNs	AGDNs
Time	$O(LKEd + LKNd^2)$	$O(LKEd + LNd^2)$
Space	$O(LKNd)$	$O(LKNd)$

variants of AGDN: AGDN-mean, AGDN-HA, and AGDN-HC. AGDN-mean utilizes uniform weights and can be viewed as a GDN.

4.2.1 Hopwise convolution

We propose a simple mechanism called hopwise convolution (HC) that utilizes only $(K + 1) \times d^{(l)}$ parameters. The channelwise filters in the spectral domain are important to produce multidimensional predictions (Wang and Zhang 2022). Thus, we define a learnable weighting tensor with the constraint $\Theta_{i,k,c}^{(l)} = \Theta_{1,k,c}^{(l)}$ for all i . However, to minimize the number of parameters, we assume that all nodes share the same weights. This allows us to simplify the weighting tensor into a 2-dimensional matrix $\Theta^{\text{HC},(l)} \in \mathbb{R}^{(K+1) \times d^{(l)}}$ by ignoring the first subscript i . We can recover the complete weighting tensor $\Theta^{(l)}$ by adding the first dimension and by repeating $\Theta^{\text{HC},(l)}$ N times in the first dimension, as illustrated in the right part of Fig. 3. In short, the output of the AGDN-HC can be formulated as follows:

$$h_{i,c}^{(l)} = \sigma \left(\sum_{k=0}^K \Theta_{k,c}^{\text{HC},(l)} \hat{h}_{i,c}^{(l,k)} \right), \quad (9)$$

where we change Eq. 8 by replacing $\Theta_{i,k,c}$ with the elements $[\Theta_{k,c}^{HC,(l)}]_{0 \leq k \leq K, 1 \leq c \leq d^{(l)}}$ of the HC kernel $\Theta^{HC,(l)}$. Consequently, in Eq. 9, the multihop representation vectors' c -th channels $[\hat{h}_{i,c}^{(l,k)}]_{0 \leq k \leq K}$ are aggregated with learnable channelwise weights $[\Theta_{k,c}^{HC,(l)}]_{0 \leq k \leq K}$

, which are directly optimized during training.

Moreover, similar to CNNs (Krizhevsky et al. 2012), HC has different kernels in different channels. However, HC does not slide along hops and only computes a single output combining all hops.

4.2.2 Hopwise attention

HC still directly learns hop weights, which may be sensitive to initialization and difficult to optimize. Thus, inspired by the attention mechanism in GAT (Vaswani et al. 2017), we propose hopwise attention (HA) to induce hop weights with $2d^{(l)}$ parameters from a learnable query vector $\mathbf{a}^{HA,(l)} \in \mathbb{R}^{2d^{(l)}}$. We suppose that hop weights should be node-specific and normalized: $\sum_{k=0}^K \Theta_{i,k,c}^{(l)} = 1, \forall i, c$. For convenience, we ignore the differences along channels: $\Theta_{i,k,c}^{(l)} = \Theta_{i,k,1}^{(l)}, \forall c$. We simplify the unified weighting tensor into a 2-dimensional weighting matrix $\Theta^{HA,(l)} \in \mathbb{R}^{N \times (K+1)}$, by ignoring the last subscript c . $\Theta^{(l)}$ can be recovered by adding the third dimension and repeating $\Theta^{HA,(l)}$ for $d^{(l)}$ times in the third dimension, as shown in the left part of Fig. 3.

The HA weighting tensor is calculated as follows:

$$\Theta_{i,k}^{HA,(l)} = \frac{\exp \left(\sigma_{leak} \left(\left[\hat{\mathbf{h}}_i^{(l,0)} \parallel \hat{\mathbf{h}}_i^{(l,k)} \right] \cdot \mathbf{a}^{HA,(l)} \right) \right)}{\sum_{k=0}^K \exp \left(\sigma_{leak} \left(\left[\hat{\mathbf{h}}_i^{(l,0)} \parallel \hat{\mathbf{h}}_i^{(l,k)} \right] \cdot \mathbf{a}^{HA,(l)} \right) \right)}, \quad (10)$$

where $\cdot$ represents the inner product, $\parallel$ represents the concatenation operation, k represents the k -th hop's representation and $\mathbf{a}^{HA,(l)}$ represents the query vector of the HA. Finally, we formulate the output of AGDN-HA as follows:

$$h_{i,c}^{(l)} = \sigma \left(\sum_{k=0}^K \Theta_{i,k}^{HA,(l)} \hat{h}_{i,c}^{(l,k)} \right). \quad (11)$$

The form of HA is similar to GAT (Vaswani et al. 2017). However, there are significant differences between them. First, the GAT calculates the edge scores used in the message-passing step, while the HA calculates the weights used in combining multihop information in the graph diffusion step. Second, the edge scores of the GAT are normalized among the 1st hop neighborhood of the destination node, while the weights of the HA are normalized among the multihop smoothed representations of the target node. In short, HA is depthwise, while GAT is breadthwise.

Positional embedding

To enhance the position (hop) information, we can add multihop representation vectors with learnable positional embeddings (PEs) $\mathbf{p}^{(k)}, k \in \{0, 1, \dots, K\}$ in $\mathbb{R}^{d^{(l)} \times 1}$. Such PEs

are identical for all nodes. PEs can encode additional hop positional information and theoretically allow AGDN with HA to achieve better universality and be able to learn arbitrary filters. We discuss the role of PEs in Subsect. 5.2.

5 Model analysis

In this section, we begin by discussing the expressiveness of AGDN-HC and AGDN-HA. We provide simple spectral properties of AGDN-HC, and then, we present the universality theorem of AGDN-HA from a spatial perspective. We explain why stacking multilayer architecture is necessary for better capturing diverse information. Finally, we provide the time and space complexity of AGDNs.

5.1 Spectral properties of AGDN-HC

To simplify our analysis, we first consider the constrained condition where all nodes are assigned the same hop weights. This allows us to establish a lower bound on the expressiveness of AGDNs in general. First, we only consider 1-dimensional output. An AGDN layer applied on a graph signal $\mathbf{X} \in \mathbb{R}^{N \times d}$ is described as follows:

$$\begin{aligned} \sum_{k=0}^K \Theta_k^{\text{HC}} \mathbf{A}^k \mathbf{X} &= \mathbf{U} \left(\sum_{k=0}^K \Theta_k^{\text{HC}} (\mathbf{I} - \mathbf{\Lambda})^k \right) \mathbf{U}^\top \mathbf{X} \\ &= \mathbf{U} g(\mathbf{\Lambda}) (\mathbf{U}^\top \mathbf{X}), \end{aligned} \quad (12)$$

where $g: [0, 2] \rightarrow \mathbb{R}$ denotes a spectral filter in the spectral domain and $g(\mathbf{\Lambda})$ applies

$g(\lambda) = \sum_{k=0}^K \theta_k (1 - \lambda)^k$ elementwise to the diagonal entries of $\mathbf{\Lambda}$. The l -th AGDN layer acts as a polynomial filter $g^{(l)}$ in the spectral domain, with coefficients given by the hop weights. Ignoring the differences in transition matrices, nonlinear activations, and feature transformations, the overall AGDN model is a nested polynomial filter $g(\lambda) = g^{(L)}(\dots g^{(2)}(g^{(1)}(\lambda)) \dots)$. For the multidimensional output of the channelwise AGDN-HC, we can add the subscript c for θ and g in Eq. 12.

Theorem 1 (Weierstrass Approximation Theorem). We suppose h is a continuous real-valued function defined on the real interval $[a, b]$. For every $\epsilon > 0$, there exists a polynomial p such that all x in $[a, b]$, we have $|f(x) - p(x)| < \epsilon$.

Lemma 5.1 For a connected graph $\mathcal{G}$ with Laplacian matrix $\bar{\mathbf{L}}$ whose eigenvalues are $0 = \lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_N \leq 2$, ignoring feature transformation and nonlinear activation, an AGDN-HC layer f can be viewed as a polynomial filter with learnable coefficients $g(\lambda) = \sum_{k=0}^K \Theta_k^{\text{HC}} (1 - \lambda)^k$ in the spectral domain. We suppose p is a polynomial defined on $[0, 2]$. For every $\epsilon > 0$, there exists an AGDN-HC layer f acting as a polynomial filter g such that all λ in $[0, 2]$, and we have $|g(\lambda) - p(\lambda)| < \epsilon$.

Based on Theorem 1 and Lemma 5.1, we can easily obtain the following theorem:

Theorem 2 For a connected graph $\mathcal{G}$ with Laplacian matrix $\bar{\mathbf{L}}$ whose eigenvalues are $0 = \lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_N \leq 2$, ignoring feature transformation and nonlinear activation, an AGDN-HC layer f can approximate arbitrary continuous real-valued function h defined on the real interval $[0, 2]$ with sufficiently large K .

However, as discussed in (Wang and Zhang 2022), to generate arbitrary output, we have to add feature transformation and consider the characteristics of eigenvalues and the given input graph signal. The sufficient conditions can be expressed in this universality theorem:

Theorem 3 (Wang and Zhang 2022). An AGDN-HC layer with a feature transformation or other spectral GNNs that can approximate arbitrary spectral filters can produce any 1-dimensional prediction if $\bar{\mathbf{L}}$ has no multiple eigenvalues and the node feature $\mathbf{X} \in \mathbb{R}^{N \times d}$ contains all frequency components ($\alpha_i > 0 \forall i \in [1, 2, \dots, N]$).

The two conditions can be easily satisfied or approximately satisfied in real-world datasets. The channelwise filters can further ensure the production of arbitrary multidimensional predictions.

The above theorems are also applicable to several spectral GNNs (Defferrard et al. 2016; Chien et al. 2021; Bianchi et al. 2021; He et al. 2021; Wang and Zhang 2022), which can be regarded as polynomial filters with learnable coefficients. However, in many cases, it can be challenging to directly learn coefficients and to achieve optimal performance. Some spectral GNNs (Defferrard et al. 2016; He et al. 2021; Wang and Zhang 2022) utilize special polynomial bases to facilitate optimization. In this paper, we propose stacking multiple AGDN-HC layers and using channelwise filters to increase the ability to learn arbitrary filters. The channelwise filters are crucial when separating and preserving different frequency components of the same input graph signal. We demonstrate that even using the simplest form, the AGDN-HC can still significantly outperform most other spectral GNNs in learning all example filters.

