Improving Graph Neural Networks with Learnable Propagation Operators

Moshe Eliasof [1] **Lars Ruthotto** [2] **Eran Treister** [1]

Abstract

Graph Neural Networks (GNNs) are limited in their propagation operators. In many cases, these operators often contain non-negative elements only and are shared across channels, limiting the expressiveness of GNNs. Moreover, some GNNs suffer from over-smoothing, limiting their depth. On the other hand, Convolutional Neural Networks (CNNs) can learn diverse propagation filters, and phenomena like over-smoothing are typically not apparent in CNNs. In this paper, we bridge these gaps by incorporating trainable channel-wise weighting factors ω to learn and mix multiple smoothing and sharpening propagation operators at each layer. Our generic method is called ωGNN, and is easy to implement. We study two variants: ωGCN and ωGAT. For ωGCN, we theoretically analyse its behaviour and the impact of ω on the obtained node features. Our experiments confirm these findings, demonstrating and explaining how both variants do not over-smooth. Additionally, we experiment with 15 real-world datasets on node- and graph-classification tasks, where our ωGCN and ωGAT perform on par with state-of-the-art methods.

1. Introduction

Graph Neural Networks (GNNs) are useful for a wide array of fields, from computer vision and graphics (Monti et al., 2017; Wang et al., 2018; Eliasof & Treister, 2020) and social network analysis (Kipf & Welling, 2017; Defferrard et al., 2016) to bio-informatics (Eliasof et al., 2022a; Jumper et al., 2021). Most GNNs are defined by applications of propagation and point-wise operators, where the former is often fixed and based on the graph Laplacian (e.g., GCN (Kipf & Welling, 2017)), or is defined by an attention mechanism (Veličković et al., 2018; Kim & Oh, 2021; Brody et al., 2022).

Most recent GNNs follow a general structure that involves two main ingredients – the propagation operator, denoted by $\mathbf{S}^{(l)}$, and a 1×1 convolution denoted by $\mathbf{K}^{(l)}$, as follows

$$\mathbf{f}^{(l+1)} = \sigma(\mathbf{S}^{(l)}\mathbf{f}^{(l)}\mathbf{K}^{(l)}), \tag{1}$$

where $\mathbf{f}^{(l)}$ denotes the feature tensor at the l-th layer. The main limitation of the above formulation is that the propagation operators in most common architectures are constrained to be non-negative. This leads to two drawbacks. First, this limits the expressiveness of GNNs (Chien et al., 2021). For example, the gradient of given graph node features can not be expressed by a non-negative operator. A mixed-sign operator, as in our proposed method, can achieve this, as demonstrated in Fig. 1 and Fig. 2. Second, the utilization of strictly non-negative propagation operators yields a smoothing process, that may lead GNNs to suffer from over-smoothing. That is, the phenomenon where node features become indistinguishable from one another as more GNN layers are stacked – causing severe performance degradation in deep GNNs (Nt & Maehara, 2019; Oono & Suzuki, 2020; Cai & Wang, 2020).

The above drawbacks are two gaps between GNNs and Convolutional Neural Networks (CNNs), which can be interpreted as structured versions of GNNs (i.e., GNNs operating on a regular grid). The structured convolutions in CNNs allow to learn diverse propagation operators, and in particular, it is known that mixed-sign (high-pass) kernels like sharpening filters are useful feature extractors in CNNs (Krizhevsky et al., 2012), and such operators cannot be obtained by non-negative (smoothing) kernels only. In the context of GNNs, Chien et al. (2021) have noticed that high-pass polynomial filters are learnt in their framework for heterophilic datasets, while low-pass filters are learnt for homophilic ones. A similar observation was noted in Eliasof et al. (2022b) as well. In addition, the over-smoothing phenomenon is typically not evident in standard CNNs where the spatial filters are learnt, and usually adding more layers improves accuracy (He et al., 2016).

A third gap between GNNs and CNNs is the ability of the latter to learn and mix multiple propagation operators. In the scope of separable convolutions, CNNs typically learn a distinct kernel per channel, known as a depth-wise

[1]Department of Computer Science, Ben-Gurion University, Beer-Sheva, Israel. [2]Department of Mathematics, Emory University, Atlanta, Georgia, USA. Correspondence to: Moshe Eliasof <eliasof@post.bgu.ac.il>, Eran Treister <erant@cs.bgu.ac.il>.

Proceedings of the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023.

convolution (Sandler et al., 2018) – a key element in modern CNNs (Tan & Le, 2019; Liu et al., 2022). On the contrary, in many GNNs the propagation operator $\mathbf{S}^{(l)}$ from (1) acts on all channels (Chen et al., 2020b; Veličković et al., 2018; Chien et al., 2021), and in some cases on all layers (Kipf & Welling, 2017; Wu et al., 2019). One exception is the multi-head GAT (Veličković et al., 2018) where several attention heads are learnt per layer. However, this approach typically employs only a few heads due to the high computational cost and is still limited by learning non-negative propagation operators only. Another exception is the recent (Wang & Zhang, 2022), a linear spectral GNN that learns a different high-order (k-hop) polynomial filter per channel. In contrast, our GNN is non-linear and faithful to the form (1) with a 1-hop spatial operator and 1×1 convolution at each layer, similar to classical and universal CNNs.

In this paper we propose a simple and effective modification of GNNs that closes the three gaps of GNNs discussed above, by introducing a parameter ω to control the contribution and type of the propagation operator. We call our general approach ωGNN, and utilize GCN (Kipf & Welling, 2017) and GAT (Veličković et al., 2018) to construct two variants, ωGCN and ωGAT. We theoretically prove and empirically demonstrate that our ωGNN can prevent over-smoothing. Secondly, we show that by learning ω, our ωGNNs can yield propagation operators with mixed signs, ranging from smoothing to sharpening operators (see Fig. 1 for an illustration). This approach enhances the expressiveness of the network, as demonstrated in Fig. 2, in contrast to many other GNNs that employ non-negative (smoothing) propagation operators only. Lastly, we propose and demonstrate that by learning different ω per layer and channel, similarly to a depth-wise convolution in CNNs, our ωGNNs obtain competitive accuracy.

Our contributions are summarized as follows:

- We propose ωGNN, an effective and computationally light modification to GNNs of a common and generic structure, that directly avoids over-smoothing and enhances the expressiveness of GNNs. Our method is demonstrated by ωGCN and ωGAT.
- A theoretical analysis and experimental validation of the behaviour of ωGNN are provided to expose its improved expressiveness compared to standard propagation operators in GNNs.
- We propose to learn multiple propagation operators by learning ω *per layer and per channel* and mixing them using a 1×1 convolution followed by a non-linear activation) to enhance the expressiveness of GNNs.
- Our experiments with 15 real-world datasets on numerous applications and settings, from semi- and fully-supervised node classification to graph classification show that our ωGCN and ωGAT read performance on par with current state-of-the-art methods.

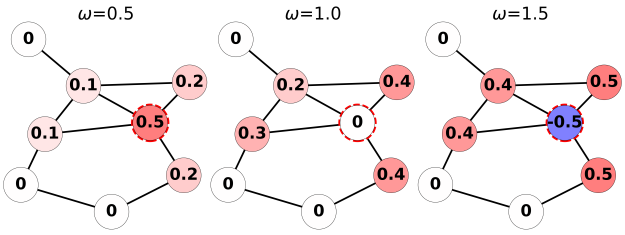

Figure 1. The impulse response of ωGCN's propagation operator for different ω values. For $\omega = 0.5, 1.0$ non-negative values are obtained, while for $\omega = 1.5$ we see mixed-sign values. The dashed node starts from a feature of 1 and the rest with 0.

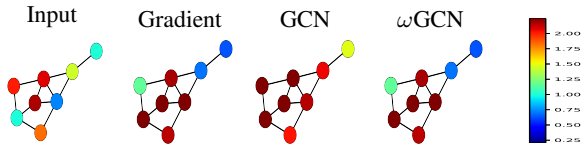

Figure 2. The expressiveness of ωGNNs. Our ωGCN can estimate the gradient of the node features while GCN cannot.

2. Method

We start by providing the notations that will be used throughout this paper, and displaying our general ωGNN in Section 2.1. Then we consider two popular GNNs that adhere to the structure presented in (1), namely GCN and GAT. We formulate and analyse the behaviour of their two counterparts ωGCN and ωGAT in Section 2.2 and 2.3, respectively.

Notations. Assume we are given an undirected graph defined by the set $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ is a set of n vertices and $\mathcal{E}$ is a set of m edges. Let us denote by $\mathbf{f}_i \in \mathbb{R}^c$ the feature vector of the i-th node of $\mathcal{G}$ with c channels. Also, we denote the adjacency matrix $\mathbf{A}$, where $\mathbf{A}_{ij} = 1$ if there exists an edge $(i, j) \in \mathcal{E}$ and 0 otherwise. We also define the diagonal degree matrix $\mathbf{D}$ where $\mathbf{D}_{ii}$ is the degree of the i-th node. The graph Laplacian is given by $\mathbf{L} = \mathbf{D} - \mathbf{A}$. Let us also denote the adjacency and degree matrices with added self-loops by $\tilde{\mathbf{A}}$ and $\tilde{\mathbf{D}}$, respectively. Lastly, we denote the symmetrically normalized graph Laplacian by $\tilde{\mathbf{L}}^{sym} = \tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{L}} \tilde{\mathbf{D}}^{-\frac{1}{2}}$ where $\tilde{\mathbf{L}} = \tilde{\mathbf{D}} - \tilde{\mathbf{A}}$.

2.1. ωGNNs

The goal of ωGNNs is to utilize learnable mixed-sign propagation operators that control smoothing and sharpening to enrich GNNs expressiveness. Below, we describe how the learnt ω influences the obtained operator and how to learn and mix multiple operators for enhanced expressiveness.

Learning propagation weight ω. To address the expressiveness and over-smoothing issues, we suggest a general form given an arbitrary non-negative and normalized (e.g., such that its row sums equal to 1) propagation operator $\mathbf{S}^{(l)}$. Our general ωGNN is then given by

$$\mathbf{f}^{(l+1)} = \sigma\left(\left(\mathbf{I} - \omega^{(l)}\left(\mathbf{I} - \mathbf{S}^{(l)}\right)\right)\mathbf{f}^{(l)}\mathbf{K}^{(l)}\right), \quad (2)$$

where $\omega^{(l)}$ is a scalar that is learnt per layer, and in the next paragraph we offer a more elaborated version with a parameter ω per layer and channel. The introduction of $\omega^{(l)}$ allows our ωGNN layer to behave in a three-fold manner. When $\omega^{(l)} \leq 1$, a smoothing process is obtained [1]. Note, that for $\omega^{(l)} = 1$, (2) reduces to the standard GNN dynamics from (1). In case $\omega^{(l)} = 0$, (2) reduces to a 1×1 convolution followed by a non-linear activation function, and does not propagate neighbouring node features. On the other hand, if $\omega^{(l)} > 1$, we obtain an operator with negative signs on the diagonal but positive on the off-diagonal entries, inducing a sharpening operator. An example of various $\omega^{(l)}$ values and their impulse response is given in Fig. 1. Thus, a learnable $\omega^{(l)}$ allows to learn a new family of operators, namely sharpening operators, that are not achieved by methods like GCN and GAT. To demonstrate the importance of sharpening operators, we consider a synthetic task of node gradient feature regression, given a graph and input node features(see Appendix B for more details). As depicted in Fig. 2, using a non-negative operator as in GCN cannot accurately express the gradient operator output, while our ωGCN estimates the gradient output with a machine precision accuracy. Also, the benefit of employing both smoothing and sharpening operators is reflected in the obtained accuracy of our method on real-world datasets in Section 4.

Multiple propagation operators. To learn multiple propagation operators, we extend (2) from a channels-shared weight to channel-wise weights by learning a vector $\vec{\omega}^{(l)} \in c$ as follows

$$\mathbf{f}^{(l+1)} = \sigma\left(\left(\mathbf{I} - \mathbf{\Omega}_{\vec{\omega}^{(l)}}\left(\mathbf{I} - \mathbf{S}^{(l)}\right)\right)\mathbf{f}^{(l)}\mathbf{K}^{(l)}\right), \quad (3)$$

where $\mathbf{\Omega}_{\vec{\omega}^{(l)}}$ is an operator that scales each channel j with a different $\omega_j^{(l)}$. As discussed in Section 1, this procedure yields a propagation operator per-channel, which is similar to depth-wise convolutions in CNNs (Howard et al., 2017; Sandler et al., 2018). Thus, the extension to a vector $\vec{\omega}^{(l)}$ helps to further bridge the gap between GNNs and CNNs.

