Brief Report

Not peer-reviewed version

On Addressing the Limitations of Graph Neural Networks

Sitao Luan [*]

Posted Date: 4 July 2023

doi: 10.20944/preprints202307.0118.v1

Keywords: Graph Neural Networks; Reinforcement Learning; Over-smoothing; Heterophily

Article

On Addressing the Limitations of Graph Neural Networks

Sitao Luan [1,2]

[1] McGill University, ; sitao.luan@mail.mcgill.ca
[2] Mila

Abstract: This report gives a comprehensive summary of two problems about graph convolutional networks (GCNs): over-smoothing and heterophily challenges, and outlines future directions to explore.

1. Introduction

Many real-world problems can be modeled as graphs. Recently, neural network based approaches have achieved significant progress for solving large, complex, graph-structured problems [9,12,17,21,24,34,50]. Inspired by the success of Convolutional Neural Networks (CNNs) [31] in computer vision [33], graph convolution defined on the graph Fourier domain stands out as the key operator and one of the most powerful tools for using machine learning to solve graph problems. Although with high expressive power, GCNs still suffer from several difficulties, *e.g.* the over-smoothing problem limits deep GCNs to sufficiently exploit multi-scale information, heterophily problem makes the graph-aware models underperform the graph-agnostic models. This report summarizes the methods we have proposed to address those challenges and puts forward some research problems we will investigate.

To fully explain the above problems, in Section 1.1, we will first introduce the notation and background knowledge of graph networks. In Section 2, we introduce the loss of expressive power of deep graph neural networks (GNNs) and propose snowball and truncated Krylov architecture to address it; in Section 3, we analyze heterophily problems for the existing GNNs and propose Adaptive Channel Mixing (ACM) architecture to address it.

Main Contribution

In Section 2, we first point out that the output of deep GCN with ReLU activation function will suffer from loss of rank problem under certain conditions and this can cause deep GCN lose expressive power. We then prove that Tanh is better at preserving the rank of the output and verify this claim with numerical tests. Then we find a way to deepen GCN in block Krylov form and propose snowball and truncated Krylov networks which perform better than state-of-the-arts (SOTA) model on semi-supervised node classification tasks on 3 benchmark datasets. Besides, we point out that finding **a specifically tailored weight initialization scheme for GCNs** can be an promising direction to address over-smoothing efficiently in Section 2.2. In Section 3, we first illustrate the insufficiency of the current homophily metrics and propose aggregation homophily based on a new similarity matrix. We then show the advantage of the new homophily metric over the existing ones on synthetic graph. Based on the similarity matrix, we define diversification distinguishability of a node and demonstrate why high-pass filters can help to address heterophily problem. To include both low-pass and high-pass in GNNs, we extend filterbank method and propose ACM and ACMII frameworks that can boost the performance of baseline GNNs on heterophilous graphs.

1.1. Notation and Background Knowledge

Suppose we have an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, A)$, where $\mathcal{V}$ is the node set with $|\mathcal{V}| = N$; $\mathcal{E}$ is the edge set without self-loop; $A \in \mathbb{R}^{N \times N}$ is the symmetric adjacency matrix with $A_{ij} = 1$ if and only if $e_{ij} \in \mathcal{E}$, otherwise $A_{ij} = 0$; D is the diagonal degree matrix, *i.e.* $D_{ii} = \sum_j A_{ij}$ and $\mathcal{N}_i = \{j : e_{ij} \in \mathcal{E}\}$ is the neighborhood set of node i. A graph signal is a vector $\boldsymbol{x} \in \mathbb{R}^N$ defined on $\mathcal{V}$, where x_i is defined on the node i. We also have a feature matrix $X \in \mathbb{R}^{N \times F}$ whose columns are graph signals and each node i has a corresponding feature vector $X_{i:}$ with dimension F, which is the i-th row of X. We denote $Z \in \mathbb{R}^{N \times C}$ as label encoding matrix, where $Z_{i:}$ is the one hot encoding of the label of node i and C is the total number of classes.

The (combinatorial) graph Laplacian is defined as $L = D - A$, which is a Symmetric Positive Semi-Definite (SPSD) matrix [7]. Its eigendecomposition gives $L = U\Lambda U^T$, where the columns of $U \in \mathbb{R}^{N\times N}$ are orthonormal eigenvectors, namely the *graph Fourier basis*, $\Lambda = \text{diag}(\lambda_1, \ldots, \lambda_N)$ with $\lambda_1 \leq \cdots \leq \lambda_N$, and these eigenvalues are also called *frequencies*. The graph Fourier transform of the graph signal $\boldsymbol{x}$ is defined as $\boldsymbol{x}_{\mathcal{F}} = U^{-1}\boldsymbol{x} = U^T\boldsymbol{x} = [\boldsymbol{u}_1^T\boldsymbol{x}, \ldots, \boldsymbol{u}_N^T\boldsymbol{x}]^T$, where $\boldsymbol{u}_i^T\boldsymbol{x}$ is the component of $\boldsymbol{x}$ in the direction of $\boldsymbol{u}_i$.