5.2 AGDN-HA

AGDN-HA uses nonnegative and normalized hop weights, while AGDN-HC uses arbitrary hop weights. HA provides effective control over the magnitude of combined diffusion results, making it less sensitive to initialization. Additionally, HA is based on input representations and has better generalizability. The nodewise weights are critical for the AGDN-HA to learn various filters. First, we still consider the condition that all nodes share the same HA weights. Then, we can obtain exactly the same form as Eq. 12 but with different constraints on hop weights: $\Theta_k^{\text{HA}} \geq 0$ and $\sum_{k=0}^K \Theta_k^{\text{HA}} = 1$. With these constraints, AGDN-HA can learn only low-pass filters.

Theorem 4 (Chien et al. 2021) For a connected graph $\mathcal{G}$ with Laplacian matrix $\bar{\mathbf{L}}$ whose eigenvalues are $0 = \lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_N \leq 2$, ignoring feature transformation and nonlinear activation, a constrained AGDN-HA layer f can be viewed as a polynomial filter $g(\lambda) = \sum_{k=0}^K \Theta_k^{\text{HA}} (1 - \lambda)^k$ with nonnegative and l_1 -normalized learnable coefficients: $\theta_k \geq 0 \forall k \in \{0, 1, \dots, K\}$, $\sum_{k=0}^K \Theta_k^{\text{HA}} = 1$ in the spectral domain. If $\exists k' > 0$ such that $\Theta_{k'}^{\text{HA}} > 0$, g can only learn a low-pass filter and $|g(\lambda_1)| > |g(\lambda_i)|, \forall i \geq 2$.

The proof can be found in (Chien et al. 2021).

In AGDN-HA, the use of nodewise weights is crucial in overcoming the limitation discussed in Theorem 4. However, since the nodewise weights can easily change eigenvectors, the spectral analysis for AGDN-HA is very difficult. Thus, we directly obtain AGDN-HA's universality for multidimensional cases from a spatial perspective by using a strong condition, which includes the ability to learn arbitrary filters. To clarify, by using the node viewpoint form in Eqs. 8 and 11, we define the multihop node feature vectors for node i as $\hat{\mathbf{x}}_i^{(k)} = \sum_{j \in \mathcal{N}_i} \bar{A}_{i,j} \hat{\mathbf{x}}_j^{(k-1)}$ in $\mathbb{R}^{d \times 1}$ ($\hat{\mathbf{x}}_i^{(0)} = \mathbf{x}_i$). We find that for each node i , all possible values of its associated output vector $\bar{\mathbf{x}}_i = \sum_{k=0}^K \Theta_{i,k}^{\text{HA}} \hat{\mathbf{x}}_i^{(k)}$ constitute the convex hull Q_i of its multihop feature vectors $\{\hat{\mathbf{x}}_i^{(0)}, \hat{\mathbf{x}}_i^{(1)}, \dots, \hat{\mathbf{x}}_i^{(K)}\}$, since $\forall i, \Theta_{i,k}^{\text{HA}} \geq 0$ and $\sum_{k=0}^K \Theta_{i,k}^{\text{HA}} = 1$. As HA weights are node specific, all possible values of the set $\{\bar{\mathbf{x}}_1, \bar{\mathbf{x}}_2, \dots, \bar{\mathbf{x}}_N\}$ constitute the set $Q_1 \times Q_2 \cdots \times Q_N$.

Proposition 5 An AGDN-HA layer with a learnable scaling factor w can produce any d -dimensional prediction $\mathbf{Z} \in \mathbb{R}^{N \times d}$ if for every node i , there exists a d -dimensional ball centered at the origin belonging to the convex hull of its multihop feature vectors given a d -dimensional node feature input $\mathbf{X} \in \mathbb{R}^{N \times d}$.

Proof With learnable w , for each node i , its output vector is defined as $\mathbf{x}'_i = w\bar{\mathbf{x}}_i^{(k)} = w \sum_{k=0}^K \Theta_{i,k}^{\text{HA}} \hat{\mathbf{x}}_i^{(k)}$. The target vector for node i is denoted by $\mathbf{z}_i \in \mathbb{R}^{d \times 1}$ and $\mathbf{Z} = [\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_N]^\top$. The condition that there exists a d -dimensional ball centered at the origin belonging to Q_i is equivalent to $\forall i, \exists \epsilon_i > 0, \{\bar{\mathbf{x}}_i \mid \|\bar{\mathbf{x}}_i\|_2 < \epsilon_i\} \subset Q_i$. The theorem becomes proving that $\forall \mathbf{Z} \in \mathbb{R}^{N \times d}, \exists \{\bar{\mathbf{x}}_1, \bar{\mathbf{x}}_2, \dots, \bar{\mathbf{x}}_N\} \in Q_1 \times Q_2 \cdots \times Q_N$ such that $\forall i, w\bar{\mathbf{x}}_i = \mathbf{z}_i$. For any target prediction $\mathbf{Z}$, by setting $w > \sup_i \frac{\|\mathbf{z}_i\|_2}{\epsilon_i}$, we can simply construct $\bar{\mathbf{x}}_i = \frac{1}{w} \mathbf{z}_i$ belonging to Q_i because $\|\bar{\mathbf{x}}_i\|_2 = \frac{1}{w} \|\mathbf{z}_i\|_2 < \epsilon_i$. In short, by setting a sufficiently large scaling factor w , we can always find a feasible set $\{\bar{\mathbf{x}}_1, \bar{\mathbf{x}}_2, \dots, \bar{\mathbf{x}}_N\}$ for producing any target prediction. Therefore, the proof is complete. $\square$

In practical datasets, it may be difficult to satisfy the strong condition of this theorem. Since we need to construct a d -dimensional convex hull, we should set $K \geq d$ and ensure that for every node i , its multihop matrix $[\hat{\mathbf{x}}_i^{(0)}, \hat{\mathbf{x}}_i^{(1)}, \dots, \hat{\mathbf{x}}_i^{(K)}]$ has a column rank equal to d . In many cases, with d that is too large, the oversmoothing problem causes high hop vectors to tend to be identical; thus, the rank is typically not d . We incorporate learnable multihop embeddings (PEs) $\mathbf{p}^{(k)}$ in $\mathbb{R}^{d \times 1}$ to solve this problem.

Corollary 6 By incorporating learnable PEs $\mathbf{p}^{(k)}$ and a learnable scaling factor w , the AGDN-HA layer can produce any d -dimensional prediction $\mathbf{Z}$ given any d -dimensional graph signal $\mathbf{X}$.

Proof We denote the output by $\mathbf{x}'_i = w\bar{\mathbf{x}}_i + \sum_{k=0}^K \Theta_{i,k}^{\text{HA}} \mathbf{p}^{(k)}$. To demonstrate this corollary, we consider a special case where w is zero, i.e., only the linear combination of PEs is considered. For each node i , the set of all possible outputs $\mathbf{z}'_i$ is exactly the convex hull P of $\{\mathbf{p}^{(0)}, \mathbf{p}^{(1)}, \dots, \mathbf{p}^{(K)}\}$. We can construct $\mathbf{p}^{(k)}$ satisfying the condition of Proposition 5. For example, we can set $K = d$ and let $\mathbf{p}^{(k)}$ contain d standard basis vectors and a single vector $[-1, -1, \dots, -1]^\top$. We ensure that P is d -dimensional and contains the origin (by setting

$\Theta_{i,k}^{\text{HA}} = \frac{1}{d+1}$); thus, there exists a d -dimensional ball centered at the origin that is a subset of P . We denote the radius of this ball as ϵ . Then, we enlarge this radius to $w\epsilon$ by multiplying an additional scaling factor $w' > \frac{\sup_i \|z_i\|_2}{\epsilon}$ to every $\mathbf{p}^{(k)}$. For all i , we always have $\mathbf{z}_i \in P$ since $\sup_i \|\mathbf{z}_i\|_2 < w'\epsilon$, i.e., for any target prediction $\mathbf{Z}$ and every i , $\exists \Theta^{\text{HA}} \in \mathbb{R}^{N \times (K+1)}$ such that $\sum_{k=0}^K \Theta_{i,k}^{\text{HA}} \mathbf{p}^{(k)} = \mathbf{z}_i$. Hence, the proof is completed. $\square$

It is worth mentioning that the value of K in the AGDN-HA layer may be significantly lower than d since the dimension of the smallest convex subspace that encompasses all target vectors can be small, and complete universality may not always be needed. Additionally, in actual cases, the first term $w\bar{\mathbf{x}}_i$ involving input features is not zero, and the second term $\sum_{k=0}^K \Theta_{i,k}^{\text{HA}} \mathbf{p}^{(k)}$ can offer complementary capability for approximating information that cannot be approximated solely by the first term. In addition, we also conduct an empirical analysis of the HA weights in the Appendix.

5.3 Stacking multilayer architecture

The stacking multilayer architecture with intermediate feature transformations and nonlinear activations is a critical aspect of AGDNs, as it improves model capacity and maintains compatibility with other important techniques. From a spectral perspective, the multilayer architecture is essential for leveraging variable filters. We assume that variable filters such as low-pass, high-pass, or other filters within the same model are crucial for capturing diverse information.

First, feature transformations between graph diffusion operators can enhance the quality of graph signals. By Theorem 3, we can ensure the production of arbitrary output by excluding the multiple eigenvalues and missing frequency components in the node features. Specifically, given a 1-dimensional graph signal $\mathbf{X} = \sum_{i=1}^N \alpha_i \mathbf{u}_i$ and a graph filter $g(\lambda)$, the output becomes $\sum_{i=1}^N g(\lambda_i) \alpha_i \mathbf{u}_i$. Consequently, if $\alpha_i = 0$, then the output signal always misses the i -th component. Feature transformation, also known as channel mixing, can alleviate this issue by mixing graph signals from different channels.

However, if all graph signals have similar frequency components, linearly mixing them cannot add other frequency components. In such cases, nonlinear activations can be beneficial. For a graph signal $\mathbf{X}$ and its spectral signal $\tilde{\mathbf{X}}$, we apply a graph filter g with nonlinear activation σ : $\mathbf{U}g(\mathbf{\Lambda})\mathbf{U}^\top \sigma(\mathbf{X}) = \mathbf{U}g(\mathbf{\Lambda})\mathbf{U}^\top \sigma(\mathbf{U}\tilde{\mathbf{X}})$. As explained in (Wang and Zhang 2022), we can define the activation function in the spectral domain as $\sigma'(\tilde{\mathbf{X}}) = \mathbf{U}^\top \sigma(\mathbf{U}\tilde{\mathbf{X}})$. For each column of $\tilde{\mathbf{X}}$, the frequency components from different rows are first linearly mixed with the eigenvector basis $\mathbf{U}$, then elementwise applied with σ , and finally transformed back with $\mathbf{U}^\top$. The nonlinear operation σ ensures that linearly mixed components become nonlinearly mixed and cannot be recovered to original separated ones by left-multiplying with the transposed orthogonal eigenvector matrix $\mathbf{U}^\top$, thereby allowing σ' to easily change elements of $\tilde{\mathbf{X}}$.