We note that using this approach, our ωGNN is suitable to many existing GNNs, and in particular to those which act as a separable convolution, as described in Eq. (1). In what follows, we present and analyse two variants based on GCN and GAT, called ωGCN and ωGAT, respectively.

[1] The use of the value 1 in this discussion corresponds to a non-negative operator $\mathbf{S}^{(l)}$ with zeros on its diagonal, normalized to have row sums of 1. Other normalizations may yield other constants. Also, if $0 < \mathbf{S}_{ii}^{(l)} < 1$, then setting $\omega^{(l)} > \frac{1}{1-\mathbf{S}_{ii}^{(l)}}$ flips the sign of the i-th diagonal entry.

2.2. ωGCN

GCNs are a class of GNNs that employ a pre-determined propagation operator $\tilde{\mathbf{P}} = \tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}$, that stems from the graph Laplacian. For instance, GCN (Kipf & Welling, 2017) is given by:

$$\mathbf{f}^{(l+1)} = \sigma(\tilde{\mathbf{P}}\mathbf{f}^{(l)}\mathbf{K}^{(l)}), \quad (4)$$

that is, by setting $\mathbf{S}^{(l)} = \tilde{\mathbf{P}}$ in (1).

Other methods like SGC (Wu et al., 2019), GCNII (Chen et al., 2020b) and EGNN (Zhou et al., 2021) also rely on $\tilde{\mathbf{P}}$ as a propagation operator.

The operator $\tilde{\mathbf{P}}$ is a fixed non-negative smoothing operator, hence, repeated applications of (4) lead to the over-smoothing phenomenon, where the feature maps converge to a single eigenvector as shown by (Wu et al., 2019; Wang et al., 2019). Moreover, $\tilde{\mathbf{P}}$ is pre-determined, and solely depends on the graph connectivity, disregarding the node features, which may harm performance.

By baking our proposed ωGNN with a learnable weight, denoted by $\omega^{(l)} \in \mathbb{R}$ into GCN we obtain the following propagation scheme, named ωGCN:

$$\mathbf{f}^{(l+1)} = \sigma\left(\left(\mathbf{I} - \omega^{(l)}\left(\mathbf{I} - \tilde{\mathbf{P}}\right)\right)\mathbf{f}^{(l)}\mathbf{K}^{(l)}\right). \quad (5)$$

We now present theoretical analyses of our ωGCN and reason about its *non* over-smoothing property. We first define the node features Dirichlet energy at the l-th layer, as in (Zhou et al., 2021):

$$E(\mathbf{f}^{(l)}) = \sum_{i\in\mathcal{V}}\sum_{j\in\mathcal{N}_i}\frac{1}{2}\left\|\frac{\mathbf{f}_i^{(l)}}{\sqrt{(1+d_i)}} - \frac{\mathbf{f}_j^{(l)}}{\sqrt{(1+d_j)}}\right\|_2^2. \quad (6)$$

Fig. 3 demonstrates how the Dirichlet energy $E(\mathbf{f}^{(l)})$ decays to zero when ω is a constant, and to a fixed positive value when ω is learnt. Next, we provide a theorem that characterizes the behaviour of ω and how it prevents over-smoothing. To this end we denote the propagation operator of ωGCN from (5) by

$$\begin{aligned}\tilde{\mathbf{P}}_\omega &= \mathbf{I} - \omega\left(\mathbf{I} - \tilde{\mathbf{P}}\right) &(7)\\ &= \mathbf{I} - \omega\left(\mathbf{I} - \tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\right) &(8)\\ &= \mathbf{I} - \omega\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{L}\tilde{\mathbf{D}}^{-\frac{1}{2}}, &(9)\end{aligned}$$

where the latter equality is shown in Appendix A. In essence, we show that repeatedly applying the operator $\tilde{\mathbf{P}}$ is equivalent to applying gradient descent steps for minimizing (6)

with a learning rate ω. We build on the observation that smoothing is beneficial (Gasteiger et al., 2019; Chamberlain et al., 2021) and assume that for each dataset there exists a plausible energy value at the last layer that satisfies $0 < E(\mathbf{f}^{(L)}) < E(\mathbf{f}^{(0)})$. We note that if the graph is strongly connected, then if $E(\mathbf{f}^{(L)}) = 0$ it means that all the feature maps are constant at the output of the network. It is reasonable to expect that such constant feature maps will not yield good performance, at least in tasks like node classification, for example.

Now, assuming that we wish to have some energy E^* at the last layer, we show that if we learn a single $\omega^{(l)} = \omega > 0$, shared across all layers, then taking L to infinity will lead the learned ω to zero, scaling as $1/L$. Thus, our ωGCN will not over-smooth, as the energy at the last layer $E(\mathbf{f}^{(L)})$ can reach to E_L. Later, in Corollary 2.2, we generalize this result for a per-layer $\omega^{(l)}$, and empirically validate both results in Section 4.4 and Fig. 4. The proofs for the Theorem and Corollary below are given in Appendix A.

Theorem 2.1. *Consider L applications of (7), i.e., $\mathbf{f}^{(L)} = (\tilde{\mathbf{P}}_\omega)^L \mathbf{f}^{(0)}$ with a shared parameter $\omega^{(l)} = \omega$ that is used in all the layers. Also, assume that there is some desired Dirichlet energy $E(\mathbf{f}^{(L)}) = E^*$ of the final feature map that satisfies $0 < E^* < E(\mathbf{f}^{(0)})$. Then, at the limit, as more layers are added and L grows, the value of the learnt ω converges to T/L for some constant T, up to first-order accuracy.*

Corollary 2.2. *Allowing a variable $\omega^{(l)} > 0$ at each layer in Theorem 2.1, yields that $\sum_{l=0}^{L-1} \omega^{(l)}$ converges to a constant independent of L up to first order accuracy.*

Next, we dwell on the second mechanism in which ωGCN prevents over-smoothing. We analyse the eigenvectors of $\tilde{\mathbf{P}}_\omega$, showing that different choices of ω yield different leading eigenvectors that alter the behaviour of the propagation operator (i.e. smoothing and sharpening processes). This result is useful because changing the leading eigenvector prevents the gravitation towards a specific eigenvector, which causes the over-smoothing to occur (Wu et al., 2019; Oono & Suzuki, 2020).

Theorem 2.3. *Assume that the graph is connected. Then, there exists some $\omega_0 \geq 1$ where for all $0 < \omega < \omega_0$, the operator $\tilde{\mathbf{P}}_\omega$ in (7) is smoothing and the leading eigenvector is $\tilde{\mathbf{D}}^{\frac{1}{2}}\mathbf{1}$. For $\omega > \omega_0$ or $\omega < 0$, the leading eigenvector changes.*

The proof for the theorem is given in Appendix A.

ωGCN with multiple propagation operators. To further increase the expressiveness of our ωGCN we extend $\omega^{(l)} \in \mathbb{R}$ to $\vec{\omega}^{(l)} \in \mathbb{R}^c$ and learn a propagation operator per channel, at each layer. To this end, we modify (5) to the following formulation

$$\mathbf{f}^{(l+1)} = \sigma\left(\left(\mathbf{I} - \mathbf{\Omega}_{\vec{\omega}^{(l)}}\left(\mathbf{I} - \tilde{\mathbf{P}}\right)\right)\mathbf{f}^{(l)}\mathbf{K}^{(l)}\right). \quad (10)$$

As we show in Section 4.4, learning a propagation operator per channel is beneficial to improve accuracy.

2.3. ωGAT

The seminal GAT (Veličković et al., 2018) learns a non-negative edge-weight as follows

$$\alpha_{ij}^{(l)} = \frac{\exp\left(\text{LeakyReLU}\left(\mathbf{a}^{(l)^\top}[\mathbf{W}^{(l)}\mathbf{f}_i^{(l)} || \mathbf{W}^{(l)}\mathbf{f}_j^{(l)}]\right)\right)}{\sum_{p\in\mathcal{N}_i}\exp\left(\text{LeakyReLU}\left(\mathbf{a}^{(l)^\top}[\mathbf{W}^{(l)}\mathbf{f}_i^{(l)} || \mathbf{W}^{(l)}\mathbf{f}_p^{(l)}]\right)\right)}, \quad (11)$$

where $\mathbf{a}^{(l)} \in \mathbb{R}^{2c}$ and $\mathbf{W}^{(l)} \in \mathbb{R}^{c\times c}$ are trainable parameters and $||$ denotes channel-wise concatenation. Here, GAT is obtained by defining the propagation operator $\mathbf{S}^{(l)}$ in (1) as $\hat{\mathbf{S}}_{ij}^{(l)} = \alpha_{ij}$.

To avoid repeated equations, we skip the per-layer ω formulation (as in (2)) and directly define the per-channel ωGAT as follows

$$\mathbf{f}^{(l+1)} = \sigma\left(\left(\mathbf{I} - \mathbf{\Omega}_{\vec{\omega}^{(l)}}\left(\mathbf{I} - \hat{\mathbf{S}}^{(l)}\right)\right)\mathbf{f}^{(l)}\mathbf{K}^{(l)}\right). \quad (12)$$

The introduction of $\mathbf{\Omega}_{\vec{\omega}^{(l)}}$ yields a learnable propagation operator per layer and channel. We note that it is also possible to obtain multiple propagation operators from GAT by using a multi-head attention. However, we distinguish our proposition from GAT in a 2-fold fashion. First, our propagation operators belong to a broader family that includes smoothing and sharpening operators as opposed to smoothing-only due to the SoftMax normalization in GAT. Secondly, our method requires less computational overhead when adding more propagation operators, as our ωGAT requires a scalar per operator, while GAT doubles the number of channels to obtain more attention-heads. Also, utilizing a multi-head GAT can still lead to over-smoothing, as all the heads induce a non-negative operator.

To study the behaviour of our ωGAT, we inspect its node features energy compared to GAT. To this end, we define the GAT energy as

$$E_{\text{GAT}}(\mathbf{f}^{(l)}) = \sum_{i\in\mathcal{V}}\sum_{j\in\mathcal{N}_i}\frac{1}{2}||\mathbf{f}_i^{(l)} - \mathbf{f}_j^{(l)}||_2^2. \quad (13)$$

This modification of the Dirichlet energy from (6) is required because in GAT (Veličković et al., 2018) the leading eigenvector of the propagation operator $\hat{\mathbf{S}}^{(l)}$ is the constant vector $\mathbf{1}$ as shown by Chen et al. (2020a), unlike the vector $\tilde{\mathbf{D}}^{\frac{1}{2}}\mathbf{1}$ in the symmetric normalized $\tilde{\mathbf{P}}$ from GCN (Kipf & Welling, 2017) where the Dirichlet energy is natural to consider (Pei et al., 2020).