Some graph Laplacian variants are commonly used, *e.g.* the symmetric normalized Laplacian $L_{\text{sym}} = D^{-1/2}LD^{-1/2} = I - D^{-1/2}AD^{-1/2}$ and the random walk normalized Laplacian $L_{\text{rw}} = D^{-1}L = I - D^{-1}A$. The eigenvalues of L_{rw} and L_{sym} are the same and are in $[0, 2)$, and their corresponding eigenvectors satisfy $\boldsymbol{u}_{\text{rw}}^i = D^{-1/2}\boldsymbol{u}_{\text{sym}}^i$.

The affinity (transition) matrices can be derived from the Laplacians, *e.g.* $A_{\text{rw}} = I - L_{\text{rw}} = D^{-1}A$, $A_{\text{sym}} = I - L_{\text{sym}} = D^{-1/2}AD^{-1/2}$. Then $\lambda_i(A_{\text{rw}}) = \lambda_i(A_{\text{sym}}) = 1 - \lambda_i(A_{\text{sym}}) = 1 - \lambda_i(A_{\text{rw}}) \in (-1, 1]$. Renormalized affinity and Laplacian matrices are introduced in [24] as $\hat{A}_{\text{sym}} = \tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}$, $\hat{L}_{\text{sym}} = I - \hat{A}_{\text{sym}}$, where $\tilde{A} \equiv A + I, \tilde{D} \equiv D + I$ and it essentially adds a self-loop and is widely used in Graph Convolutional Network (GCN) as follows:

$$Y = \text{softmax}(\hat{A}_{\text{sym}}\ \text{ReLU}(\hat{A}_{\text{sym}}XW_0)\ W_1) \tag{1}$$

where $W_0 \in \mathbb{R}^{F\times F_1}$ and $W_1 \in \mathbb{R}^{F_1\times O}$ are parameter matrices. GCN can learn by minimizing the following cross entropy loss

$$\mathcal{L} = -\text{trace}(Z^T \log Y). \tag{2}$$

The random walk renormalized matrix $\hat{A}_{\text{rw}} = \tilde{D}^{-1}\tilde{A}$ can also be applied to GCN and it has the same eigenvalues as $\hat{A}_{\text{sym}}$. The corresponding Laplacian is defined as $\hat{L}_{\text{rw}} = I - \hat{A}_{\text{rw}}$ Specifically, the nature of random walk matrix makes $\hat{A}_{\text{rw}}$ behaves as a mean aggregator $(\hat{A}_{\text{rw}}\boldsymbol{x})_i = \sum_{j\in\{\mathcal{N}_i \cup i\}} x_j/(D_{ii} + 1)$ which is applied in [17] and is important to bridge the gap between spatial- and spectral-based graph convolution methods.

2. Loss of Expressive Power of Deep Graph Neural Networks

One major problem of the existing GCNs is the low expressive power limited by their shallow learning mechanisms [61,66]. There are mainly two reasons why an architecture that is scalable in depth has not been achieved yet. First, this problem is difficult: considering graph convolution as a special form of Laplacian smoothing [32], networks with multiple convolutional layers will suffer from an over-smoothing problem that makes the representation of even distant nodes indistinguishable [66]. Second, some people think it is unnecessary: for example, [4] states that it is not necessary for the label information to totally traverse the entire graph and one can operate on the multi-scale coarsened input graph and obtain the same flow of information as GCNs with more layers. Acknowledging the difficulty, we hold on to the objective of deepening GCNs since the desired compositionality[1] will yield easy articulation and consistent performance for problems with different scales.

In Section 2.1, we first analyze the limits of deep GCNs brought by over-smoothing and the activation functions. Then, we show that any graph convolution with a well-defined analytic spectral filter can be written as a product of a block Krylov matrix and a learnable parameter matrix in a special form. Based on this, we propose two GCN architectures that leverage multi-scale information in different ways and are scalable in depth, with stronger expressive powers and abilities to extract richer representations of graph-structured data. For empirical validation, we test different instances of the proposed architectures on multiple node classification tasks. The results show that even the simplest instance of the architectures achieves state-of-the-art performance, and the complex ones achieve surprisingly higher performance. In Section 2.2, we propose to study an over-smoothing problem and give some ideas.

[1] The expressive power of a sound deep Neural Network (NN) architecture should be expected to grow with the increment of network depth [19,30].