In actual datasets, raw node features may rarely have missing frequency components. However, after a GNN layer or graph filter, the output graph signal can easily tend to contain certain frequency components. For instance, a low-pass filter emphasizes the low-frequency components, so applying another high-pass filter to the output of this filter does not yield

enough valuable information. With feature transformations and nonlinear activations in the stacking multilayer architecture, we can mitigate this problem.

5.4 Complexity analysis

The time complexity of an L -layer AGDN model is $O(LKEd + LKNd + LNd^2)$, but it is simplified to $O(LKEd + LNd^2)$ under the assumptions of $N \ll E$ and $K \ll d$. The adaptive diffusion mechanism in AGDN layers increases their time complexity compared to common GNN layers due to K -hop aggregations, elementwise multiplications, and optional HA computation. However, this mechanism significantly enhances the expressiveness of each AGDN layer. As a result, a compact AGDN model with fewer layers can match or surpass the performance of typical GNN models, where fewer layers counteract the increased complexity per layer. Furthermore, AGDNs are orthogonal to memory-saving techniques (Li et al. 2018; Zeng et al. 2020; Li et al. 2021) for GNNs, making them a promising approach for large-scale graph learning tasks.

6 Experiments

In this section, we present the evaluation of AGDNs on various datasets, involving tasks such as learning arbitrary filters, node classification, and link prediction. For the OGB datasets, we utilize their official data splits and metrics, which have been carefully selected in the paper. For the heterophily-prone datasets, we perform a random split of the data into training, validation, and test sets according to fixed ratios in each run. We also conducted additional experiments with complementary metrics. Table 4 provides a comprehensive summary of these datasets. We conducted all experiments on a single Nvidia Tesla V100 with 16 GB GPU memory. To address the memory constraints, we employed graph-based sampling techniques such as random partition (Li et al. 2020; Shi et al. 2021) for ogbn-proteins and ogbn-products and GraphSAINT (Zeng et al. 2020) for ogbl-citation2. For all except ogbn-products and ogbl-citation2 (which were evaluated on CPU), we performed both training and inference of AGDNs on the same GPU card. We conducted 10 runs of each proposed model for the last two tasks, fixing random seed 0–9, and reported means. In comparison tables, we highlighted the best results with *bold fonts*. The unavailable results were indicated by –, as some baselines had not been implemented or reported for certain datasets. We listed the critical hyperparameters in Table 5.

6.1 Task 1: learning filters

We conducted an experiment on 50 real images from the image processing toolbox in MATLAB with a resolution of 100×100 following the approach of (He et al. 2021). We treated each image as a 2D regular 4-neighborhood grid graph with a related adjacency matrix of 10000×10000 and a node feature vector of 1000×1 . The task was to learn 5 different filters (low-pass, high-pass, bandpass, band-rejection, and comb) by applying models with graph signals as input. The formulas for the filters are shown in Table 6. The loss function used was the sum of squared error between the output of the models and the filtering signal from the target filter, which made this a simple regression task. We utilized six different

Table 4 Dataset statistics

Datasets	#Nodes	#Edges	Metrics
Learning filters	10,000	39,600	Sum of squared error & R^2
ogbn-arxiv	169,343	1,166,243	Accuracy
ogbn-proteins	132,534	39,561,252	ROC-AUC
ogbn-products	2,449,029	61,859,140	Accuracy
Chameleon	2277	31,371	Accuracy
Squirrel	5201	198,353	Accuracy
Actor	7600	26,659	Accuracy
ogbl-ppa	576,289	30,326,273	Hits@100
ogbl-ddi	4267	1,334,889	Hits@20
ogbl-citation2	2,927,963	30,561,187	MRR

Table 5 Hyperparameters of AGDNs. K =diffusion depth for each layer, L =number of layers, d =hidden dimension, $heads$ =number of multiple heads, $\overline{A}$ = transition matrix, Θ = weighting tensor style (HA or HC)

Dataset	K	L	d	$heads$	$\overline{A}$	Θ	Sampling
Learning filters	5	2	8	4, 8	$\overline{A}_{\text{gcn}}$	HA/HC	None
ogbn-arxiv	3	2	256	3	$\overline{A}_{\text{gat.sym}}$	HA	None
ogbn-proteins	2	6	150	6	$\overline{A}_{\text{gat}}$	HC	Random partition
ogbn-products	3	4	120	4	$\overline{A}_{\text{gat}}$	HA	Random partition
Chameleon	3	3	1024	1	$\overline{A}_{\text{gat}}$	HA	None
Squirrel	3	3	1024	1	$\overline{A}_{\text{gat}}$	HA	None
Actor	3	3	1024	1	$\overline{A}_{\text{gat}}$	HA	None
ogbl-ppa	2	2	128	1	$\overline{A}_{\text{gat}}$	HA	None
ogbl-ddi	3	2	512	1	$\overline{A}_{\text{gat}}$	HA	None
ogbl-citation2	2	3	256	1	$\overline{A}_{\text{gat}}$	HA	GraphSAINT

baseline models: GCN (Kipf and Welling 2017), GAT (Vaswani et al. 2017), ARMA (Bianchi et al. 2021), ChebyNet (Defferrard et al. 2016), GPR-GNN (Chien et al. 2021), BernNet (He et al. 2021), and JacobiConv (Wang and Zhang 2022). The reported results of the baselines were taken from (He et al. 2021) except for JacobiConv since its paper does not provide R^2 scores. We re-evaluate JacobiConv with R^2 scores reported. To implement the AGDN, we designed a 2-layer architecture with 8 hidden dimensions (4 heads) in the first layer and 8 hidden dimensions (8 heads) in the second layer, following the setting of the GAT in (He et al. 2021). We used a linear layer at the end of the model to output predicted values and an ELU as an intermediate nonlinear activation function. We scaled up the scores and removed LeakyReLU before softmax to enhance the imbalance in HA weights, which empirically improved performance. To ensure a fair comparison, we used the default hyperparameters from (He et al. 2021), except for the early stopping parameter: learning rate 0.01, 2000 epochs, and early stopping 300. We utilize PEs in the AGDN. We did not use regularization techniques or tune the hyperparameters with tools, unlike (Wang and Zhang 2022).

Table 6 shows the average sum of squared error and R^2 scores. As expected in (Balcilar et al. 2021), GCN and GAT are capable of learning only low-pass filters. On the other

Table 6 Task 1: Learning filters on 50 images

Models	Low-pass $\exp(-10\lambda^2)$	High-pass $1 - \exp(-10\lambda^2)$	Band-pass $\exp(-10(\lambda - 1)^2)$	Band-rejection $1 - \exp(-10(\lambda - 1)^2)$	Comb $ \sin(\pi\lambda) $
GCN	3.4799 (.9872)	67.6635 (.2364)	25.8755 (.1148)	21.0747 (.9438)	50.5120 (.2977)
GAT	2.3574 (.9905)	21.9618 (.7529)	14.4326 (.4823)	12.6384 (.9652)	23.1813 (.6957)
GPR-GNN	0.4169 (.9984)	0.0943 (.9986)	3.5121 (.8551)	3.7917 (.9905)	4.6549 (.9311)
ARMA	1.8478 (.9932)	1.8632 (.9793)	7.6922 (.7098)	8.2732 (.9782)	15.1214 (.7975)
ChebNet	0.8220 (.9973)	0.7867 (.9903)	2.2722 (.9104)	2.5296 (.9934)	4.0735 (.9447)
BernNet	0.0314 (.9999)	0.0113 (.9999)	0.0411 (.9984)	0.9313 (.9973)	0.9982 (.9868)
JacobiConv	0.0003 (1.0000)	0.0011 (1.0000)	0.0213 (.9977)	0.0156 (1.0000)	0.2933 (.9961)
AGDN-HA	0.0762 (.9997)	0.0483 (.9994)	0.6425 (.9733)	0.9340 (.9976)	1.7097 (.9775)
AGDN-HC	0.0834 (.9997)	0.0241 (.9997)	0.4810 (.9817)	0.8696 (.9976)	1.3635 (.9814)
AGDN-HA *	0.0359 (.9999)	0.0446 (.9995)	0.0421 (.9984)	0.2303 (.9995)	0.2963 (.9962)
AGDN-HC *	0.0144 (1.0000)	0.0245 (.9997)	0.0135 (.9995)	0.0471 (.9999)	0.1547 (.9980)
AGDN-HC* (3-layer)	0.0088 (1.0000)	0.0001 (1.0000)	0.0000 (1.0000)	0.0143 (1.0000)	0.0238 (.9997)

The averaged sum of squared error (the lower is better) and R^2 scores (the higher is better) are reported. AGDN-HA*=AGDN-HA+Bernstein bases. AGDN-HC*=AGDN-HC+Bernstein bases

hand, GPR-GNN, ARMA, and ChebyNet can learn several filters, but they are significantly outperformed by AGDNs, BernNet, and JacobiConv on all tasks. AGDN-HC outperforms GPR-GNN on all tasks with a similar structure, demonstrating the effectiveness of the stacking multilayer architecture. BernNet and JacobiConv also outperform AGDNs with more complicated bases and carefully tuned hyperparameters. The margin between AGDNs and BernNet or JacobiConv is small and acceptable considering that we did not design AGDNs with polynomial bases. Incorporating Bernstein bases into AGDN, as denoted with AGDN-HA* and AGDN-HC* in Table 6, leads to improved performance, outperforming BernNet on all datasets except the high-pass filter, and surpassing JacobiConv on the band-pass and comb filters. AGDN-HC outperforms AGDN-HA on most datasets. This is reasonable since these tasks require the models to learn the same filters applied to all nodes, making node-specific HA weights difficult to optimize. Nonetheless, validating Corollary 6, the AGDN-HA can still learn all filters and outperform other baselines except BernNet and JacobiConv. Furthermore, by tuning a few hyperparameters ($L = 3$, $K = 15$, $d = 16$, $heads = 8$), the resulting AGDN-HC* (3-layer) significantly improves performance and outperforms all baselines on all filters, except for JacobiConv on the low-pass filter, where the gap is minimal. These results demonstrate that both AGDN-HA and AGDN-HC have the powerful ability to learn arbitrary filters in the spectral domain and can be further improved with certain polynomial bases.