We present the energy of a 64 layer GAT trained on the Cora dataset in Fig. 3. It is evident that the accuracy degradation

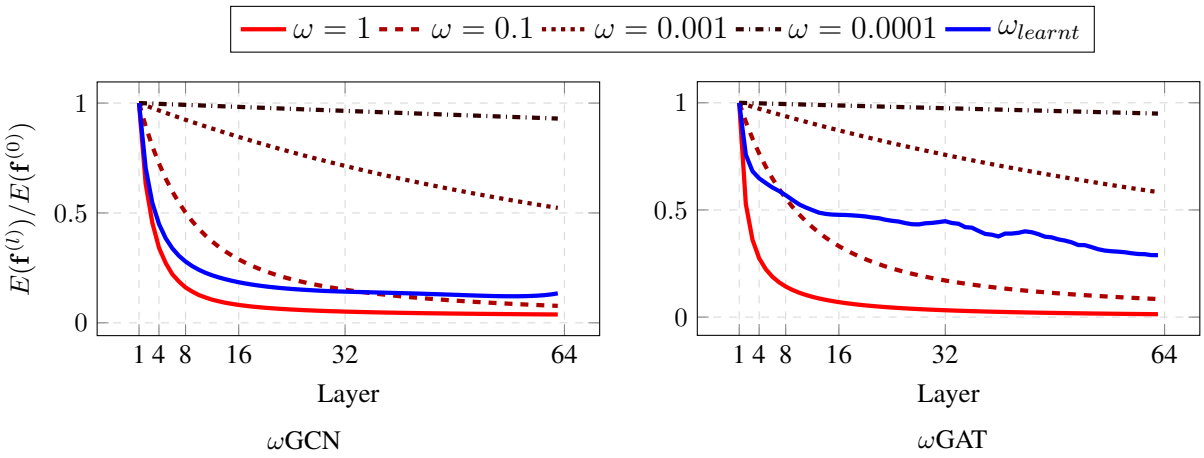

Figure 3. Node features energy at the l-th layer relative to the initial node embedding energy on Cora. Both ωGCN and ωGAT control the respective energies from (6) and (13) to avoid over-smoothing, while the baselines with $\omega = 1$ reduce the energies to 0 and over-smooth.

of GAT reported by (Zhao & Akoglu, 2020) is in congruence with the decaying energy in (13), while our ωGAT does not experience decaying energy nor accuracy degradation as more layers are added, as can be seen in Table 2. To further validate our findings, we repeat this experiment in Appendix D on additional datasets and reach the same conclusion.

2.4. Computational Costs

Our ωGNN approach is general and can be applied to any GNN that conforms to the structure of (1) and can be modified into (3). The additional parameters compared to the baseline GNN are the added $\mathbf{\Omega}_{\vec{\omega}^{(l)}} \in \mathbb{R}^c$ parameters at each layer, yielding a relatively low computational overhead. For example, in GCN (Kipf & Welling, 2017) there are $c \times c$ trainable parameters requiring $c \times c \times n$ multiplications due to the 1×1 convolution $\mathbf{K}^{(l)}$. In our ωGCN, we will have $c \times c + c$ parameters and $(c+1) \times c \times n$ multiplications. That is in addition to applying the propagation operators $\mathbf{S}^{(l)}$, which are identical for both methods. A similar analysis holds for GAT. To validate the actual complexity of our method, we present the training and inference times for ωGCN and ωGAT in Appendix G. We see a negligible addition to the runtimes compared to the baselines, at the return of better performance.

3. Other Related Work

Over-smoothing in GNNs. The over-smoothing phenomenon was identified by (Li et al., 2018), and was profoundly studied in recent years. Various methods stemming from different approaches were proposed. For example, methods like DropEdge (Rong et al., 2020), PairNorm (Zhao & Akoglu, 2020), and EGNN (Zhou et al., 2021) propose augmentation, normalization and energy-based penalty methods to alleviate over-smoothing, respectively. Other methods like (Min et al., 2020) propose to augment GCN with geometric scattering transforms and residual convolutions, and GCNII (Chen et al., 2020b) present a spectral analysis of the smoothing property of GCN (Kipf & Welling, 2017) and propose adding an initial identity residual connection and a decay of the weights of deeper layers, which are also used in EGNN (Zhou et al., 2021). Also, in (Luan et al., 2020), low and high pass filter banks are utilized to alleviate over-smoothing.

Graph Neural Diffusion. The view of GNNs as a diffusion process has gained popularity in recent years. Methods like APPNP (Klicpera et al., 2019) propose to use a personalized PageRank (Page et al., 1999) algorithm to determine the diffusion of features, and GDC (Gasteiger et al., 2019) imposes constraints on the ChebNet (Defferrard et al., 2016) architecture to obtain diffusion kernels, showing accuracy improvement. Other works like GRAND (Chamberlain et al., 2021), CFD-GCN (Belbute-Peres et al., 2020), PDE-GCN (Eliasof et al., 2021) and GRAND++ (Thorpe et al., 2022) propose to view GNN layers as time steps in the integration process of ODEs and PDEs that arise from a non-linear heat equation, allowing to control the diffusion (smoothing) in the network to prevent over-smoothing. In addition, some GNNs (Eliasof et al., 2021; Rusch et al., 2022a) propose a mixture between diffusion and oscillatory processes to avoid over-smoothing by frequency preservation of the features.

Mixed-sign operators in GNNs. The importance of mixed-sign high-pass vs low-pass filters was discussed in the spectral GPRGNN (Chien et al., 2021), which utilizes a single high-order polynomial (k-hop) filter. The authors show that weights corresponding to high-pass filters are obtained for heterophilic datasets. In Appendix K, we show similar conclusions. Other similar spectral methods include JacobiConv (Wang & Zhang, 2022), BernNet (He et al., 2021a). In all these methods, a high-order polynomial is learnt, where

all the propagation layers are stacked linearly one after the other. In contrast, we wish to follow the non-linear form (1) with a 1-hop propagation and 1×1 convolution for each layer, as this is a form that is closest to standard CNNs, which have been proven to be effective for many challenging tasks in computer vision (Chen et al., 2017; Tan & Le, 2019; Sandler et al., 2018). Mixed-sign operators were also presented in the works of Yang et al. (2021) and Yan et al. (2021), employing attention-based propagation operator with a tanh/cosine activation function to obtain values in $[-1, 1]$. This mechanism is different than our weighting approach which in which the type of operation is learnt directly. Lastly, mixed-sign operators were also discussed in (Eliasof et al., 2022b), where k-hop filters and stochastic path sampling mechanisms are utilized. However, such a method requires significantly more computational resources than a standard GNN like (Kipf & Welling, 2017) due to the path sampling strategy and larger filters of 5-hop required for optimal accuracy. However, our ωGNNs perform 1-hop propagations and, as we show in Appendix G, obtain competitive results without significant added computational costs.

4. Experiments

We demonstrate our ωGCN and ωGAT [2] on node classification, inductive learning and graph classification tasks. Additionally, we conduct an ablation study of the different configurations of our method and experimentally verify the theorems from Section 2. A description of the network architectures is given in Appendix E. We use the Adam (Kingma & Ba, 2014) optimizer in all experiments, and perform grid search to determine the hyper-parameters reported in Appendix F. The objective function in all experiments is the cross-entropy loss, besides inductive learning on PPI (Hamilton et al., 2017) where we use the binary cross-entropy loss. Our code is implemented with PyTorch (Paszke et al., 2019) and PyTorch-Geometric (Fey & Lenssen, 2019) and trained on an Nvidia Titan RTX GPU.

We show that for all the considered tasks and datasets, whose statistics are provided Appendix C, our ωGCN and ωGAT are either better or on par with other state-of-the-art models.

4.1. Node Classification

We employ the Cora, Citeseer, and Pubmed datasets using the standard training/validation/testing split by (Yang et al., 2016), with 20 nodes per class for training, 500 validation nodes, and 1,000 testing nodes. We follow the training and evaluation scheme of (Chen et al., 2020b) and compare with models like GCN, GAT, superGAT (Kim & Oh, 2021), Inception (Szegedy et al., 2017), APPNP (Klicpera et al., 2019), JKNet (Xu et al., 2018), DropEdge (Rong et al., 2020), GCNII (Chen et al., 2020b), GRAND (Chamberlain et al., 2021), PDE-GCN (Eliasof et al., 2021) and EGNN (Zhou et al., 2021). We summarize the results in Table 1 where we see better or on par performance with other existing methods. Additionally, we report the accuracy per number of layers, from 2 to 64 in Table 2, where it is evident that our ωGCN and ωGAT do not over-smooth. We report additional results with deeper layers in Appendix H as well as an experiment with 100 random splits in Appendix J where our ωGCN and ωGAT perform similarly to the splits in Table 1.

To further validate the efficacy of our method on fully-supervised node classification, both on homophilic and heterophilic datasets as defined in (Pei et al., 2020). Specifically, examine our ωGCN and ωGAT on Cora, Citeseer, Pubmed, Chameleon (Rozemberczki et al., 2021), Film, Cornell, Texas and Wisconsin using the identical train/validation/test splits of $48\%, 32\%, 20\%$, respectively, and report the average performance over 10 random splits from (Pei et al., 2020). We compare our performance with, GCN, GAT, Geom-GCN (Pei et al., 2020), APPNP (Klicpera et al., 2019), JKNet (Xu et al., 2018), MixHop (Abu-El-Haija et al., 2019), WRGAT(Suresh et al., 2021), GCNII (Chen et al., 2020b), PDE-GCN (Eliasof et al., 2021), GRAND (Chamberlain et al., 2021), GraphCON (Rusch et al., 2022a), MagNet (Zhang et al., 2021), FAGCN (Bo et al., 2021), GPRGNN (Chien et al., 2021), ACMP-GCN (Wang et al., 2022), NSD (Bodnar et al., 2022), H2GCN (Zhu et al., 2020), GGCN (Yan et al., 2021), C&S (Huang et al., 2020), DMP (Yang et al., 2021), LINKX (Lim et al., 2021), ACMII-GCN++ (Luan et al., 2022), and G^2 (Rusch et al., 2022b) as presented in Tables 3-4. Additionally, we evaluate our ωGCN and ωGAT on the Ogbn-arxiv (Hu et al., 2020) dataset, as reported in Table 18 in the Appendix. We see an accuracy improvement across all benchmarks compared to the considered methods. In Appendix K we present the learnt $\vec{\omega}$ for homophilic and heterophilic datasets.

4.2. Inductive Learning

We employ the PPI dataset (Hamilton et al., 2017) for the inductive learning task. We use 8 layer ωGCN and ωGAT, with a learning rate of 0.001, dropout of 0.2 and no weight-decay. As a comparison, we consider several methods and report the micro-averaged F1 score in Table 5. Our ωGCN achieves a score of 99.60, which is higher than the other methods like GAT, JKNet, GeniePath, Cluster-GCN and PDE-GCN.

4.3. Graph Classification

Previous experiments considered the *node-classification* task. To further demonstrate the efficacy of our ωGNNs

[2] We discuss additional backbones in Appendix L.

Table 1. Summary of semi-supervised node classification accuracy (%)

Method	GCN	GAT	APPNP	GCNII	GRAND	superGAT	EGNN	ωGCN (Ours)	ωGAT (Ours)
Cora	81.1	83.1	83.3	85.5	84.7	84.3	85.7	**85.9**	84.8
Citeseer	70.8	70.8	71.8	73.4	73.6	72.6	–	73.3	**74.0**
Pubmed	79.0	78.5	80.1	80.3	71.0	81.7	80.1	81.1	**81.8**

Table 2. Semi-supervised node classification accuracy (%). – indicates not available results.

Dataset	Cora						Citeseer						Pubmed					
Layers	2	4	8	16	32	64	2	4	8	16	32	64	2	4	8	16	32	64
GCN	**81.1**	80.4	69.5	64.9	60.3	28.7	**70.8**	67.6	30.2	18.3	25.0	20.0	**79.0**	76.5	61.2	40.9	22.4	35.3
GCN (Drop)	**82.8**	82.0	75.8	75.7	62.5	49.5	**72.3**	70.6	61.4	57.2	41.6	34.4	**79.6**	79.4	78.1	78.5	77.0	61.5
JKNet	–	80.2	80.7	80.2	**81.1**	71.5	–	68.7	67.7	**69.8**	68.2	63.4	–	78.0	**78.1**	72.6	72.4	74.5
JKNet (Drop)	–	**83.3**	82.6	83.0	82.5	83.2	–	72.6	71.8	**72.6**	70.8	72.2	–	78.7	78.7	**79.7**	79.2	78.9
Incep	–	77.6	76.5	81.7	**81.7**	80.0	–	69.3	68.4	**70.2**	68.0	67.5	–	77.7	**77.9**	74.9	–	–
Incep (Drop)	–	82.9	82.5	83.1	83.1	**83.5**	–	**72.7**	71.4	72.5	72.6	71.0	–	**79.5**	78.6	79.0	–	–
GCNII	82.2	82.6	84.2	84.6	85.4	**85.5**	68.2	68.8	70.6	72.9	**73.4**	73.4	78.2	78.8	79.3	**80.2**	79.8	79.7
GCNII*	80.2	82.3	82.8	83.5	84.9	**85.3**	66.1	66.7	70.6	72.0	**73.2**	73.1	77.7	78.2	78.8	**80.3**	79.8	80.1
PDE-GCN$_{\text{D}}$	82.0	83.6	84.0	84.2	84.3	**84.3**	74.6	75.0	75.2	75.5	**75.6**	75.5	79.3	**80.6**	80.1	80.4	80.2	80.3
EGNN	83.2	–	–	85.4	–	**85.7**	–	–	–	–	–	–	79.2	–	–	80.0	–	**80.1**
ωGCN (Ours)	82.6	83.8	84.2	84.4	85.5	**85.9**	71.3	71.6	72.1	72.4	73.3	**73.3**	79.7	80.2	80.1	80.5	80.8	**81.1**
ωGAT (Ours)	83.4	83.7	84.0	84.3	84.4	**84.8**	72.5	73.1	73.3	73.5	73.9	**74.0**	80.3	81.0	81.2	81.3	81.5	**81.8**

Table 3. Node classification accuracy (%) on *homophilic* datasets. † denotes the maximal accuracy of several proposed variants.