2.1. A Stronger Multi-Scale Deep GNN with Truncated Krylov Architecture

Suppose we deepen GCN in the same way as [24,32], we have

$$Y = \text{softmax}(\hat{A}_{\text{sym}}\ \text{ReLU}(\cdots \hat{A}_{\text{sym}}\ \text{ReLU}(\hat{A}_{\text{sym}}\ \text{ReLU}(\hat{A}_{\text{sym}} X W_0)\ W_1)\ W_2 \cdots)\ W_n) \equiv \text{softmax}(Y') \quad (3)$$

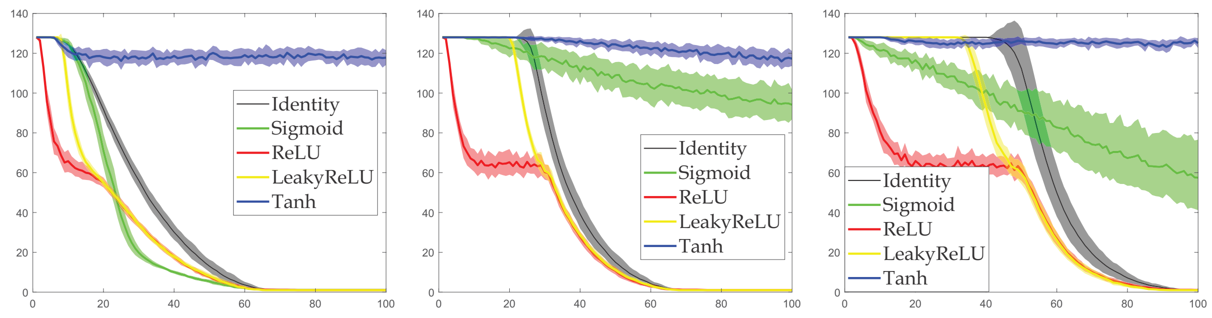
For this architecture, without considering the ReLU activation function, [32] shows that Y' will converge to a space spanned by the eigenvectors of $\hat{A}_{\text{sym}}$ with eigenvalue 1. Taking activation function into consideration, our analyses on (3) can be summarized in the following theorems (see proof in the appendix of [44]).

Theorem 1. Suppose that $\mathcal{G}$ has k connected components. Let $X \in \mathbb{R}^{N\times F}$ be any feature matrix and let W_j be any non-negative parameter matrix with $\|W_j\|_2 \leq 1$ for $j = 0, 1, \ldots$. If $\mathcal{G}$ has no bipartite components, then in (3), as $n \to \infty$, $\text{rank}(Y') \leq k$.

Theorem 2. Suppose the n-dimensional $\boldsymbol{x}$ and $\boldsymbol{y}$ are independently sampled from a continuous distribution and the activation function $\text{Tanh}(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$ is applied to $[\boldsymbol{x}, \boldsymbol{y}]$ pointwisely, then

$$\mathbb{P}(\text{rank}\left(\text{Tanh}([\boldsymbol{x}, \boldsymbol{y}])\right) = \text{rank}([\boldsymbol{x}, \boldsymbol{y}])) = 1$$

Theorem 1 shows that, even considering ReLU, if we simply deepen GCN as (3), the extracted features will degrade under certain conditions, *i.e.* Y' only contains the stationary information of the graph structure and loses all the local information in node for being smoothed. In addition, from the proof we see that the pointwise ReLU transformation is a conspirator. Theorem 2 tells us that Tanh is better at keeping linear independence among column features. We design a numerical experiment on synthetic data to test, under a 100-layer GCN architecture, how activation functions affect the rank of the output in each hidden layer during the feedforward process. As Figure 1a shows, the rank of hidden features decreases rapidly with ReLU, while having little fluctuation under Tanh, and even the identity function performs better than ReLU. So we propose to replace ReLU by Tanh.



(a) Deep GCN **(b)** Snowball **(c)** Truncated Block Krylov

Figure 1. Changes in the number of independent features with the increment of network depth.

Besides activation function, to find a way to deepen GCN, we first show that any graph convolution with well-defined analytic spectral filter defined on $\hat{A}_{\text{sym}} \in \mathbb{R}^{N\times N}$ can be written as a product of a block Krylov matrix with a learnable parameter matrix in a specific form. Based on this, we propose snowball network and truncated Krylov network.