6.2 Task 2: semisupervised node classification

For semisupervised node classification, we evaluated AGDNs on three homophily-prone datasets (ogbn-arxiv, ogbn-proteins and ogbn-products) (Hu et al. 2020) and three heterophily-prone datasets (Chameleon, Squirrel and Actor) (Rozemberczki et al. 2021; Pei et al. 2020). We select the best model based on validation scores.

For the first three datasets, we utilize GCN (Kipf and Welling 2017), GraphSAGE (Hamilton et al. 2017), DeeperGCN (Li et al. 2020), GCNII (Chen et al. 2020), MAGNA (Wang et al. 2021), UniMP (Shi et al. 2021), LEGNN (Yu et al. 2022) and RevGNN (Li et al. 2021) as baselines. LEGNN is a recently proposed label-enhanced GNN. RevGNN has been shown to be SOTA GNN and is still the main backbone of recent SOTA submissions with other tricks on OGB leaderboard. We also compare BernNet (He et al. 2021), utilizing its official implementation with the same hyperparameters used for the OGB baselines (3 layers with 256 hidden dimension, learning rate 0.01, weight decay 0, and 1000 epochs). We adopt the official dataset split from (Hu et al. 2020). We incorporate several popular tricks for AGDN on ogbn-arxiv, including the bag of tricks on GNNs (BoT) (Wang 2021), self-knowledge distillation (Self-KD) (Zhang et al. 2019), and graph information-aided node feature extraction with XR-transformers (GIANT-XRT) (Chien et al. 2022).

For the last three datasets, we utilize MLP, GCN (Kipf and Welling 2017), GAT (Vaswani et al. 2017), APPNP (Klicpera et al. 2019a), ChebyNet (Defferrard et al. 2016), GPR-GNN (Chien et al. 2021), JKNet (Chen et al. 2020), JacobiConv (Wang and Zhang 2022) and GOAL (Zheng et al. 2023) as baselines. We adopt the random dataset split, consistent across all classes, as established in the GOAL paper (Zheng et al. 2023), using the baseline results reported therein, except for JacobiConv (Wang and Zhang 2022), which was not evaluated in that study and uses a class-specific random split. We re-evaluated JacobiConv using its official implementation and the dataset split specified in GOAL to ensure consistency and

Table 7 Task 2: Semisupervised node classification on the ogbn-arxiv

Models	Accuracy (%)		Params
	Test	Valid	
GCN	71.74 $\pm$ 0.29	73.00 $\pm$ 0.17	0.11M
GraphSAGE	71.49 $\pm$ 0.27	72.77 $\pm$ 0.16	0.22M
DeeperGCN	71.92 $\pm$ 0.16	72.62 $\pm$ 0.14	0.49M
SIGN	71.95 $\pm$ 0.11	73.23 $\pm$ 0.06	3.57M
BernNet	71.49 $\pm$ 0.25	72.76 $\pm$ 0.41	0.07M
MAGNA	72.76 $\pm$ 0.14	–	–
GCNII	72.74 $\pm$ 0.16	–	2.15M
RevGNN	76.15 $\pm$ 0.10	77.16 $\pm$ 0.09	1.30M
LEGNN	73.37 $\pm$ 0.07	74.80 $\pm$ 0.09	5.37M
AGDN	76.37 $\pm$ 0.11	77.24 $\pm$ 0.06	1.31M

Except for AGDN, other results are from their papers or the OGB leaderboard

Table 8 Task 2: Semisupervised node classification on ogbn-proteins

Models	ROC-AUC(%)		Params
	Test	Valid	
GCN	72.51 $\pm$ 0.35	79.21 $\pm$ 0.18	0.10M
GraphSAGE	77.68 $\pm$ 0.20	83.34 $\pm$ 0.13	0.19M
DeeperGCN	85.80 $\pm$ 0.17	91.06 $\pm$ 0.16	2.37M
UniMP	86.42 $\pm$ 0.08	91.75 $\pm$ 0.06	1.91M
RevGNN-deep	87.74 $\pm$ 0.13	93.26 $\pm$ 0.06	20.03M
RevGNN-wide	88.24 $\pm$ 0.15	94.50 $\pm$ 0.08	68.47M
AGDN	88.65 $\pm$ 0.13	94.18 $\pm$ 0.05	8.61M

DeeperGCN, UniMP, RevGNN, and AGDN are implemented with random partitioning. GAT is implemented with neighbor sampling. Except for AGDN, other results are from their papers or the OGB leaderboard

comparability. In detail, we randomly split the node set of a dataset according to the ratio 60%/20%/20% for training, validation and test set respectively. We randomly generate the split 10 times and run each experiment.

Comparison with classical and decoupled GNNs

AGDN, which offers more expressive graph diffusion than graph convolution, outperforms classical GNNs, such as GCN and GraphSAGE, in semisupervised node classification tasks in Tables 7, 8, and 9. Moreover, the stacking multilayer architecture of AGDN endows it with better expressiveness than that of the decoupled GNN, SIGN, on ogbn-arxiv and ogbn-products. Additionally, AGDN outperforms BernNet on ogbn-arxiv, where learning arbitrary filters may not be as crucial since node features are important and low-pass filters are adequate to capture critical information. The stacking multilayer architecture also facilitates the application of critical tricks, such as BoT, self-KD, and GIANT-XRT, on AGDN in ogbn-arxiv. Remarkably, the AGDN achieves better performance (73.51% test accuracy) than that of other classical and decoupled GNNs, even without these tricks.

Comparison with spectral GNNs

As validated in Sect. 6.1, AGDNs demonstrate the ability to learn arbitrary filters, enabling strong performance on heterophily-prone datasets. Table 10 summarizes the average accuracy and standard deviation, showing that AGDN consistently outperforms all other spectral GNNs across the three datasets. Notably, AGDN surpasses the baselines by at least

Table 9 Task 2: Semisupervised node classification on ogbn-products

Models	Accuracy(%)		Params
	Test	Valid	
GCN	75.64 $\pm$ 0.21	92.00 $\pm$ 0.03	0.10M
GraphSAGE	78.50 $\pm$ 0.14	92.24 $\pm$ 0.07	0.21M
DeeperGCN	80.98 $\pm$ 0.20	92.38 $\pm$ 0.09	0.25M
SIGN	80.52 $\pm$ 0.16	92.99 $\pm$ 0.04	3.48M
UniMP	82.56 $\pm$ 0.31	93.08 $\pm$ 0.17	1.48M
RevGNN-112	83.07 $\pm$ 0.30	92.90 $\pm$ 0.07	2.95M
AGDN	83.34 $\pm$ 0.27	92.29 $\pm$ 0.10	1.54M

GAT, DeeperGCN, RevGNN, and AGDN are implemented with random partitioning. GraphSAGE and UniMP are implemented with neighbor sampling. Except for AGDN, all results are from their papers or the OGB leaderboard

Table 10 Task 2: Semisupervised node classification on heterophily-prone datasets

Models	Chameleon	Squirrel	Actor
GCN	62.72 $\pm$ 2.09	47.26 $\pm$ 0.34	29.98 $\pm$ 1.18
GAT	62.19 $\pm$ 3.78	51.80 $\pm$ 1.04	28.17 $\pm$ 1.19
APNP	50.88 $\pm$ 1.18	33.58 $\pm$ 1.00	29.82 $\pm$ 0.82
ChebyNet	59.98 $\pm$ 1.54	40.18 $\pm$ 0.55	35.85 $\pm$ 1.05
GPR-GNN	67.96 $\pm$ 2.55	49.52 $\pm$ 5.00	30.78 $\pm$ 0.61
JKNet	64.63 $\pm$ 3.08	44.91 $\pm$ 1.94	28.48 $\pm$ 1.25
JacobiConv	73.09 $\pm$ 1.35	56.70 $\pm$ 1.92	30.37 $\pm$ 0.93
GOAL	71.65 $\pm$ 1.66	60.53 $\pm$ 1.60	36.46 $\pm$ 1.02
AGDN	73.53 $\pm$ 1.45	61.37 $\pm$ 1.93	37.37 $\pm$ 1.38

Except for JacobiConv and AGDN, all results are from (Zheng et al. 2023). We use the official implementation of JacobiConv, along with its corresponding best parameters, and apply the dataset split from (Zheng et al. 2023) for consistency with the other baselines

0.44%, highlighting its capability to learn non-low-pass filters effectively. These improvements stem from the adaptive graph diffusion mechanism and the stacking multilayer architecture, which captures diverse structural information and aligns the learned weighting tensor with the target filter characteristics, yielding superior performance.

Comparison with residual GNNs

AGDN demonstrates superior performance compared to that of complex residual GNNs, including GCNII, LEGNN, and SOTA RevGNNs, as shown in Tables 7, 8, and 9, thanks to its adaptive graph diffusion. Furthermore, in Table 11, AGDN consistently outperforms RevGNNs (RevGAT) with similar complexity when critical tricks (BoT, self-KD, and GIANT-XRT) are progressively applied. The AGDN achieves superior performance on all datasets with moderate complexity, leveraging multihop information more effectively with a moderate receptive field and significantly fewer layers than deep residual GNNs. For instance, on ogbn-proteins, compact AGDN (6 layers) outperforms RevGNN-wide (448 layers) with only 13% parameter count, and on ogbn-products, compact AGDN (4 layers) outperforms RevGNN-112 (112 layers) with just 52% parameter count.