Method Homophily	Cora 0.81	Citeseer 0.80	Pubmed 0.74
GCN	85.77	73.68	88.13
GAT	86.37	74.32	87.62
GCNII†	88.49	77.13	90.30
Geom-GCN†	85.27	77.99	90.05
APPNP	87.87	76.53	89.40
JKNet	85.25	75.85	88.94
WRGAT	88.20	76.81	88.52
PDE-GCN$_{\text{M}}$	88.60	**78.48**	89.93
NSD†	87.14	77.14	89.49
GGCN	87.95	77.14	89.15
H2GCN	87.87	77.11	89.49
C&S	89.05	77.29	90.01
DMP†	86.52	76.87	89.27
LINKX	84.64	73.19	87.86
ACMII-GCN++	88.25	77.12	89.71
ωGCN (Ours)	**89.30**	77.88	90.45
ωGAT (Ours)	89.25	78.01	**90.65**

Table 4. Node classification accuracy (%) on *heterophilic* datasets. † denotes the maximal accuracy of several proposed variants.

Method Homophily	Squirrel 0.22	Film 0.22	Cham. 0.23	Corn. 0.30	Texas 0.11	Wisc. 0.21
GCN	23.96	26.86	28.18	52.70	52.16	48.92
GAT	30.03	28.45	42.93	54.32	58.38	49.41
GCNII	38.47	32.87	60.61	74.86	69.46	74.12
Geom-GCN†	38.32	31.63	60.90	60.81	67.57	64.12
MixHop	43.80	32.22	60.50	73.51	77.84	75.88
PDE-GCN$_{\text{M}}$	–	–	66.01	89.73	93.24	91.76
GRAND	40.05	35.62	54.67	82.16	75.68	79.41
NSD†	56.34	37.79	68.68	86.49	85.95	89.41
WRGAT	48.85	36.53	65.24	81.62	83.62	86.98
MagNet	–	–	–	84.30	83.30	85.70
GGCN	55.17	37.81	71.14	85.68	84.86	86.86
H2GCN	36.48	35.70	60.11	82.70	84.86	87.65
GraphCON†	–	–	–	84.30	85.40	87.80
FAGCN	42.59	34.87	55.22	79.19	82.43	82.94
GPRGNN	31.61	34.63	46.58	80.27	78.38	82.94
DMP†	47.26	35.72	62.28	89.19	89.19	92.16
ACMP-GCN	–	–	–	85.4	86.2	86.1
LINKX	61.81	36.10	68.42	77.84	74.60	75.49
G^{2}†	64.26	37.30	71.40	87.30	87.57	87.84
ACMII-GCN++	**67.40**	37.09	74.76	86.49	88.38	88.43
ωGCN (Ours)	59.41	**38.94**	70.02	91.35	94.05	92.35
ωGAT (Ours)	58.96	38.64	**72.23**	**91.62**	**94.59**	**92.94**

Table 5. Inductive learning on PPI dataset. Results are reported in micro-averaged F1 score.

Method	Micro-averaged F1
GCN (Kipf & Welling, 2017)	60.73
GraphSAGE (Hamilton et al., 2017)	61.20
VR-GCN (Chen et al., 2018)	97.80
GaAN (Zhang et al., 2018a)	98.71
GAT (Veličković et al., 2018)	97.30
JKNet (Xu et al., 2018)	97.60
GeniePath (Liu et al., 2018)	98.50
Cluster-GCN (Chiang et al., 2019)	99.36
GCNII* (Chen et al., 2020b)	99.58
PDE-GCN$_{\mathrm{M}}$ (Eliasof et al., 2021)	99.18
ωGCN (Ours)	**99.60**
ωGAT (Ours)	99.48

Table 6. Graph classification accuracy (%) on TUDatasets (Morris et al., 2020).

Method	MUTAG	PTC	PROTEINS	NCI1	NCI109
PK	76.0 ± 2.7	59.5± 2.4	73.7± 0.7	82.5± 0.5	–
WL Kernel	90.4± 5.7	59.9± 4.3	75.0± 3.1	**86.0± 1.8**	–
DGCNN	85.8±1.8	58.6±2.5	75.5±0.9	74.4±0.5	–
IGN	83.9±13.0	58.5±6.9	76.6±5.5	74.3±2.7	72.8±1.5
PPGNS	90.6±8.7	66.2±6.6	77.2±4.7	83.2±1.1	82.2±1.4
GSN	92.2±7.5	68.2±7.2	76.6±5.0	83.5±2.0	–
SIN	–	–	76.4±3.3	82.7±2.1	–
CIN	92.7±3.6	68.2±3.5	77.0±3.4	83.6±3.1	84.0±3.1
GIN	89.4±5.6	64.6±7.0	76.2±2.8	82.7±1.7	82.2±1.6
GCONV	90.5±4.6	64.9±10.4	73.9±6.1	82.4±2.7	81.7±1.0
RNI	91.0±4.9	64.3±6.1	73.3±3.3	82.1±1.7	81.7±1.0
ωGCN (Ours)	94.6 ± 4.1	73.8 ± 4.3	80.2 ± 2.5	84.1 ± 1.2	**84.5 ± 1.8**
ωGAT (Ours)	**95.2 ± 3.7**	**75.8 ± 3.5**	**80.7 ± 3.7**	84.4 ± 1.7	83.6 ± 1.2

we experiment with graph classification on TUDatasets (Morris et al., 2020). Here, we follow the same experimental settings from (Xu et al., 2019), and report the 10 fold cross-validation performance on MUTAG, PTC, PROTEINS, NCI1 and NCI109 datasets. The hyper-parameters are determined by a grid search, as in (Xu et al., 2019) and are reported in Appendix E. We compare our ωGCN and ωGAT with recent and popular methods like GIN (Xu et al., 2019), GCONV (Morris et al., 2019), RNI (Abboud et al., 2020), DGCNN (Zhang et al., 2018b), IGN (Maron et al., 2018), GSN (Bouritsas et al., 2022), SIN (Bodnar et al., 2021b), CIN (Bodnar et al., 2021a) and others. We also compare with methods that stem from 'classical' graph algorithms like PK (Neumann et al., 2016) and WL Kernel (Shervashidze et al., 2011). All the results are summarized in Table 6, with an evident improvement or similar results to current deep learning as well as classical methods, highlighting the efficacy of our approach.

Table 7. Accuracy (%) of variants of ωGCN on semi-supervised classification.

Data.	Variant	Layers 2	4	8	16	32	64
Cora	ωGCN$_{\mathrm{G}}$	83.4	84.3	84.2	84.1	84.3	84.4
	ωGCN$_{\mathrm{PL}}$	83.0	83.6	84.0	84.2	84.5	84.8
	ωGCN	82.6	83.8	84.2	84.4	85.5	85.9
Cite.	ωGCN$_{\mathrm{G}}$	71.0	71.4	71.3	71.7	72.0	71.8
	ωGCN$_{\mathrm{PL}}$	71.1	71.3	71.5	71.8	72.4	72.6
	ωGCN	71.3	71.6	72.1	72.4	73.3	73.3
Pub.	ωGCN$_{\mathrm{G}}$	79.8	80.4	80.5	80.4	80.2	80.3
	ωGCN$_{\mathrm{PL}}$	79.8	80.0	80.2	80.4	80.5	80.8
	ωGCN	79.7	80.2	80.1	80.5	80.8	81.1

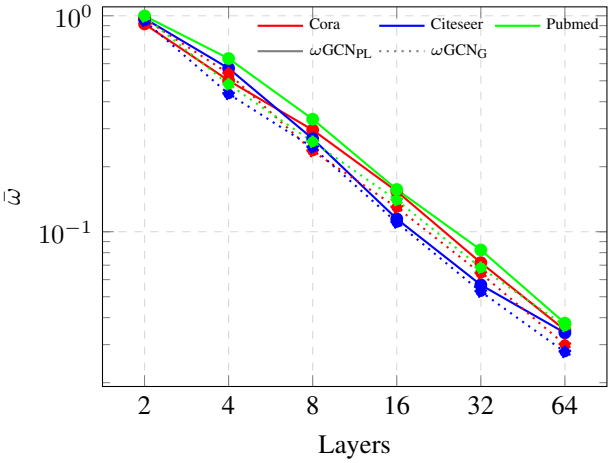

Figure 4. The average value of ω across the layers (denoted by $\bar{\omega} = \frac{1}{L}\sum_{l=0}^{L-1}\omega^{(l)}$) vs. the number layers for ωGCN$_{\mathrm{G}}$ and ωGCN$_{\mathrm{PL}}$. Here, $\bar{\omega}$ scales like $\frac{1}{L}$ for a varying L, in congruence with Theorem 2.1.

4.4. Ablation Study

In this section we study the different components and configurations of our ωGNN. We start by allowing a global (single) ω to be learnt throughout all the layers—this architecture is dubbed as ωGCN$_{\mathrm{G}}$. We validate that this simple variant does not over-smooth, depicted in Table 7. The table also shows ωGCN$_{\mathrm{PL}}$, that includes a single parameter $\omega^{(l)}$ per layer, and ωGCN shown in the results earlier that has $\Omega^{(l)}$, i.e., a parameter per layer and channel, which yields further accuracy improvements. In addition, we empirically verify our theoretical results from Section 2 in Fig. 4, where we show that the obtained values of ω (whether the global or averaged per-layer ones) scale as $1/L$ and behave according to Theorem 2.1 and Corollary 2.2. For completeness, we also perform the ablation study on ωGAT in Appendix I.

5. Summary

In this work we propose an effective and computationally light modification of the large family of GNNs that carries the form of a separable propagation and 1×1 convolutions. In particular, we demonstrate its efficacy on the popular GCN and GAT architectures. Our theorems show that ωGNNs can avoid over-smoothing as their learnable weighting factors $\vec{\omega}$ enable mixing smoothing and sharpening propagation operators. This flexibility also enhances expressiveness. Through an extensive set of experiments on 15 datasets (ranging from node classification to graph classification), an ablation study, and comparisons to several recent methods, we validate our theoretical findings and demonstrate the performance of our ωGNN.

Acknowledgements

The research reported in this paper was supported by grant no. 2018209 from the United States - Israel Binational Science Foundation (BSF), Jerusalem, Israel, and in part by the Israeli Council for Higher Education (CHE) via the Data Science Research Center, Ben-Gurion University of the Negev, Israel. ME is supported by Kreitman High-tech scholarship. LR's work is also partially supported by NSF DMS 2038118, AFOSR grant FA9550- 20-1-0372, and US DOE Office of Advanced Scientific Computing Research Field Work Proposal 20-023231.

References

Abboud, R., Ceylan, I. I., Grohe, M., and Lukasiewicz, T. The surprising power of graph neural networks with random node initialization. *arXiv preprint arXiv:2010.01179*, 2020.

Abu-El-Haija, S., Perozzi, B., Kapoor, A., Alipourfard, N., Lerman, K., Harutyunyan, H., Ver Steeg, G., and Galstyan, A. Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing. In *international conference on machine learning*, pp. 21–29. PMLR, 2019.

Belbute-Peres, F. d. A., Economon, T. D., and Kolter, J. Z. Combining Differentiable PDE Solvers and Graph Neural Networks for Fluid Flow Prediction. In *International Conference on Machine Learning (ICML)*, 2020.

Bo, D., Wang, X., Shi, C., and Shen, H. Beyond low-frequency information in graph convolutional networks. In *AAAI*. AAAI Press, 2021.