Table 11 Comparison with SOTA RevGNN on ogbn-arxiv

Models	Accuracy(%)		Params
	Test	Valid	
RevGNN+①	74.02 ± 0.18	75.01 ± 0.10	2.10M
AGDN+①	74.11 ± 0.12	75.25 ± 0.05	1.51M
RevGNN+①+②	74.26 ± 0.17	74.97 ± 0.08	2.10M
AGDN+①+②	74.31 ± 0.12	75.22 ± 0.09	1.51M
RevGNN+①+③	75.90 ± 0.19	77.01 ± 0.09	1.30M
AGDN+①+③	76.18 ± 0.16	77.24 ± 0.06	1.31M
RevGNN+①+②+③	76.15 ± 0.10	77.16 ± 0.09	1.30M
AGDN+①+②+③	76.37 ± 0.11	77.19 ± 0.08	1.31M

We report scores on the test set. The RevGNN results are from their papers or the OGB leaderboard. ①=BoT, ②=Self-KD, ③=GIANT-XRT embedding

Table 12 Task 3: Link prediction on ogbl-ppa

Models	Hits@100(%)		Params
	Test	Valid	
DeepWalk	28.88 ± 1.53	—	150.14M
Matrix Factorization	32.29 ± 0.94	32.28 ± 4.28	147.66M
Common Neighbor	27.65 ± 0.00	28.23 ± 0.00	0
Adamic Adar	32.45 ± 0.00	32.68 ± 0.00	0
Resource Allocation	49.33 ± 0.00	47.22 ± 0.00	0
GCN	18.67 ± 1.32	18.45 ± 1.40	0.28M
GraphSAGE	16.55 ± 2.40	17.24 ± 2.64	0.42M
PLNLP	32.38 ± 2.58	—	—
AGDN	30.27 ± 1.32	31.18 ± 1.07	0.02M

6.3 Task 3: link prediction

For link prediction, we evaluate models on ogbl-ppa, ogbl-ddi, and ogbl-citation2. We utilize DeepWalk (Perozzi et al. 2014), Matrix Factorization (Menon and Elkan 2011) (MF), Common Neighbor (Liben-Nowell and Kleinberg 2007) (CN), Adamic Adar (Adamic and Adar 2003) (AA), Resource Allocation (Zhou et al. 2009) (RA), GCN (Kipf and Welling 2017), GraphSAGE (Hamilton et al. 2017), and PLNLP (Wang et al. 2021) as baselines. GNN methods are implemented with the encoder-decoder framework (Kipf and Welling 2016; Hu et al. 2020), where a GNN encoder outputs node representations, and a simple decoder or predictor makes predictions using pairs of node representations as input. We select the best model based on validation scores.

As shown in Tables 12, 13 and 14, the performance of deterministic heuristic methods is better on ogbl-ppa but significantly worse on ogbl-ddi and ogbl-citation2 than that of encoder-decoder GNNs. With adaptive graph diffusion, AGDN surpasses some heuristic methods (such as common neighbor and Adamic Adar) on ogbl-ppa.

AGDN significantly outperforms GCN and GraphSAGE on ogbl-ppa, ogbl-ddi, and ogbl-citation2 due to its deeper receptive field. We trained AGDN in a full-batch fashion similar to GCN and GraphSAGE for ogbl-ppa and ogbl-ddi. However, we trained AGDN in a mini-batch fashion by using GraphSAINT for ogbl-citation2, which may negatively impact the final results. Nonetheless, AGDN still outperforms full-batch GCN and GraphSAGE on this dataset. Moreover, AGDN outperforms PLNLP on ogbl-ddi and ogbl-citation2, while it achieves comparable results on ogbl-ppa. Additionally, with learnable node

Table 13 Task 3: Link prediction on ogbl-ddi

Models	Hits@20(%)		Params
	Test	Valid	
DeepWalk	22.46 ± 2.90	–	11.54M
Matrix Factorization	13.68 ± 4.75	33.70 ± 2.64	1.22M
Common Neighbor	17.73 ± 0.00	9.47 ± 0.00	0
Adamic Adar	18.61 ± 0.00	9.66 ± 0.00	0
Resource Allocation	6.23 ± 0.00	7.25 ± 0.00	0
GCN	37.07 ± 5.07	55.50 ± 2.08	1.29M
GraphSAGE	53.90 ± 4.74	62.62 ± 0.37	1.42M
PLNLP	90.88 ± 3.13	82.42 ± 2.53	3.50M
AGDN	95.38 ± 0.94	89.43 ± 2.81	3.51M

Table 14 Task 3: Link prediction on ogbl-citation2

Models	MRR(%)		Params
	Test	Valid	
Matrix Factorization	51.86 ± 4.43	51.81 ± 4.36	281.11M
Common Neighbor	51.47 ± 0.00	51.19 ± 0.00	0
Adamic Adar	51.89 ± 0.00	51.67 ± 0.00	0
Resource Allocation	51.98 ± 0.00	51.77 ± 0.00	0
GCN	84.74 ± 0.31	84.79 ± 0.23	0.30M
GraphSAGE	82.60 ± 0.36	82.63 ± 0.33	0.46M
PLNLP	84.92 ± 0.29	84.90 ± 0.31	146.51M
AGDN	85.49 ± 0.29	85.56 ± 0.33	0.31M

GCN, GraphSAGE, and PLNLP are implemented with full batch training. AGDN is implemented with GraphSAINT

embeddings, AGDN achieves 41.23% Test and 43.32% Validation Hits@100 on ogbl-ppa. This comparison shows that enlarging the receptive field of GNNs can improve link prediction performance, and the adaptive graph diffusion in AGDN can exploit valuable information in the deep neighborhood for link prediction.

6.4 Runtime

Table 15 displays the training and inference runtimes of AGDN, GAT, and RevGNNs on the ogbn-arxiv, ogbn-proteins, and ogbn-products datasets. To account for the extra runtime introduced by generalized graph diffusion, we evaluate GATs by using the same configuration as AGDNs, including the number of layers. In ogbn-proteins, the inference of AGDN is performed on the same GPU, while the inference of RevGNNs is performed on a different Nvidia RTX A6000 (48 GB) without reporting the inference runtime (Li et al. 2021). In ogbn-products, the inference of all models is executed on CPU due to memory constraints. As expected, AGDNs demonstrate reasonable additional runtime for both training and inference. When comparing AGDNs and GATs with the same receptive fields, AGDNs require less time for training and similar time for inference. Moreover, we compare AGDNs with SOTA RevGNNs. RevGAT with only two layers on ogbn-arxiv requires less time for training than that needed for AGDNs. However, with more layers, RevGNNs require more time on ogbn-proteins and ogbn-products than do AGDNs. This suggests that AGDNs can achieve a better balance between accuracy and efficiency.

Table 15 Runtime comparison on Tesla V100 cards

Dataset	Model	Training	Inference
ogbn-arxiv	GAT (6×256 , 3 heads)	0.7h	0.6s
ogbn-arxiv	GAT (2×256 , 3 heads)	0.1h	0.2s
ogbn-arxiv	RevGAT (2×256 , 3 heads)	0.2h	0.2s
ogbn-arxiv	AGDN (2×256 , 3 heads, $K = 3$)	0.3h	0.3s
ogbn-proteins	GAT (12×150 , 6 heads)	2.7h	9.8s
ogbn-proteins	GAT (6×150 , 6 heads)	1.6h	8.0s
ogbn-proteins	RevGNN-Deep (1001×224)	324.0h	–
ogbn-proteins	RevGNN-Wide (448×224)	410.4h	–
ogbn-proteins	AGDN (6×150 , 6 heads, $K = 2$)	2.4h	9.3s
ogbn-products	GAT (12×120 , 4 heads)	7.3h	4.4m
ogbn-products	GAT (4×120 , 4 heads)	3.4h	1.7m
ogbn-products	RevGNN (112×160)	27.2h	14.5m
ogbn-products	AGDN (4×120 , 4 heads, $K = 3$)	6.5h	4.7m

We report the equivalent training runtime of 2000 epochs for each model. We describe the model architecture with $layers \times hidden$, possible heads, and diffusion depth K . s seconds, m minutes, h hours

6.5 Parameter analysis

A parameter sensitivity analysis was performed on AGDN by using the ogbn-arxiv dataset. AGDN utilizes similar hyperparameters to other GNNs or neural networks, such as learning rate, weight decay, and dropout. In this analysis, we primarily focused on the hyperparameter of the receptive field, a central aspect of this work. For AGDN, the receptive field (LK) can be expanded either by increasing the layer count (L) or the diffusion depth (K).

Table 16 presents the outcomes with various combinations of L and K . The optimal combination was identified as $L = 3$, $K = 2$. Some combinations led to exceeding the maximum memory (16 GB) and hence were omitted from the report. The results reveal a general trend of improved model performance when either L or K or both are increased. Expanding both L and K seems to enhance the receptive field more effectively than typical GNNs, without introducing excessive complexity or triggering oversmoothing. By comparing the significant difference between 1-layer cases and others in Table 16, it is evident that a minimum 2-layer architecture is required for nice expressiveness.

To balance efficiency and accuracy, an optimal range for L and K is between 2 and 3. Therefore, most experiments across various tasks and datasets in this paper incorporate combinations within this range.

6.6 Ablation study

Nodewise, channelwise

In this paper, we introduce nodewise and channelwise diffusion weights associated with HA and HC, respectively, which is a major contribution. An ablation study was conducted on the first learning filters task to demonstrate their effectiveness. To remove nodewise and

Table 16 Parameter analysis for the number of layers and diffusion depth on ogbn-arxiv

	$K = 1$	$K = 2$	$K = 3$	$K = 4$	$K = 5$	$K = 6$
$L = 1$	65.59	68.28	68.64	68.76	68.87	68.78
$L = 2$	72.28	73.00	73.18	73.20	73.14	73.20
$L = 3$	73.11	73.51	73.42	—	—	—
$L = 4$	72.90	—	—	—	—	—

We run each Experiment 10 times and report average scores (%) on the test set. Some results are not reported due to memory limitations

channelwise characteristics for HA and HC, respectively, weights were computed by using the same average node representation for each node (HA) and the same convolution kernel for each channel (HC). Table 17 shows that channelwise weights can consistently improve AGDN-HC performance. Moreover, nodewise weights can dramatically enhance the ability of AGDN-HA to learn all filters, especially nonlow-pass filters, overcoming the limitation discussed in Subsect. 5.2.

Stacking multilayer architecture

We investigated the impact of stacking multilayer architecture on AGDN by implementing AGDN-HA and AGDN-HC as decoupled models, each consisting of a single AGDN layer followed by linear layers. The only change in hyperparameters was setting $K = 10$ for consistent receptive fields. The results in Table 17 demonstrate that AGDN-HA and AGDN-HC suffer from significant performance degradation when used as single layers. Stacking multiple AGDN layers with variable filters allows for more flexible capture of target information, as discussed in Subsect. 5.3.

Positional embeddings

The importance of PEs in AGDN-HA was discussed in Corollary 6. These embeddings play a crucial role in ensuring that the AGDN-HA can theoretically produce any value, which is particularly important in learning nonlow-pass filters. To investigate this further, we conducted an ablation study on the learning filters task. The results, shown in Table 17, indicate that AGDN-HA performs significantly worse without PEs, especially on nonlow-pass filters. However, since AGDN-HC is already theoretically capable of learning arbitrary filters, PEs appear to be redundant for AGDN-HC. As a result, AGDN-HC performs slightly better without PEs, except in the cases of bandpass and band-rejection filters.

AGDN versus GAT

In Tables 18, we present an ablation study in which the performance of GAT and AGDN is compared on six OGB datasets. For ogbn-arxiv, we disabled additional techniques such as BoT, self-KD, and GIANT-XRT while maintaining consistent settings for both models. AGDN demonstrated superior performance over GAT on all datasets, highlighting the effectiveness of adaptive graph diffusion in AGDN.