Bodnar, C., Frasca, F., Otter, N., Wang, Y., Lio, P., Montufar, G. F., and Bronstein, M. Weisfeiler and lehman go cellular: Cw networks. *Advances in Neural Information Processing Systems*, 34:2625–2640, 2021a.

Bodnar, C., Frasca, F., Wang, Y., Otter, N., Montufar, G. F., Lio, P., and Bronstein, M. Weisfeiler and lehman go topological: Message passing simplicial networks. In *International Conference on Machine Learning*, pp. 1026–1037. PMLR, 2021b.

Bodnar, C., Di Giovanni, F., Chamberlain, B. P., Liò, P., and Bronstein, M. M. Neural sheaf diffusion: A topological perspective on heterophily and oversmoothing in gnns. *arXiv preprint arXiv:2202.04579*, 2022.

Bouritsas, G., Frasca, F., Zafeiriou, S. P., and Bronstein, M. Improving graph neural network expressivity via subgraph isomorphism counting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

Brody, S., Alon, U., and Yahav, E. How attentive are graph attention networks? In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=F72ximsx7C1.

Cai, C. and Wang, Y. A note on over-smoothing for graph neural networks. *arXiv preprint arXiv:2006.13318*, 2020.

Chamberlain, B. P., Rowbottom, J., Gorinova, M., Webb, S., Rossi, E., and Bronstein, M. M. Grand: Graph neural diffusion. *arXiv preprint arXiv:2106.10934*, 2021.

Chen, D., Lin, Y., Li, W., Li, P., Zhou, J., and Sun, X. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34:3438–3445, 04 2020a. doi: 10.1609/aaai.v34i04.5747.

Chen, J., Zhu, J., and Song, L. Stochastic training of graph convolutional networks with variance reduction. In Dy, J. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 942–950. PMLR, 10–15 Jul 2018. URL http://proceedings.mlr.press/v80/chen18p.html.

Chen, L.-C., Papandreou, G., Kokkinos, I., Murphy, K., and Yuille, A. L. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):834–848, 2017.

Chen, M., Wei, Z., Huang, Z., Ding, B., and Li, Y. Simple and deep graph convolutional networks. In III, H. D. and Singh, A. (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 1725–1735. PMLR, 13–18 Jul 2020b. URL http://proceedings.mlr.press/v119/chen20v.html.

Chiang, W.-L., Liu, X., Si, S., Li, Y., Bengio, S., and Hsieh, C.-J. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, 2019. URL https://arxiv.org/pdf/1905.07953.pdf.

Chien, E., Peng, J., Li, P., and Milenkovic, O. Adaptive universal generalized pagerank graph neural network. In *International Conference on Learning Representations*, 2021.

Corso, G., Cavalleri, L., Beaini, D., Liò, P., and Veličković, P. Principal neighbourhood aggregation for graph nets. *Advances in Neural Information Processing Systems*, 33: 13260–13271, 2020.

Defferrard, M., Bresson, X., and Vandergheynst, P. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems*, pp. 3844–3852, 2016.

Eliasof, M. and Treister, E. Diffgcn: Graph convolutional networks via differential operators and algebraic multigrid pooling. *34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.*, 2020.

Eliasof, M., Haber, E., and Treister, E. PDE-GCN: Novel architectures for graph neural networks motivated by partial differential equations. *Advances in Neural Information Processing Systems*, 34:3836–3849, 2021.

Eliasof, M., Boesen, T., Haber, E., Keasar, C., and Treister, E. Mimetic neural networks: A unified framework for protein design and folding. *Frontiers in Bioinformatics*, 2:39, 2022a.

Eliasof, M., Haber, E., and Treister, E. pathgcn: Learning general graph spatial operators from paths. In *International Conference on Machine Learning*, pp. 5878–5891. PMLR, 2022b.

Fey, M. and Lenssen, J. E. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

Gasteiger, J., Weißenberger, S., and Günnemann, S. Diffusion improves graph learning. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2019.

Glorot, X. and Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256. JMLR Workshop and Conference Proceedings, 2010.

Hamilton, W. L., Ying, R., and Leskovec, J. Inductive representation learning on large graphs. In *NIPS*, 2017.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.

He, M., Wei, Z., Xu, H., et al. Bernnet: Learning arbitrary graph spectral filters via bernstein approximation. *Advances in Neural Information Processing Systems*, 34: 14239–14251, 2021a.

He, T., Ong, Y. S., and Bai, L. Learning conjoint attentions for graph neural nets. *Advances in Neural Information Processing Systems*, 34:2641–2653, 2021b.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. MobileNets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

Hu, W., Fey, M., Zitnik, M., Dong, Y., Ren, H., Liu, B., Catasta, M., and Leskovec, J. Open graph benchmark: Datasets for machine learning on graphs. *arXiv preprint arXiv:2005.00687*, 2020.

Huang, Q., He, H., Singh, A., Lim, S.-N., and Benson, A. R. Combining label propagation and simple models out-performs graph neural networks. *arXiv preprint arXiv:2010.13993*, 2020.

Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., Tunyasuvunakool, K., Bates, R., Žídek, A., Potapenko, A., Bridgland, A., Meyer, C., Kohl, S. A. A., Ballard, A. J., Cowie, A., Romera-Paredes, B., Nikolov, S., Jain, R., Adler, J., Back, T., Petersen, S., Reiman, D., Clancy, E., Zielinski, M., Steinegger, M., Pacholska, M., Berghammer, T., Silver, D., Vinyals, O., Senior, A. W., Kavukcuoglu, K., Kohli, P., and Hassabis, D. Applying and improving alphafold at casp14. *Proteins*, 2021. ISSN 1097-0134. doi: 10.1002/prot.26257. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/prot.26257.

Kim, D. and Oh, A. How to find your friendly neighborhood: Graph attention design with self-supervision. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=Wi5KUNlqWty.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. *International Conference on Learning Representations (ICLR)*, 2017.

Klicpera, J., Bojchevski, A., and Günnemann, S. Combining neural networks with personalized pagerank for classification on graphs. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=H1gL-2A9Ym.

Krizhevsky, A., Sutskever, I., and Hinton, G. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 61: 1097–1105, 2012.

Li, Q., Han, Z., and Wu, X.-M. Deeper insights into graph convolutional networks for semi-supervised learning. In *Thirty-Second AAAI conference on artificial intelligence*, 2018.

Lim, D., Hohne, F. M., Li, X., Huang, S. L., Gupta, V., Bhalerao, O. P., and Lim, S.-N. Large scale learning on non-homophilous graphs: New benchmarks and strong simple methods. In Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W. (eds.), *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=DfGu8WwT0d.

Liu, Z., Chen, C., Li, L., Zhou, J., Li, X., and Song, L. Geniepath: Graph neural networks with adaptive receptive paths. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33, 02 2018. doi: 10.1609/aaai.v33i01.33014424.

Liu, Z., Mao, H., Wu, C.-Y., Feichtenhofer, C., Darrell, T., and Xie, S. A convnet for the 2020s. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11976–11986, 2022.

Luan, S., Zhao, M., Hua, C., Chang, X.-W., and Precup, D. Complete the missing half: Augmenting aggregation filtering with diversification for graph convolutional networks. *arXiv preprint arXiv:2008.08844*, 2020.

Luan, S., Hua, C., Lu, Q., Zhu, J., Zhao, M., Zhang, S., Chang, X.-W., and Precup, D. Revisiting heterophily for graph neural networks. *Conference on Neural Information Processing Systems*, 2022.

Maron, H., Ben-Hamu, H., Shamir, N., and Lipman, Y. Invariant and equivariant graph networks. *ICLR*, 2018.

Min, Y., Wenkel, F., and Wolf, G. Scattering gcn: Overcoming oversmoothness in graph convolutional networks. *Advances in Neural Information Processing Systems*, 33: 14498–14508, 2020.

Monti, F., Boscaini, D., Masci, J., Rodola, E., Svoboda, J., and Bronstein, M. M. Geometric deep learning on graphs and manifolds using mixture model cnns. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5115–5124, 2017.

Morris, C., Ritzert, M., Fey, M., Hamilton, W. L., Lenssen, J. E., Rattan, G., and Grohe, M. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pp. 4602–4609, 2019.

Morris, C., Kriege, N. M., Bause, F., Kersting, K., Mutzel, P., and Neumann, M. Tudataset: A collection of benchmark datasets for learning with graphs. In *ICML 2020 Workshop on Graph Representation Learning and Beyond (GRL+ 2020)*, 2020. URL www.graphlearning.io.

Neumann, M., Garnett, R., Bauckhage, C., and Kersting, K. Propagation kernels: efficient graph kernels from propagated information. *Machine Learning*, 102(2):209–245, 2016.

Nt, H. and Maehara, T. Revisiting graph neural networks: All we have is low-pass filters. *arXiv preprint arXiv:1905.09550*, 2019.

Oono, K. and Suzuki, T. Graph neural networks exponentially lose expressive power for node classification. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=S1ldO2EFPr.

Page, L., Brin, S., Motwani, R., and Winograd, T. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019.

Pei, H., Wei, B., Chang, K. C.-C., Lei, Y., and Yang, B. Geom-gcn: Geometric graph convolutional networks. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=S1e2agrFvS.

Rong, Y., Huang, W., Xu, T., and Huang, J. Dropedge: Towards deep graph convolutional networks on node classification. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=Hkx1qkrKPr.

Rozemberczki, B., Allen, C., and Sarkar, R. Multi-Scale Attributed Node Embedding. *Journal of Complex Networks*, 9(2), 2021.

Rusch, T. K., Chamberlain, B., Rowbottom, J., Mishra, S., and Bronstein, M. Graph-coupled oscillator networks. In *International Conference on Machine Learning*, pp. 18888–18909. PMLR, 2022a.

Rusch, T. K., Chamberlain, B. P., Mahoney, M. W., Bronstein, M. M., and Mishra, S. Gradient gating for deep multi-rate learning on graphs. *arXiv preprint arXiv:2210.00513*, 2022b.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. MobileNetV2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4510–4520, 2018.

Shervashidze, N., Schweitzer, P., Van Leeuwen, E. J., Mehlhorn, K., and Borgwardt, K. M. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(9), 2011.

Suresh, S., Budde, V., Neville, J., Li, P., and Ma, J. Breaking the limit of graph neural networks by improving the assortativity of graphs with local mixing patterns. *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 2021.

Szegedy, C., Ioffe, S., Vanhoucke, V., and Alemi, A. A. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

Tan, M. and Le, Q. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pp. 6105–6114. PMLR, 2019.

Thorpe, M., Nguyen, T. M., Xia, H., Strohmer, T., Bertozzi, A., Osher, S., and Wang, B. GRAND++: Graph neural diffusion with a source term. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=EMxu-dzvJk.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. Graph Attention Networks. *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=rJXMpikCZ.

Wang, G., Ying, R., Huang, J., and Leskovec, J. Improving graph attention networks with large margin-based constraints. *arXiv preprint arXiv:1910.11945*, 2019.

Wang, X. and Zhang, M. How powerful are spectral graph neural networks. *International Conference on Machine Learning (ICML)*, 2022.

Wang, Y., Sun, Y., Liu, Z., Sarma, S. E., Bronstein, M. M., and Solomon, J. M. Dynamic graph cnn for learning on point clouds. *arXiv preprint arXiv:1801.07829*, 2018.

Wang, Y., Yi, K., Liu, X., Wang, Y. G., and Jin, S. Acmp: Allen-cahn message passing for graph neural networks with particle phase transition. *arXiv preprint arXiv:2206.05437*, 2022.

Williamson, D. P. Lecture notes in spectral graph theory course. https://people.orie.cornell.edu/dpw/orie6334/Fall2016/lecture7.pdf, 2016.

Wu, F., Souza, A., Zhang, T., Fifty, C., Yu, T., and Weinberger, K. Simplifying graph convolutional networks. In *International conference on machine learning*, pp. 6861–6871. PMLR, 2019.

Xu, K., Li, C., Tian, Y., Sonobe, T., Kawarabayashi, K.-i., and Jegelka, S. Representation learning on graphs with jumping knowledge networks. In Dy, J. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 5453–5462. PMLR, 10–15 Jul 2018. URL http://proceedings.mlr.press/v80/xu18c.html.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=ryGs6iA5Km.

Yan, Y., Hashemi, M., Swersky, K., Yang, Y., and Koutra, D. Two sides of the same coin: Heterophily and oversmoothing in graph convolutional neural networks. *arXiv preprint arXiv:2102.06462*, 2021.

Yang, L., Li, M., Liu, L., Wang, C., Cao, X., Guo, Y., et al. Diverse message passing for attribute with heterophily. *Advances in Neural Information Processing Systems*, 34: 4751–4763, 2021.

Yang, Z., Cohen, W., and Salakhudinov, R. Revisiting semi-supervised learning with graph embeddings. In *International conference on machine learning*, pp. 40–48. PMLR, 2016.

Zhang, J., Shi, X., Xie, J., Ma, H., King, I., and Yeung, D. Gaan: Gated attention networks for learning on large and spatiotemporal graphs. In *Proceedings of the Thirty-Fourth Conference on Uncertainty in Artificial Intelligence*, pp. 339–349, 2018a.

Zhang, M., Cui, Z., Neumann, M., and Chen, Y. An end-to-end deep learning architecture for graph classification. In *Thirty-second AAAI conference on artificial intelligence*, 2018b.

Zhang, X., He, Y., Brugnone, N., Perlmutter, M., and Hirn, M. Magnet: A neural network for directed graphs. *Advances in Neural Information Processing Systems*, 34: 27003–27015, 2021.

Zhao, J., Dong, Y., Ding, M., Kharlamov, E., and Tang, J. Adaptive diffusion in graph neural networks. In Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W. (eds.), *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=0Kb33DHJ1g.

Zhao, L. and Akoglu, L. Pairnorm: Tackling oversmoothing in gnns. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=rkecl1rtwB.

Zhou, K., Huang, X., Zha, D., Chen, R., Li, L., Choi, S.-H., and Hu, X. Dirichlet energy constrained learning for deep graph neural networks. *Advances in Neural Information Processing Systems*, 34, 2021.

Zhu, J., Yan, Y., Zhao, L., Heimann, M., Akoglu, L., and Koutra, D. Beyond homophily in graph neural networks: Current limitations and effective designs. *Advances in Neural Information Processing Systems*, 33:7793–7804, 2020.

A. Proofs of Theorems

Here we repeat the theorems, observations and corollaries from the main paper, for convenience, and provide their proofs or derivation.

$\tilde{\mathbf{P}}$ is a Scaled Diffusion Operator

Assume that $\mathbf{A}$ is the adjacency matrix, and $\mathbf{D}$ is the degree matrix. Denote the adjacency matrix with added self-loops by $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$. Then, the convolution operator from GCN (Kipf & Welling, 2017) is

$$\tilde{\mathbf{P}} = \tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}} \tag{14}$$

We first note that the Laplacian including self-loops is the same as the regular Laplacian:

$$\tilde{\mathbf{L}} = \tilde{\mathbf{D}} - \tilde{\mathbf{A}} = \mathbf{D} + \mathbf{I} - \mathbf{A} - \mathbf{I} = \mathbf{D} - \mathbf{A} = \mathbf{L}. \tag{15}$$

Therefore, it holds that:

$$\begin{aligned}
\tilde{\mathbf{P}} &= \mathbf{I} - \mathbf{I} + \tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}} \\
&= \mathbf{I} - \tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{D}}\tilde{\mathbf{D}}^{-\frac{1}{2}} + \tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}} \\
&= \mathbf{I} - \tilde{\mathbf{D}}^{-\frac{1}{2}}(\tilde{\mathbf{D}} - \tilde{\mathbf{A}})\tilde{\mathbf{D}}^{-\frac{1}{2}} \\
&= \mathbf{I} - \tilde{\mathbf{D}}^{-\frac{1}{2}}(\mathbf{D} - \mathbf{A})\tilde{\mathbf{D}}^{-\frac{1}{2}} \\
&= \mathbf{I} - \tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{L}\tilde{\mathbf{D}}^{-\frac{1}{2}}.
\end{aligned} \tag{16}$$

Proof of Theorem 2.1

Proof. First, note that (6) from the main paper can be written as

$$E(\mathbf{f}^{(l)}) = \sum_{i\in\mathcal{V}}\sum_{j\in\mathcal{N}_i}\frac{1}{2}\left\|\frac{\mathbf{f}_i^{(l)}}{\sqrt{(1+d_i)}} - \frac{\mathbf{f}_j^{(l)}}{\sqrt{(1+d_j)}}\right\|_2^2 = \tfrac{1}{2}\|\mathbf{G}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{f}^{(l)}\|_2^2, \tag{17}$$

where $\mathbf{G}$ is the graph gradient operator, also known as the incidence matrix, that for each edge subtracts the features of the two connected nodes, i.e., $\mathbf{G}\mathbf{f}^{(l)}_{(i,j)} = \mathbf{f}_i^{(l)} - \mathbf{f}_j^{(l)}$ for $(i,j) \in \mathcal{E}$. Let us assume that the initial feature $\mathbf{f}^{(0)}$ has some Dirichlet energy $E_0 > E_{opt}$ as defined in (17). Since

$$\nabla E = \tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{G}^\top\mathbf{G}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{f}^{(l)}$$

we see that the forward propagation through a GCN approximates the gradient flow of the Dirichlet energy. That is, for given L and and ω we have that

$$\mathbf{f}^{(l+1)} = \mathbf{f}^{(l)} - \omega\nabla E = \mathbf{f}^{(l)} - \omega\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{G}^\top\mathbf{G}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{f}^{(l)} = (\mathbf{I} - \omega\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{L}\tilde{\mathbf{D}}^{-\frac{1}{2}})\mathbf{f}^{(l)} \tag{18}$$

where we used that $\mathbf{G}^\top\mathbf{G} = \mathbf{L}$. Equation 18 can be seen both as a gradient descent step to reduce E, and also as a forward Euler approximation with step size ω of the solution of

$$\frac{\partial\mathbf{f}(t)}{\partial t} = -\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{L}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{f}(t), \quad \mathbf{f}(0) = \mathbf{f}^{(0)}. \tag{19}$$

It is known that the solution to (19) is given by

$$\mathbf{f}(t) = \exp\left(-t\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{L}\tilde{\mathbf{D}}^{-\frac{1}{2}}\right)\mathbf{f}(0). \tag{20}$$

Since the Dirichlet energy of $\mathbf{f}(t)$ is continuous in t and decays monotonically from E_0 to zero, there exists a T such that $E(\mathbf{f}(T)) = E^*$. Now, considering discrete time intervals $0 = t_0, ..., t_L = T$, then, similarly to (20), for any two subsequent time steps t_{l+1} and t_l we have that

$$\mathbf{f}(t_{l+1}) = \exp\left(-(t_{l+1} - t_l)\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{L}\tilde{\mathbf{D}}^{-\frac{1}{2}}\right)\mathbf{f}(t_l). \tag{21}$$

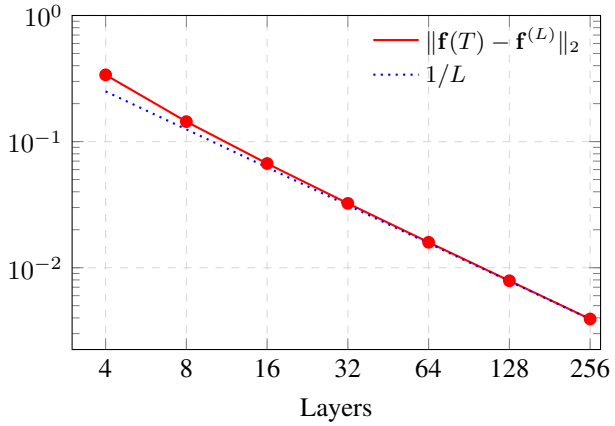

Figure 5. Difference norm $\|\mathbf{f}(T) - \mathbf{f}^{(L)}\|_2$ between an analytical solution $\mathbf{f}(t) = \exp(-t\mathbf{A})\,\mathbf{f}(0)$ like in (20) and a propagated solution $\mathbf{f}^{(l+1)} = (\mathbf{I} - \omega\mathbf{A})\mathbf{f}^{(l)}$ like in (18) using a random and diagonally normalized symmetric positive definite matrix $\mathbf{A} \in \mathbb{R}^{100\times100}$, and a random initial feature $\mathbf{f}(0)$. The integration goes from $t = 0$ to $T = 1$, and hence $\omega = 1/L$. It is clear that the difference norm at the last layer scales like $1/L$, as expected.

Taking fixed-interval time steps such that $t_{l+1} - t_l = \omega = T/L$ for $l = 0, ..., L$, we get

$$\mathbf{f}(t_{l+1}) = \exp\left(-\omega\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{L}\tilde{\mathbf{D}}^{-\frac{1}{2}}\right)\mathbf{f}(t_l) = (\mathbf{I} - \omega\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{L}\tilde{\mathbf{D}}^{-\frac{1}{2}})\mathbf{f}(t_l) + O(\omega^2), \tag{22}$$

where the rightmost approximation holds due to the Taylor expansion, up to first-order approximation. Denoting $\mathbf{f}^{(l)} = \mathbf{f}(t_l)$ and $\bar{\omega} = T$, we complete the proof. □

Remark 1. *At the basis of our analysis above there is the analytical solution in (20), which, as shown in Eq. (22), is $O(\omega^2)$ different than the propagated solution through (18). After L layers, the $O(\omega^2)$ term may accumulate L times. Since $\omega = T/L$ where T is fixed, then $O(L\omega^2)$ is equivalent to $O(\omega)$, resulting in a first order approximation to the analytical solution in (20). This is often referred to as forward Euler integration. To demonstrate and verify this, we perform a small experiment with a random and diagonally normalized symmetric positive definite matrix $\mathbf{A}$ (in the role of symmetric normalized Laplacian). See Fig. 5 for details. Indeed, the difference between the analytical and propagated solutions at the last layer scales as ω or $1/L$ where L is the number of layers.*

Proof of Corollary 2.2

Proof. The proof follows immediately by setting variable $t_{l+1} - t_l = \omega^{(l)}$ and placing in (22). □

Remark 2 (The non-negativity of $\tilde{\mathbf{P}}_\omega$)**.** *By definition, for $0 < \omega \leq 1$ all the spatial weights of $\tilde{\mathbf{P}}_\omega$ defined in (7) are non-negative, and it is that the operator is smoothing as it is a low-pass filter. For $\omega > 1$ or $\omega < 0$, by definition we have an operator with mixed signs.*

Proof of Theorem 2.3

Proof. Assuming that the graph is connected, it is known that the graph Laplacian matrix has the eigenvector $\mathbf{1}$ whose eigenvalue is 0, i.e. $\mathbf{L}\mathbf{1} = 0$. Hence, we get that $\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{L}\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{D}}^{\frac{1}{2}}\mathbf{1} = 0$ so $\tilde{\mathbf{D}}^{\frac{1}{2}}\mathbf{1}$ is the eigenvector of the normalized Laplacian with eigenvalue of 0.