Oversmoothing

We conducted experiments to compare AGDN variants with the various base models (GCN, GraphSAGE, GAT) and diffusion depths K ($K = 1, 2, \dots, 8$) on the ogbn-arxiv dataset by using the official OGB implementation. We also used classical GNNs (MPNNs) with equivalent receptive fields as baselines, where for a 3-layer AGDN model with $K = 4$, the associated MPNN baseline has $3 \times 4 = 12$ layers. Figure 4 shows that the MPNN baseline curves, especially for GAT, exhibit a distinct oversmoothing problem. AGDN-mean shows a quickly rising but significantly dropping curve, which is also affected by oversmoothing. In

Table 17 Ablation study for nodewise weights, channelwise weights and stacking multihop architecture on 50 images

Models	Low-pass	High-pass	Bandpass	Band-rejection	Comb
AGDN-HA	0.0762 (.9997)	0.0483 (.9994)	0.6425 (.9733)	0.9340 (.9976)	1.7097 (.9775)
No nodewise	3.5675↓ (.9896↓)	84.4571↓ (.0614↓)	28.9593↓ (.0152↓)	43.3306↓ (.8893↓)	63.9906↓ (.1367↓)
No stacking	0.4894↓ (.9987↓)	0.3879↓ (.9953↓)	3.4306↓ (.8709↓)	4.1852↓ (.9883↓)	5.8687↓ (.9221↓)
No PEs	2.6302↓ (.9917↓)	7.6725↓ (.9031↓)	11.3745 ↓ (.6004↓)	13.6497↓ (.9618↓)	13.0371↓ (.8132↓)
AGDN-HC	0.0834 (.9997)	0.0241 (.9997)	0.4810 (.9817)	0.8696 (.9976)	1.3635 (.9814)
No channelwise	0.0931↓ (.9996↓)	0.0312↓ (.9996↓)	0.6742↓ (.9709↓)	1.1427↓ (.9971↓)	1.8510↓ (.9745↓)
No stacking	0.3022↓ (.9989↓)	0.2415↓ (.9967↓)	3.4726↓ (.8575↓)	3.7194↓ (.9901↓)	5.8836↓ (.9190↓)
No PEs	0.0610↑ (.9997↑)	0.0236↑ (.9997↑)	0.4843↓ (.9800↓)	0.9155↓ (.9977↑)	1.3003↑ (.9817↑)

The symbol ↓ indicates a decrease in performance, as indicated by higher values for the sum of loss and lower values for the R^2 scores

Table 18 Ablation study on six OGB datasets

Models	ogbn-arxiv		ogbn-proteins		ogbn-products		ogbl-ppa		ogbl-ddi		ogbl-citation2	
	Accuracy (%)		ROC-AUC (%)		Accuracy (%)		Hits@100 (%)		Hits@10 (%)		MRR (%)	
	Test	Valid	Test	Valid	Test	Valid	Test	Valid	Test	Valid	Test	Valid
GAT	72.98	74.05	88.15	93.85	81.77	91.75	28.83	29.55	85.84	77.08	83.07	83.12
AGDN	73.41	74.23	88.65	94.18	83.34	92.29	30.27	31.18	95.38	89.43	85.49	85.56

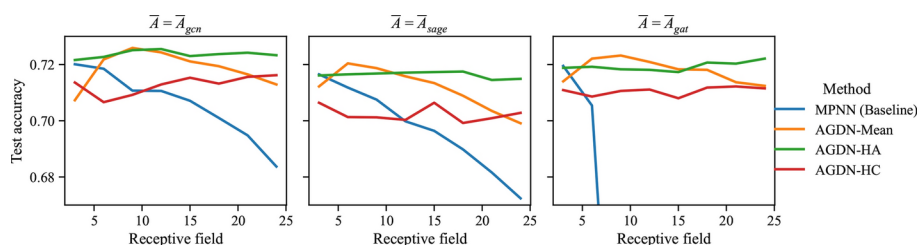


Fig. 4 Comparison among AGDN-Mean, AGDN-HA, AGDN-HC, and related base models with diffusion depths on ogbn-arxiv. For MPNN baselines, the “Receptive field” is the number of layers (L). For AGDN, the “Receptive field” is the product of the number of layers and diffusion depth (LK)

contrast, AGDN-HA and AGDN-HC show much more stable curves. AGDN-HA has similar optimal results to AGDN-mean, while AGDN-HC cannot outperform shallow baseline models. AGDN-HC directly learns weights that can take any values, which may be difficult to optimize on ogbn-arxiv. These results demonstrate that adaptive graph diffusion (HA or HC) can effectively alleviate the oversmoothing problem.

7 Discussion

In this work, we introduced adaptive graph diffusion networks (AGDNs) to tackle challenges posed by deep GNNs in numerous applications. The approach adopted by AGDNs involves learning nodewise (HA) and channelwise (HC) graph diffusion with a stacked multilayer architecture. This capability empowers them to effectively learn arbitrary filters in the spectral domain, thereby discerning intricate patterns within graph data.

Through this methodology, AGDNs have shown improved performance in tasks such as node classification and link prediction. This improvement places AGDNs as an alternative that strikes a better balance between complexity and performance when compared to those of complex residual GNNs and oversimplified decoupled GNNs.

However, despite considerable progress, certain areas warrant further research. The spectral characteristics of AGDN-HA, for instance, need additional exploration. Moreover, it is essential to assess the performance of AGDNs in more real-world applications to fully understand their practical implications and limitations.

8 Conclusion

In this paper, we present AGDNs to address the challenges faced by deep GNNs in various applications. AGDNs leverage the power of learnable nodewise (HA) and channelwise (HC) graph diffusion and the stacking multilayer architecture, enabling them to learn arbitrary filters in the spectral domain and to capture complex patterns and relationships in graph data. This approach leads to improved performance on tasks such as node classification, both on homophily-prone and heterophily-prone datasets, as well as link prediction. AGDNs offer a better balance between complexity and accuracy than those of complicated residual GNNs and oversimplified decoupled GNNs. They present a promising solution for improving the performance of GNNs on a range of graph-based applications. The spectral

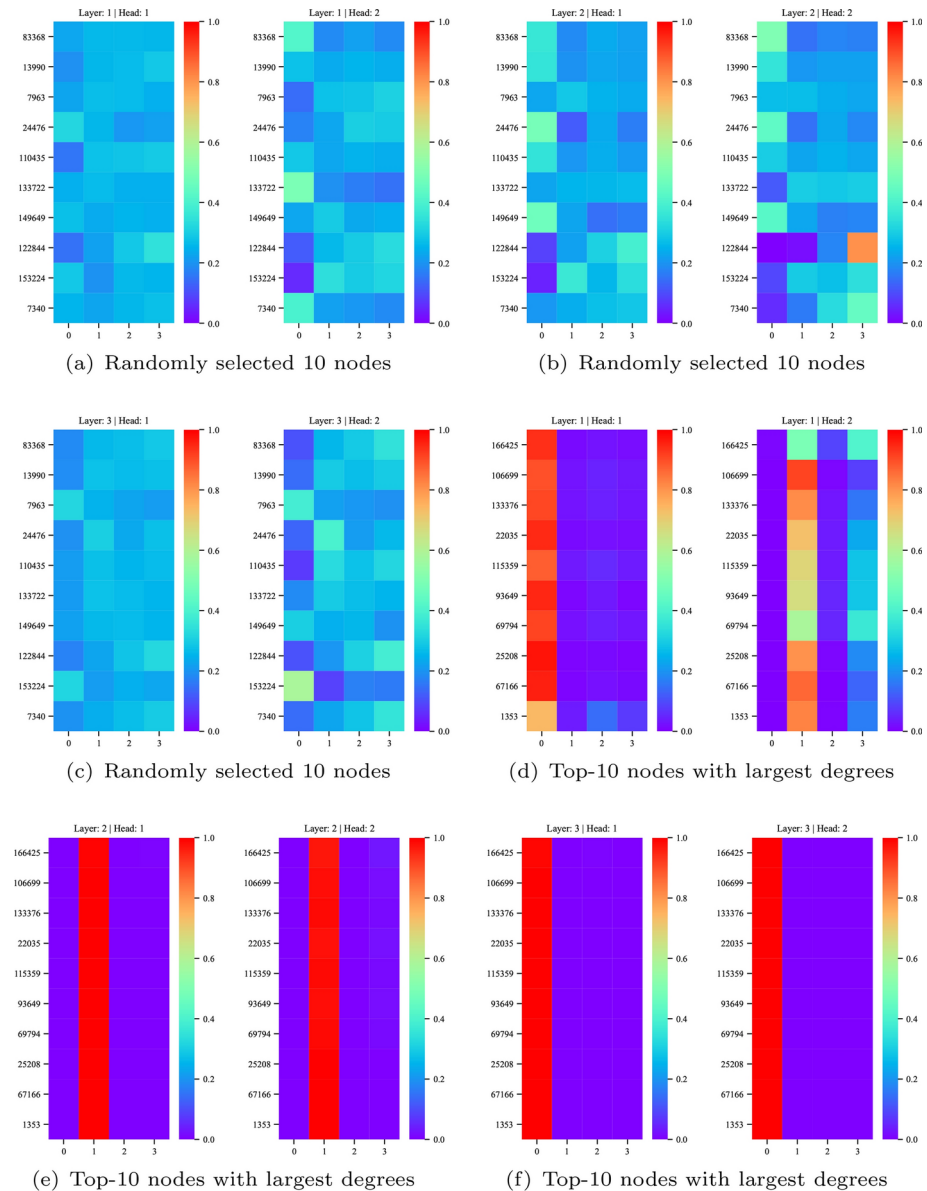


Fig. 5 Heatmaps of HA weight for 10 randomly selected nodes and top-10 nodes with the largest degrees across different layers and heads of a 3-layer AGDN-HA model with a GCN transition matrix trained on ogbn-arxiv. Warmer colors indicate higher weights

characteristics of AGDN-HA remain an area for future research, and the efficiency of the hopwise attention mechanism can be further optimized. Future work will focus on enhancing the spectral interpretability and efficiency of AGDN-HA, as well as exploring more effective positional embeddings.

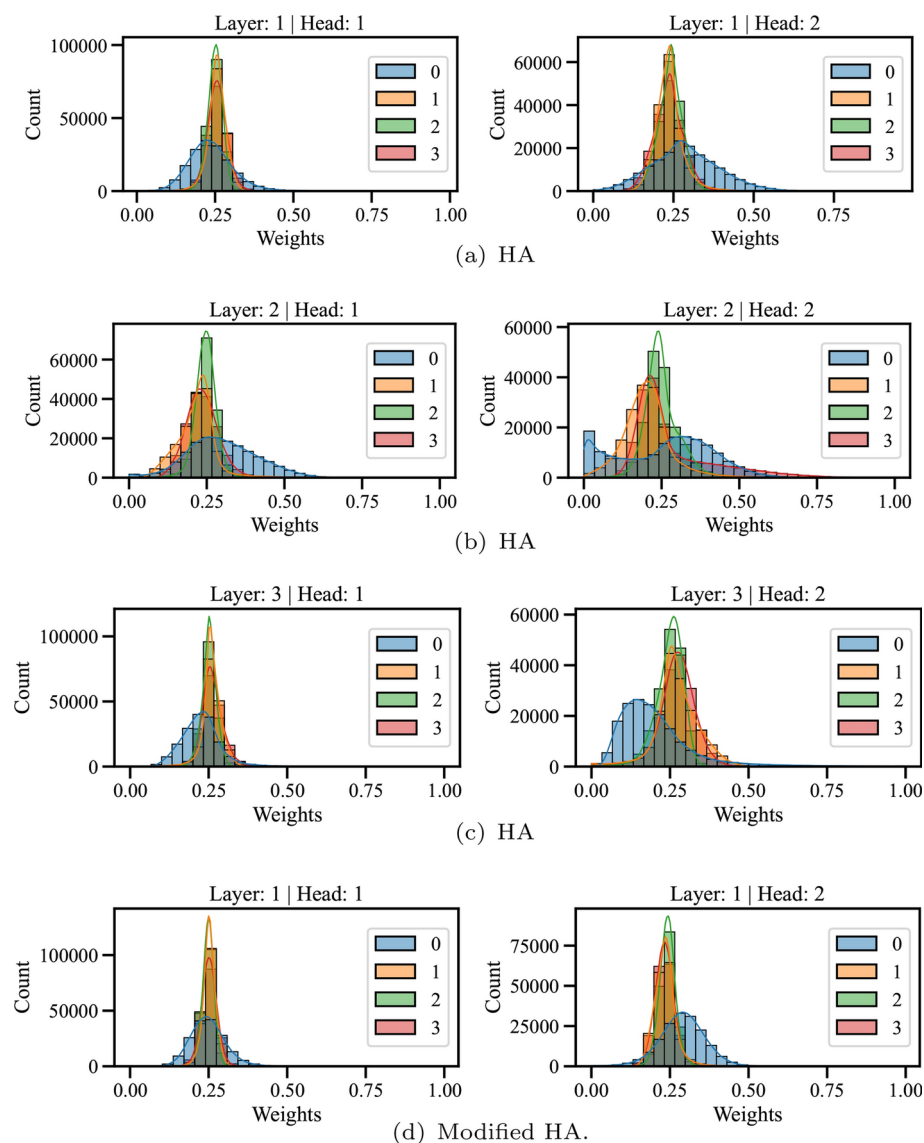


Fig. 6 Histograms showing the distribution of HA weights for different hops across different layers and heads of a 3-layer AGDN-HA model with a GCN transition matrix trained on ogbn-arxiv. The x-axis represents the values of HA weights, while the y-axis represents the number of nodes assigned with associated weight values. The values in the legends represent different hops

Appendix a empirical analysis of HA

We analyzed the distribution of HA weights across hops and nodes by using the ogbn-arxiv dataset (Hu et al. 2020). We evaluated a 3-layer AGDN-HA model with a GCN transition matrix and varied K and $heads$. We randomly selected 10 nodes and plotted their

HA weight heatmaps for different layers and heads (Fig. 5). We also plotted heatmaps and histograms for the top 10 nodes with the highest degrees to provide a more comprehensive view (Fig. 6). Furthermore, we investigated the effect of replacing the 0-th hop representation with the 1st hop representation in HA (Eq. 10).

Our results show that most nodes have HA weights smaller than 0.5, and HA weight distributions for most nodes vary across layers, with two heads sometimes learning complementary weights, as shown in Figs. 5a–c. For nodes with large degrees, as shown in Figs. 5d–f, the distribution of HA weights is imbalanced, and 0-th hop information is heavily emphasized for the first head in the first layer, indicating that the model tends not to use smoothed information for these nodes. In Fig. 6, histograms of HA weights for each hop are close to Gaussian and centered around the average value of 0.25, with small variances for all hops except the 0-th hop, indicating that different nodes' representations become similar after message-passing steps, acting as low-pass filters. In addition, in Fig. 6d, histograms from additional experiments with the modified HA, which replaces the 0-th hop representation with the 1st hop representation, are similar, suggesting that the 0-th hop information has larger variances than other hops since it has not been smoothed.

Author contributions All authors contributed to the study conception and design. Material preparation, data collection and analysis were performed by Chuxiong Sun, Hongming Gu and Jie Hu. The first draft of the manuscript was written by Chuxiong Sun, and all authors commented on previous versions of the manuscript. All the authors have read and approved the final manuscript.

Funding The authors did not receive support from any organization for the submitted work.

Data availability The datasets for the learning filters task are available [here](#). The Open Graph Benchmark datasets are available [here](#). The heterophily-prone datasets are available [here](#).

Code availability The code is available [here](#).

Declarations

Conflict of interest All authors certify that they have no affiliations with or involvement in any organization or entity with any financial interest or nonfinancial interest.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

- Adamic LA, Adar E (2003) Friends and neighbors on the web. *Soc Netw* 25(3):211–230. [https://doi.org/10.1016/S0378-8733\(03\)00009-1](https://doi.org/10.1016/S0378-8733(03)00009-1)
- Bianchi FM, Grattarola D, Livi L, Alippi C (2021) Graph neural networks with convolutional ARMA filters. *IEEE Trans Pattern Anal Mach Intell* 44(7):3496–3507. <https://doi.org/10.1109/TPAMI.2021.3054830>

- Balcilar M, Renton G, Héroux P, Gaüzère B, Adam S, Honeine P (2021) Analyzing the expressive power of graph neural networks in a spectral perspective. In: 9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3–7, 2021. <https://openreview.net/forum?id=qh0M9XWxnv>
- Chien E, Chang W, Hsieh C, Yu H, Zhang J, Milenkovic O, Dhillon IS (2022) Node feature extraction by self-supervised multi-scale neighborhood prediction. In: The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25–29, 2022. <https://openreview.net/forum?id=KJgglHbs8>
- Chung FR (1996) Lectures on spectral graph theory. CBMS Lectures, Fresno 6(92):17–21
- Chen J, Ma T, Xiao C (2018) Fastgcn: Fast learning with graph convolutional networks via importance sampling. In: 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30–May 3, Conference Track Proceedings (2018). <https://openreview.net/forum?id=rytstxWAW>
- Chien E, Peng J, Li P, Milenkovic O (2021) Adaptive universal generalized pagerank graph neural network. In: 9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3–7, 2021. <https://openreview.net/forum?id=n6jl7fLxrP>
- Chen M, Wei Z, Huang Z, Ding B, Li Y (2020) Simple and deep graph convolutional networks. In: Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13–18 July 2020, Virtual Event. Proceedings of Machine Learning Research, vol. 119, pp 1725–1735. <http://proceedings.mlr.press/v119/chen20v.html>
- Defferrard M, Bresson X, Vandergheynst P (2016) Convolutional neural networks on graphs with fast localized spectral filtering. In: Lee DD, Sugiyama M, Luxburg U, Guyon I, Garnett R (eds.) Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5–10, 2016, Barcelona, Spain, pp 3837–3845. <https://proceedings.neurips.cc/paper/2016/hash/04df4d434d481c5bb723be1b6df1ee65-Abstract.html>
- Du J, Zhang S, Wu G, Moura JM, Kar S (2017) Topology adaptive graph convolutional networks. Preprint at [arXiv:1710.10370](https://arxiv.org/abs/1710.10370)
- Fout A, Byrd J, Shariat B, Ben-Hur A (2017) Protein interface prediction using graph convolutional networks. In: Guyon I, Luxburg U, Bengio S, Wallach HM, Fergus R, Vishwanathan SVN, Garnett R (eds.) Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4–9, 2017, Long Beach, CA, USA, pp 6530–6539. <https://proceedings.neurips.cc/paper/2017/hash/f507783927f2ec2737ba40afbd17efb5-Abstract.html>
- Hu W, Fey M, Zitnik M, Dong Y, Ren H, Liu B, Catasta M, Leskovec J (2020) Open graph benchmark: Datasets for machine learning on graphs. In: Larochelle H, Ranzato M, Hadsell R, Balcan M, Lin H (eds.) Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6–12, 2020, Virtual. <https://proceedings.neurips.cc/paper/2020/hash/fb60d411a5c5b72b2e7d3527cfc84fd0-Abstract.html>
- Hou Z, Liu X, Cen Y, Dong Y, Yang H, Wang C, Tang J (2022) Graphmae: Self-supervised masked graph autoencoders. In: Zhang A, Rangwala H (eds.) KDD '22: The 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, August 14–18, 2022, pp 594–604. <https://doi.org/10.1145/3534678.3539321>
- He M, Wei Z, Huang Z, Xu H (2021) Bernnet: Learning arbitrary graph spectral filters via bernstein approximation. In: Ranzato, M., Beygelzimer, A., Dauphin, Y.N., Liang, P., Vaughan, J.W. (eds.) Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6–14, 2021, Virtual, pp 14239–14251. <https://proceedings.neurips.cc/paper/2021/hash/76f1cfd7754a6e4fc3281bcccb3d0902-Abstract.html>
- Hamilton WL, Ying Z, Leskovec J (2017) Inductive representation learning on large graphs. In: Guyon, I., Luxburg, U., Bengio, S., Wallach, H.M., Fergus, R., Vishwanathan, S.V.N., Garnett, R. (eds.) Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4–9, 2017, Long Beach, CA, USA, pp 1024–1034. <https://proceedings.neurips.cc/paper/2017/hash/5dd9db5e033da9c6fb5ba83c7a7e9ea9-Abstract.html>
- Klicpera J, Bojchevski A, Günnemann S (2019) Predict then propagate: Graph neural networks meet personalized pagerank. In: 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6–9, 2019. <https://openreview.net/forum?id=H1gL-2A9Ym>
- Kondor RI, Lafferty J (2002) Diffusion kernels on graphs and other discrete structures. In: Proceedings of the 19th International Conference on Machine Learning, vol. 2002, pp 315–22