Furthermore, denote the normalized Laplacian by $\tilde{\mathbf{L}} = \tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{L}\tilde{\mathbf{D}}^{-\frac{1}{2}}$. Consider the range

$$0 < \omega < \frac{2}{\rho(\tilde{\mathbf{L}})} = \omega_0,$$

where $\rho(\tilde{\mathbf{L}})$ denotes the spectral radius of the matrix $\tilde{\mathbf{L}}$. It is easy to verify that for this range of values for ω, the largest eigenvalue in magnitude of $\tilde{\mathbf{P}}_\omega$ is 1, and it corresponds to the null eigenvector of $\tilde{\mathbf{L}}$, i.e., $\tilde{\mathbf{D}}^{\frac{1}{2}}\mathbf{1}$. Hence, for this range, $\tilde{\mathbf{P}}$ is

smoothing. For $\omega > \omega_0$ and $\omega < 0$, the leading eigenvector of $\tilde{\mathbf{P}}_\omega$ becomes the leading eigenvector of $\tilde{\mathbf{L}}$. Furthermore, it can be shown that $\rho(\tilde{\mathbf{L}}) \leq 2$ (see (Williamson, 2016) for the proof), hence $\omega_0 \geq 1$. □

B. Synthetic Expressiveness Task

To demonstrate the importance and benefit of learning sharpening propagation operators in addition to smoothing operators, we propose the following synthetic node gradient regression task. Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with some input node features $\mathbf{f}^{in} \in \mathbb{R}^{n \times c_{in}}$, we wish a GNN to regress the node features gradient, $\nabla \mathbf{f}^{in}$, where the node feature gradient of the i-th node is defined as an upwind gradient operator:

$$\nabla \mathbf{f}_i^{in} = \max_{\mathrm{j} \in \mathcal{N}_\mathrm{i}} (\mathbf{f}_\mathrm{i} - \mathbf{f}_\mathrm{j}), \tag{23}$$

where the goal of the considered GNN is to minimize the following objective:

$$\|\mathrm{GNN}(\mathbf{f}^{\mathrm{in}}, \mathcal{G}) - \nabla \mathbf{f}^{\mathrm{in}}\|_2^2. \tag{24}$$

As a comparison, we consider two GNNs: GCN (Kipf & Welling, 2017) and our ωGCN with 64 channels and 2 layers. In both cases we use a learning rate of $1e-4$ without weight decay and train the network for 5000 iterations (no further benefit was obtained with any of the considered methods). The input graph is a random Erdős–Rényi graph with 8 nodes and an edge rate of 30%, with input node features sampled from a uniform distribution in the range of 0 to 1. The obtained loss of GCN is of order $1e-1$, while our ωGCN obtains a loss of order $1e-12$, also as can be seen in Fig. 2. We therefore conclude that introducing the ability to learn mixed-sign operators by ω is beneficial to enhance the expressiveness of GNNs.

C. Datasets

In this section we provide the statistics of the datasets used throughout our experiments. Table 8 presents information regarding node-classification datasets, and Table 9 summarizes the graph-classification datasets. For each dataset, we also provide the homophily score as defined by (Pei et al., 2020).

Table 8. Node classification datasets statistics. Hom. score denotes the homophily score.

Dataset	Cora	Citeseer	Pubmed	Chameleon	Film	Cornell	Texas	Wisconsin	PPI	Ogbn-arxiv
Classes	7	6	3	5	5	5	5	5	121	40
Nodes	2,708	3,327	19,717	2,277	7,600	183	183	251	56,944	169,343
Edges	5,429	4,732	44,338	36,101	33,544	295	309	499	818,716	1,116,243
Features	1,433	3,703	500	2,325	932	1,703	1,703	1,703	50	128
Hom. score	0.81	0.80	0.74	0.23	0.22	0.30	0.11	0.21	0.17	0.63

Table 9. TUDatasets graph classification statistics.

Dataset	MUTAG	PTC	PROTEINS	NCI1	NCI109
Classes	2	2	2	2	2
Graphs	188	344	1113	4110	4127
Avg. nodes	17.93	14.29	39.06	29.87	32.13
Avg. edges	19.79	14.69	72.82	32.30	32.13

D. Over-smoothing in GAT

In addition to the observation presented in Section 2.1 and specifically in Fig. 3 where we see that recurrent applications of GAT reduce the node feature energy from (13), which causes over-smoothing as shown by (Wu et al., 2019; Wang et al., 2019) (as discussed in the main paper), here, we also show that the same behaviour is evident with Citeseer and Pubmed datasets in Fig. 6.

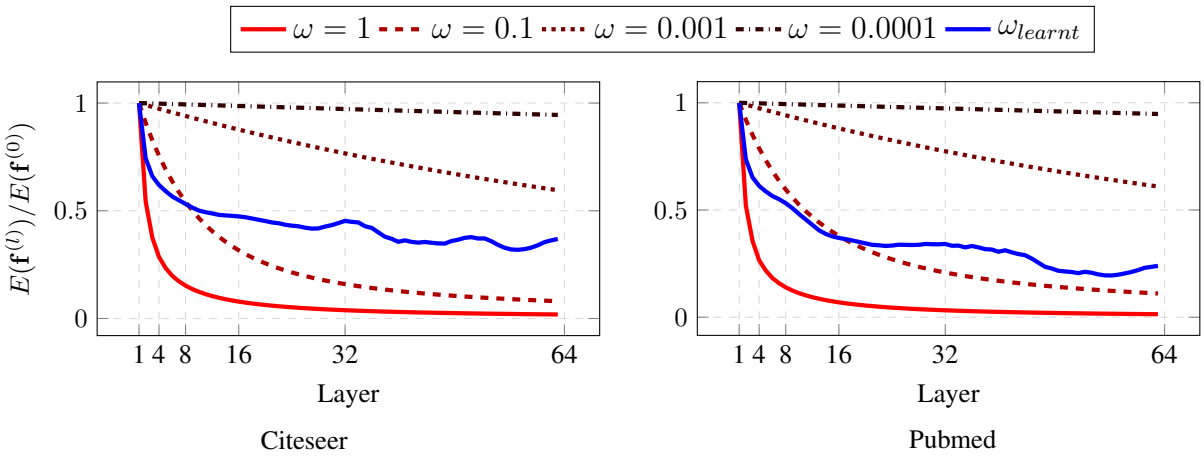

Figure 6. Node features energy at the l-th layer relative to the initial node embedding energy on Citeseer and Pubmed. ωGAT controls the energy from Eq. (13) to avoid over-smoothing, while the baseline GAT with $\omega = 1$ reduce the energy to 0 and over-smooth.

E. Architectures in Details

We now elaborate on the specific architectures used in our experiments in Section 4. As noted in the main paper, all our network architectures consist of an opening (embedding) layer (1×1 convolution), a sequence of ωGNN (i.e., ωGCN or ωGAT) layers, and a closing (classifier) layer (1×1 convolution). In total, we have two types of architectures – one that is based on GCN, for node classification tasks reported in Table 10, and the other for the graph classification task which is based on (Xu et al., 2019) and is reported in Table 11. Throughout the following, we denote by c_{in} and c_{out} the input and output channels, respectively, and c denotes the number of features in hidden layers (which is reported in Appendix F). We initialize the embedding and classifier layers with the Glorot (Glorot & Bengio, 2010) initialization, and $\mathbf{K}^{(l)}$ from (2) is initialized with an identity matrix of shape $c \times c$. The initialization of $\Omega^{(l)}$ also starts from a vector of ones. We note that our initialization yields a standard smoothing process, which is then adapted to the data as the learning process progresses, and if needed also changes the process to a non-smoothing one by the means of mixed-signs, as discussed earlier and specifically in Theorem. 2.3. We denote the number of ωGNN layers by L, and the dropout probability by p. The main differences between the two architectures are as follows. First, for the graph classification we use the standard add-pool operation as in GIN (Xu et al., 2019) to obtain a global graph feature. Second, we follow GIN and in addition to the graph layer (which is ωGNN in our work), we add batch normalization (denoted by BN), 1×1 convolution and a ReLU activation past each graph layer.

Table 10. The architecture used for node classification and inductive learning.

Input size	Layer	Output size
$n \times c_{in}$	1×1 Dropout(p)	$n \times c_{in}$
$n \times c_{in}$	1×1 Convolution	$n \times c$
$n \times c$	ReLU	$n \times c$
$n \times c$	$L\times$ ωGNN layers	$n \times c$
$n \times c$	Dropout(p)	$n \times c$
$n \times c$	1×1 Convolution	$n \times c_{out}$

F. Hyper-parameters Details

We provide the selected hyper-parameters in our experiments. We denote the learning rate of our ωGNN layers by LR_{GNN}, and the learning rate of the 1×1 opening and closing as well as any additional classifier layers by LR_{oc}. Also, the weight decay for the opening and closing layers is denoted by WD_{oc}. We denote the ω parameter learning rate and weight decay by LR_{ω} and WD_{ω}, respectively. c denotes the number of hidden channels. In the case of ωGAT, the attention head vector $\mathbf{a}$ are learnt with the same learning rate as LR_{GNN} and WD_{GNN}.

Table 11. The architecture used for graph classification.

Input size	Layer	Output size
$n \times c_{in}$	1×1 Convolution	$n \times c$
$n \times c$	ReLU	$n \times c$
$n \times c$	$L \times [\, \omega$GNN , BN, 1×1 Convolution, ReLU $]$	$n \times c$
$n \times c$	1×1 Add-pool	$1 \times c$
$1 \times c$	1×1 Convolution	$1 \times c$
$1 \times c$	1×1 Dropout(p)	$1 \times c$
$1 \times c$	1×1 Convolution	$1 \times c_{out}$

F.1. Semi-supervised Node Classification

The hyper-parameters for this experiment are summarized in Table 12.

Table 12. Semi-supervised node classification hyper-parameters.

Architecture	Dataset	LR_{GNN}	LR_{oc}	LR_{ω}	WD_{GNN}	WD_{oc}	WD_{ω}	c	p
ωGCN	Cora	0.01	0.01	0.01	1e-4	8e-5	2e-4	64	0.6
	Citeseer	1e-4	0.005	0.005	1e-5	5e-6	2e-4	256	0.7
	Pubmed	0.001	5e-4	0.005	2e-4	1e-4	1e-4	256	0.5
ωGAT	Cora	0.01	0.01	0.005	1e-5	1e-5	1e-5	64	0.6
	Citeseer	0.005	0.005	0.001	1e-4	1e-5	1e-4	256	0.7
	Pubmed	0.005	0.001	0.05	4e-5	1e-5	1e-4	256	0.5

F.2. Full-supervised Node Classification

The hyper-parameters for this experiment are summarized in Table 13. For Ogbn-arxiv from Table 18, 8 layer ωGCN and ωGAT were employed

F.3. Inductive Learning

The hyper-parameters for the inductive learning on PPI are listed in Section 4.2 in the main paper, and are the same for ωGCN and ωGAT.

F.4. Graph Classification

The hyper-parameters for the graph classification experiment on TUDatasets are reported in Table 14. We followed the same grid-search procedure as in GIN (Xu et al., 2019). In all experiments, a 5 layer (including the initial embedding layer) ωGCN and ωGAT are used, similarly to GIN.

F.5. Ablation Study

In this experiment we used the same hyper-parameters as reported in Table 12.

G. Runtimes

Following the computational cost discussion from Section 2.4 in the main paper, we also present in Table 15 the measured training and inference times of our baselines GCN and GAT with 2 layers, where we see that indeed the addition of ω per layer and channel requires a negligible addition of time, at the return of a significantly more accurate GNN. We note that further accuracy gain can be achieved when adding more ωGNN layers as reported in Table 2 in the main paper. However, since GCN and GAT over-smooth, the comparison here is done with 2 layers, where the highest accuracy is obtained for the baseline models.

Table 13. Full-supervised node classification hyper-parameters.

Architecture	Dataset	LR_{GNN}	LR_{oc}	LR_{ω}	WD_{GNN}	WD_{oc}	WD_{ω}	c	p
ωGCN	Cora	0.01	0.05	0.005	0.01	1e-4	1e-4	64	0.5
	Citeseer	0.001	0.08	0.005	0.005	1e-4	0	64	0.5
	Pubmed	0.005	0.005	0.01	0.003	5e-5	0.01	64	0.5
	Chameleon	1e-4	0.005	5e-4	1e-4	1e-4	1e-5	64	0.5
	Film	0.05	0.01	0.05	1e-4	1e-4	1e-5	64	0.5
	Cornell	0.01	0.05	0.01	0.005	1e-4	0	64	0.5
	Texas	0.08	0.08	0.005	0.005	5e-4	0	64	0.5
	Wisconsin	0.001	0.05	0.005	1e-4	3e-4	3e-4	64	0.5
	Ogbn-arxiv	0.01	0.01	0.01	0	0	0	256	0
ωGAT	Cora	0.001	0.01	0.05	0.001	5e-4	0	64	0.5
	Citeseer	0.005	0.05	0.03	0.005	5e-4	0.001	64	0.5
	Pubmed	0.05	0.005	0.005	0.003	1e-6	0.003	64	0.5
	Chameleon	0.005	0.005	3e-4	5e-4	5e-4	1e-5		
	Film	0.05	0.01	0.01	5e-4	0.001	4e-4	64	0.5
	Cornell	0.001	0.01	0.005	1e-4	1e-5	0	64	0.5
	Texas	1e-4	0.02	0.05	5e-4	5e-4	0	64	0.5
	Wisconsin	0.01	0.05	0.005	0.001	5e-4	0	64	0.5
	Ogbn-arxiv	0.01	0.01	0.01	0	0	0	256	0

Table 14. Graph classification hyper-parameters. BS denoted batch size.