- Krizhevsky A, Sutskever I, Hinton GE (2012) Imagenet classification with deep convolutional neural networks. In: Bartlett, P.L., Pereira, F.C.N., Burges, C.J.C., Bottou, L., Weinberger, K.Q. (eds.) *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a Meeting Held December 3–6, 2012, Lake Tahoe, Nevada, United States*, pp 1106–1114. <https://proceedings.neurips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html>
- Kipf TN, Welling M (2016) Variational graph auto-encoders. Preprint at [arXiv:1611.07308](https://arxiv.org/abs/1611.07308)
- Kipf TN, Welling M (2017) Semi-supervised classification with graph convolutional networks. In: 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24–26, 2017, Conference Track Proceedings. <https://openreview.net/forum?id=SJU4ayYgl>
- Klicpera J, Weißenberger S, Günnemann S (2019) Diffusion improves graph learning. In: Wallach HM, Larochelle H, Beygelzimer A, d'Alché-Buc F, Fox EB, Garnett R (eds.) *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8–14, 2019, Vancouver, BC, Canada*, pp 13333–13345. <https://proceedings.neurips.cc/paper/2019/hash/23c894276a2c5a16470e6a31f4618d73-Abstract.html>
- LeCun Y, Bottou L, Bengio Y, Haffner P (1998) Gradient-based learning applied to document recognition. *Proc IEEE* 86(11):2278–2324. <https://doi.org/10.1109/5.726791>
- Liu M, Gao H, Ji S (2020) Towards deeper graph neural networks. In: Gupta, R., Liu, Y., Tang, J., Prakash, B.A. (eds.) *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23–27*, pp 338–348 (2020). <https://doi.org/10.1145/3394486.3403076>
- Li Q, Han Z, Wu X (2018) Deeper insights into graph convolutional networks for semi-supervised learning. In: McIlraith, S.A., Weinberger, K.Q. (eds.) *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18), the 30th Innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2–7, 2018*, pp 3538–3545. <https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16098>
- Li, G., Müller M, Ghanem B, Koltun V (2021) Training graph neural networks with 1000 layers. In: Meila, M., Zhang, T. (eds.) *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18–24 July 2021, Virtual Event. Proceedings of Machine Learning Research*, vol. 139, pp 6437–6449. <http://proceedings.mlr.press/v139/li21o.html>
- Li, G., Müller M, Thabet AK, Ghanem B (2019) Deepgcn: Can gcns go as deep as cnns? In: 2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27–November 2, 2019, pp 9266–9275. <https://doi.org/10.1109/ICCV.2019.00936>
- Liben-Nowell D, Kleinberg J (2007) The link-prediction problem for social networks. *J Am Soc Inform Sci Technol* 58(7):1019–1031. <https://doi.org/10.1145/956863.956972>
- Li G, Xiong C, Thabet A, Ghanem B (2020) Deepergcn: All you need to train deeper gcns. Preprint at [arXiv:2006.07739](https://arxiv.org/abs/2006.07739)
- Menon AK, Elkan C (2011) Link prediction via matrix factorization. In: Gunopulos, D., Hofmann, T., Malerba, D., Vazirgiannis, M. (eds.) *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2011, Athens, Greece, September 5–9, 2011, Proceedings, Part II. Lecture Notes in Computer Science*, vol. 6912, pp 437–452. https://doi.org/10.1007/978-3-642-23783-6_28
- NT H, Machara T (2019) Revisiting graph neural networks: All we have is low-pass filters. *CoRR abs/1905.09550*[SPACE][arXiv: 1905.09550](https://arxiv.org/abs/1905.09550)
- Perozzi B, Al-Rfou R, Skiena S (2014) Deepwalk: online learning of social representations. In: Macskassy SA, Perlich C, Leskovec J, Wang W, Ghani, R (eds.) *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24–27, 2014*, pp 701–710. <https://doi.org/10.1145/2623330.2623732>
- Page L, Brin S, Motwani R, Winograd T (1999) The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab
- Pan S, Hu R, Long G, Jiang J, Yao L, Zhang C (2018) Adversarially regularized graph autoencoder for graph embedding. In: Lang J (ed.) *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13–19, 2018, Stockholm, Sweden*, pp 2609–2615. <https://doi.org/10.24963/ijcai.2018/362>
- Pei H, Wei B, Chang KC, Lei Y, Yang B (2020) Geom-gcn: Geometric graph convolutional networks. In: 8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26–30, 2020. OpenReview.net <https://openreview.net/forum?id=S1e2agrFvS>
- Rozemberczki B, Allen C, Sarkar R (2021) Multi-scale attributed node embedding. *J Complex Networks* <https://doi.org/10.1093/COMNET/CNAB014>
- Rossi E, Frasca F, Chamberlain B, Eynard D, Bronstein M, Monti F (2020) Sign: Scalable inception graph neural networks. Preprint at [arXiv:2004.11198](https://arxiv.org/abs/2004.11198)

- Shi Y, Huang Z, Feng S, Zhong H, Wang W, Sun Y (2021) Masked label prediction: Unified message passing model for semi-supervised classification. In: Zhou, Z. (ed.) Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19–27 August, pp 1548–1554. <https://doi.org/10.24963/ijcai.2021/214>
- Velickovic P, Cucurull G, Casanova A, Romero A, Liò, P, Bengio Y (2018) Graph attention networks. In: 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30–May 3, Conference Track Proceedings (2018). <https://openreview.net/forum?id=rJXMpikCZ>
- Von Luxburg U (2007) A tutorial on spectral clustering. *Stat Comput* 17:395–416. <https://doi.org/10.1007/s11222-007-9033-z>
- Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser L, Polosukhin I (2017) Attention is all you need. In: Guyon, I., Luxburg, U., Bengio, S., Wallach, H.M., Fergus, R., Vishwanathan, S.V.N., Garnett, R. (eds.) Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems, December 4–9, 2017, Long Beach, CA, USA, pp 5998–6008 (2017). <https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html>
- Wang Y (2021) Bag of tricks of semi-supervised classification with graph neural networks. Preprint at [arXiv:2103.13355](https://arxiv.org/abs/2103.13355)
- Wu F, Jr, AHS, Zhang T, Fifty C, Yu T, Weinberger KQ (2019) Simplifying graph convolutional networks. In: Chaudhuri K, Salakhutdinov R (eds.) Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9–15 June 2019, Long Beach, California, USA. Proceedings of Machine Learning Research, vol. 97, pp 6861–6871. <http://proceedings.mlr.press/v97/wu19e.html>
- Wang G, Ying R, Huang J, Leskovec J (2021) Multi-hop attention graph neural networks. In: Zhou Z (ed.) Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event/Montreal, Canada, 19–27 August, pp 3089–3096 (2021). <https://doi.org/10.24963/ijcai.2021/425>
- Wang X, Zhang M (2022) How powerful are spectral graph neural networks. In: Chaudhuri K, Jegelka S, Song L, Szepesvári C, Niu G, Sabato S (eds.) International Conference on Machine Learning, ICML 2022, 17–23 July, Baltimore, Maryland, USA. Proceedings of Machine Learning Research, vol. 162, pp 23341–23362 (2022). <https://proceedings.mlr.press/v162/wang22am.html>
- Wang Z, Zhou Y, Hong L, Zou Y, Su H (2021) Pairwise learning for neural link prediction. Preprint at [arXiv:2112.02936](https://arxiv.org/abs/2112.02936)
- Xu K, Li C, Tian Y, Sonobe T, Kawarabayashi K, Jegelka S (2018) Representation learning on graphs with jumping knowledge networks. In: Dy JG, Krause A (eds.) Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10–15, 2018. Proceedings of Machine Learning Research, vol. 80, pp 5449–5458. <http://proceedings.mlr.press/v80/xu18c.html>
- Yu L, Sun L, Du B, Zhu T, Lv W (2022) Label-enhanced graph neural network for semi-supervised node classification. *IEEE Trans Knowl Data Eng.* <https://doi.org/10.1109/TKDE.2022.3231660>
- Zhang M, Cui Z, Neumann M, Chen Y (2018) An end-to-end deep learning architecture for graph classification. In: McIlraith, S.A., Weinberger, K.Q. (eds.) Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th Innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2–7, pp 4438–4445 (2018). <https://www.aaai.org/ocs/index.php/AAI/AAAI18/paper/view/17146>
- Zhou T, Lü L, Zhang Y-C (2009) Predicting missing links via local information. *Eur Phys J B* 71(4):623–630. <https://doi.org/10.1140/epjb/e2009-00335-8>
- Zhang L, Song J, Gao A, Chen J, Bao C, Ma K (2019) Be your own teacher: Improve the performance of convolutional neural networks via self distillation. In: 2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27–November 2, 2019, pp 3712–3721. <https://doi.org/10.1109/ICCV.2019.00381>
- Zheng Y, Zhang H, Lee VC, Zheng Y, Wang X, Pan S (2023) Finding the missing-half: Graph complementary learning for homophily-prone and heterophily-prone graphs. In: Krause, A., Brunskill, E., Cho, K., Engelhardt, B., Sabato, S., Scarlett, J. (eds.) International Conference on Machine Learning, ICML 2023, 23–29 July 2023, Honolulu, Hawaii, USA. Proceedings of Machine Learning Research, vol. 202, pp 42492–42505. PMLR. <https://proceedings.mlr.press/v202/zheng23h.html>
- Zeng H, Zhou H, Srivastava A, Kannan R, Prasanna VK (2020) Graphsaint: Graph sampling based inductive learning method. In: 8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26–30, 2020. <https://openreview.net/forum?id=BJe8pkHFwS>

Authors and Affiliations

Chuxiong Sun¹ · Muhan Zhang² · Jie Hu¹ · Hongming Gu¹ · Jinpeng Chen³ · Mingchuan Yang¹

✉ Chuxiong Sun
chuxionsun@gmail.com

Muhan Zhang
muhan@pku.edu.cn

Jie Hu
hujie1@chinatelecom.cn

Hongming Gu
guhbm@chinatelecom.cn

Jinpeng Chen
jpchen@bupt.edu.cn

Mingchuan Yang
yangmch@chinatelecom.cn

¹ Big Data and AI Division, China Telecom Research Institute, Future City of Science, Changping 102218, Beijing, China

² Institute for Artificial Intelligence, Peking University, No.5 Yiheyuan Road, Haidian 100871, Beijing, China

³ School of Computer Science (National Pilot Software Engineering School), Beijing University of Posts and Telecommunications, No.10 West TuCheng Road, Haidian 100876, Beijing, China