Architecture	Dataset	LR_{GNN}	LR_{oc}	LR_{ω}	WD_{GNN}	WD_{oc}	WD_{ω}	c	p	BS
ωGCN	MUTAG	0.01	0.01	0.01	0	0	0	32	0	32
	PTC	0.01	0.01	0.01	0	0	0	32	0	32
	PROTEINS	0.01	0.01	0.01	0	0	0	32	0	128
	NCI1	0.01	0.01	0.01	0	0	0	32	0.5	32
	NCI109	0.01	0.01	0.01	0	0	0	32	0	32
ωGAT	MUTAG	0.01	0.01	0.01	0	0	0	32	0	32
	PTC	0.01	0.01	0.01	0	0	0	32	0	128
	PROTEINS	0.01	0.01	0.01	0	0	0	32	0	128
	NCI1	0.01	0.01	0.01	0	0	0	32	0.5	128
	NCI109	0.01	0.01	0.01	0	0	0	32	0.5	32

H. Depth Study

In addition to the results in Section 4.1, we now provide results with up to 256 layers, on the Cora, Citeseer, Pubmed, Chameleon, and Cornell datasets, to demonstrate that our ωGCN does not over-smooth, in Figure 7.

I. Ablation Study using ωGAT

To complement our ablation study on ωGCN in Section 4.4 in the main paper, we perform a similar study on ωGAT. Here, we show in Table 16, that indeed the single ω variant, dubbed $\omega\text{GAT}_{\text{G}}$ does not over-smooth, and that by allowing the greater flexibility of a per-layer and per layer and channel of our $\omega\text{GAT}_{\text{PL}}$ and ωGAT, respectively, better performance is obtained.

J. Statistical Significance of Semi-supervised Node Classification Results

Throughout our semi-supervised node classification experiment in Section 4 on Cora, Citeseer and Pubmed, the standard split from (Kipf & Welling, 2017) was considered, to a direct comparison with as many as possible methods. However, since

Table 15. Training and inference GPU runtimes [ms] on Cora.

Runtime	GCN	GAT	ωGCN (Ours)	ωGAT (Ours)
Training	7.71	14.59	7.79	14.95
Inference	1.75	2.98	1.88	3.09
Accuracy (%)	81.1	83.1	82.6	83.4

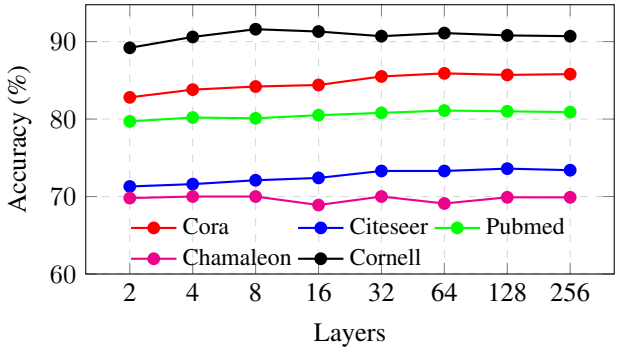

Figure 7. The obtained node classification accuracy (%) with varying number of layers of ωGCN.

Table 16. Ablation study on ωGAT.

Data.	Variant	Layers 2	4	8	16	32	64
Cora	$\omega\text{GAT}_\text{G}$	83.3	83.3	83.4	83.6	83.7	83.9
	$\omega\text{GAT}_\text{PL}$	83.4	83.5	83.8	84.0	84.1	84.0
	ωGAT	83.4	83.7	84.0	84.3	84.4	84.8
Cite.	$\omega\text{GAT}_\text{G}$	71.5	71.8	71.9	72.2	72.4	72.9
	$\omega\text{GAT}_\text{PL}$	72.1	72.3	72.4	72.8	73.1	73.2
	ωGAT	72.5	73.1	73.3	73.5	73.9	74.0
Pub.	$\omega\text{GAT}_\text{G}$	80.0	80.2	80.3	80.5	80.6	80.9
	$\omega\text{GAT}_\text{PL}$	80.0	80.4	80.7	81.1	81.2	81.4
	ωGAT	80.3	81.0	81.2	81.3	81.5	81.8

this result reflects the accuracy from a single split, we also repeat this experiment with 100 random splits as in (Chamberlain et al., 2021) and compare with applicable methods that also conducted such statistical significance test. In Table 17, we report our obtained accuracy on Cora, Citeseer, and Pubmed. It is possible to see that in this experiment our ωGCN and ωGAT outperform or obtain similar results compared with the considered methods, which further highlights the performance advantage of our method.

K. The Learnt $\vec{\omega}$

One of the main advantages of our method in Section 2 is that our method is capable of learning both smoothing and sharpening propagation operators, which cannot be obtained in most current GNNs. In Fig. 8 we present the actual $\{\vec{\omega^{(l)}}\}_{l=1}^{L}$ as a matrix of size $L \times c$ that was learnt for two dataset of different types—with high and low homophily score (as described in (Pei et al., 2020)). Namely, the Cora dataset with a high homophily score of 0.81, and the Texas dataset with a low homophily score of 0.11 (i.e., a heterophilic dataset). We see that on a homophilic dataset like Cora, the network learnt to perform diffusion, albeit in a controlled manner, and not to simply employ the standard averaging operator $\tilde{\mathbf{P}}$. We can further see that for a heterophilic dataset the ability to learn contrastive (i.e., sharpening) propagation operators in addition

Table 17. Semi-supervised node classification test accuracy 100 random train-val-test splits.

Method	Cora	Citeseer	Pubmed
GCN (Kipf & Welling, 2017)	81.5	71.9	77.8
GAT (Veličković et al., 2018)	81.8	71.4	78.7
MoNet (Monti et al., 2017)	81.3	71.2	78.6
GRAND-l (Chamberlain et al., 2021)	83.6	73.4	78.8
GRAND-nl (Chamberlain et al., 2021)	82.3	70.9	77.5
GRAND-nl-rw (Chamberlain et al., 2021)	83.3	74.1	78.1
GraphCON-GCN (Rusch et al., 2022a)	81.9	72.9	78.8
GraphCON-GAT (Rusch et al., 2022a)	83.2	73.2	79.5
GraphCON-Tran (Rusch et al., 2022a)	84.2	**74.2**	79.4
ωGCN (ours)	**84.5**	73.8	**82.9**
ωGAT (ours)	84.3	73.6	82.6

Table 18. Node classification accuracy (%) on additional datasets.

Method	Ogbn-arxiv
GCN (Kipf & Welling, 2017)	71.74
GAT (Veličković et al., 2018)	71.59
GATv2 (Brody et al., 2022)	71.87
APPNP (Klicpera et al., 2019)	71.82
Geom-GCN-P (Pei et al., 2020)	–
JKNet (Xu et al., 2018)	72.19
SGC (Wu et al., 2019)	69.20
GCNII (Chen et al., 2020b)	72.74
EGNN (Zhou et al., 2021)	72.70
GRAND (Chamberlain et al., 2021)	72.23
AGDN (Zhao et al., 2021)	**73.41**
ωGCN (Ours)	73.02
ωGAT (Ours)	72.76

to diffusive kernels is beneficial, and is also reflected in our results in Table 3-4, where a larger improvement is achieved in datasets like Cornell, Texas and Wisconsin, which have low homophily scores (Rusch et al., 2022a).

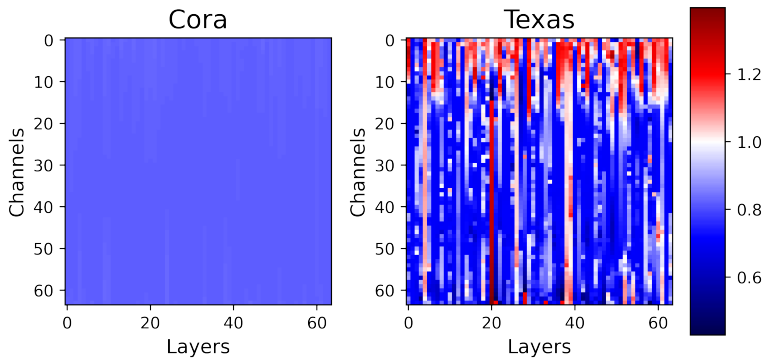

Figure 8. The learnt $\vec{\omega} \in \mathbb{R}^{64\times 64}$ of ωGCN with 64 layers (x-axis) and 64 channels (y-axis) for Cora (homophilic) and Texas (heterophilic) datasets. Smoothing operators appear in blue, while sharpening operators appear in red. White entries are obtained for $\omega = 1$.

L. Additional Backbones

In the main paper we considered GCN and GAT as our backbone to demonstrate ωGNNs. However, other architectures may also be augmented by learning the appropriate ω parameters. For instance, it may be possible to augment GRAND (Chamberlain et al., 2021), or the attention based (He et al., 2021b), as well as (Corso et al., 2020) that proposed to use multiple fixed propagation operators. We now provide a discussion regarding the augmentation of GRAND to obtain ωGRAND, followed by several experimental results.

ωGRAND Let us consider the GRAND method, as described in Equation 2 in (Chamberlain et al., 2021):

$$\frac{\partial f(t)}{\partial t} = (\hat{\mathbf{S}} - \mathbf{I}) f(t) \quad , t \in (0, T], \tag{25}$$

equipped with some initial value $f(t=0) = f_0$. Here, $\hat{\mathbf{S}}$ is the learnable attention propagation operator based on GAT (Veličković et al., 2018), which as discussed in Section 2.3 is non-negative and therefore smooths the node features $f(t)$. By discretizing Equation (25) with the forward Euler method, one obtains:

$$\frac{\mathbf{f}^{(l+1)} - \mathbf{f}^{(l)}}{h} = (\hat{\mathbf{S}} - \mathbf{I})\mathbf{f}^{(l)} \quad , l \in 0, \ldots, L-1. \tag{26}$$

By rearranging Equation (26), one obtains:

$$\mathbf{f}^{(l+1)} = (\mathbf{I} - h(\mathbf{I} - \hat{\mathbf{S}}))\mathbf{f}^{(l)} \quad , l = 0, \ldots, L-1. \tag{27}$$

Importantly, we note that Equation (27) here is the same as Equation (2) in the main paper without the 1×1 convolution, and the non-linearity activation function σ. The time h step is in fact equivalent to our learnable weight ω. The major difference is that we let ω to be learned rather than a fixed hyperparameter as in GRAND. In particular, GRAND employs the integration scheme in Equation (27) here, to compute (see Section 4 of GRAND (Chamberlain et al., 2021) for more information):

$$f(T) = f(0) + \int_0^T \frac{\partial f(t)}{\partial t} dt. \tag{28}$$

However, in GRAND the time step h (which is equal to our ω), is fixed, and the time T is replaced with L layers. Because in our ωGNNs (and specifically ωGRAND) we let ω change, we essentially learn the time integration length T. Furthermore, as shown in GRAND++ (Thorpe et al., 2022) (in the supplementary material, Table 5), increasing T, which is equivalent to adding more layers, causes accuracy degradation that is associated with the over-smoothing phenomenon. In light of our discussion above, and the findings in (Thorpe et al., 2022), we propose to consider an ωGRAND variant. Below, in Table 19 we show the results obtained with ωGRAND, on Cora and Citeseer using the 10 splits from Geom-GCN with a varying number of layers, from 8 to 64. Our results show that adding more layers to ωGRAND with a parameter per layer does not lead to performance degradation, unlike simply increasing T in GRAND. Note, that in our case, the time integration length is equal to the sum of all ω (in all layers and channels), i.e., $T = \sum \omega$. Therefore, we also report the effective time integration length in Table 19.

Table 19. The obtained accuracy (%) / total integration length (T) with GRAND and ωGRAND with a varying number of layers.

Dataset	Method	Layers			
		8	16	32	64
Cora	ωGRAND	88.12 / 7.88	88.20 / 9.19	88.18 / 9.06	88.11 / 9.24
	GRAND	87.83 / 8	87.31 / 16	86.20 / 32	73.06 / 64
Citeseer	ωGRAND	77.21 / 6.11	77.31 / 6.33	77.28 / 7.08	77.19 / 6.88
	GRAND	76.89 / 8	76.05 / 16	73.14 / 32	69.90 / 64