We take $\mathbb{S} = \mathbb{R}^{F\times F}$. Given a set of block vectors $\{X_k\}_{k=1}^m \subset \mathbb{R}^{N\times F}$, the $\mathbb{S}$-span of $\{X_k\}_{k=1}^m$ is defined as $\text{span}^{\mathbb{S}}\{X_1, \ldots, X_m\} := \{\sum\limits_{k=1}^{m} X_k C_k : C_k \in \mathbb{S}\}$. Then, the order-$m$ block Krylov subspace with respect to the matrix $A \in \mathbb{R}^{N\times N}$, the block vector $B \in \mathbb{R}^{N\times F}$ and the vector space $\mathbb{S}$, and its corresponding block Krylov matrix are respectively defined as

$$\mathcal{K}_m^{\mathbb{S}}(A, B) \equiv \text{span}^{\mathbb{S}}\{B, AB, \ldots, A^{m-1}B\},\ \ K_m(A, B) \equiv [B, AB, \ldots, A^{m-1}B] \in \mathbb{R}^{N\times mF}.$$

It is shown in [11,15] that there exists a smallest m such that for any $k \geq m$, $A^k B \in \mathcal{K}_m^{\mathbb{S}}(A, B)$, where m depends on A and B.

Let $\rho(\hat{A}_{\text{sym}})$ denote the spectrum radius of $\hat{A}_{\text{sym}}$ and suppose $\rho(\hat{A}_{\text{sym}}) < R$ where R is the radius of convergence for a real analytic scalar function g. Based on the above definitions and conclusions, the graph convolution can be written as

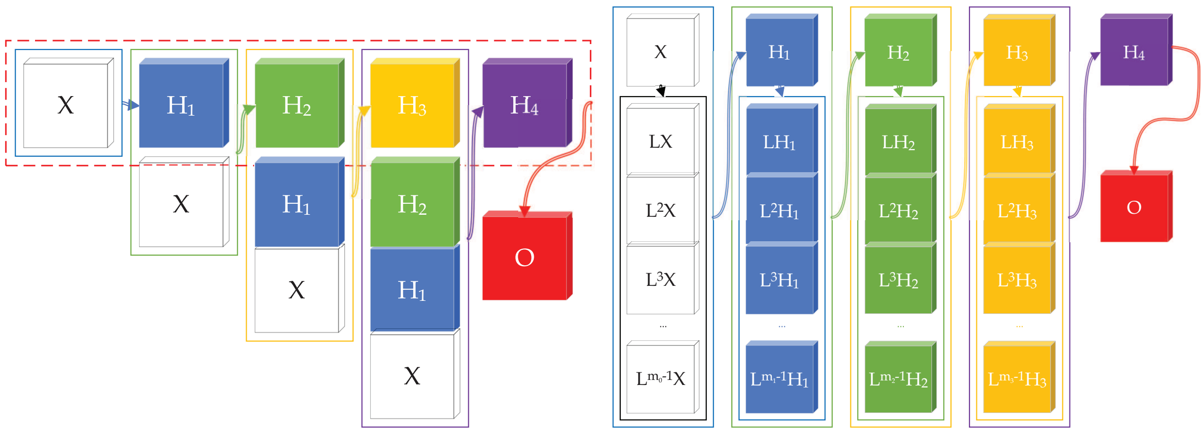
$$g(\hat{A}_{\text{sym}})X = \sum_{n=0}^{\infty} \frac{g^{(n)}(0)}{n!} \hat{A}_{\text{sym}}^{n} X \equiv \left[X, \hat{A}_{\text{sym}}X, \dots, \hat{A}_{\text{sym}}^{m-1}X\right]\left[(\Gamma_0{}^{\mathbb{S}})^T, (\Gamma_1{}^{\mathbb{S}})^T, \cdots, (\Gamma_{m-1}^{\mathbb{S}})^T\right]^T \equiv K_m(\hat{A}_{\text{sym}}, X)\Gamma^{\mathbb{S}} \tag{4}$$

where $\Gamma_i^{\mathbb{S}} \in \mathbb{R}^{F\times F}$ for $i = 1, \dots, m-1$ are parameter matrix blocks and $\Gamma^{\mathbb{S}} \in \mathbb{R}^{mF\times F}$. Then, a graph convolutional layer can generally be written as

$$g(\hat{A}_{\text{sym}})XW' = K_m(\hat{A}_{\text{sym}}, X)\Gamma^{\mathbb{S}}W' = K_m(\hat{A}_{\text{sym}}, X)W^{\mathbb{S}} \tag{5}$$

where $W' \in \mathbb{R}^{F\times O}$ is a parameter matrix, and $W^{\mathbb{S}} \equiv \Gamma^{\mathbb{S}}W' \in \mathbb{R}^{mF\times O}$. The essential number of learnable parameters is $mF \times O$.

The block Krylov form provides an insight about why an architecture that concatenates multi-scale features in each layer will boost the expressive power of GCN. Based on this idea, we propose the snowball and truncated Block Krylov architectures [44] shown in Figure 2, where we stack multi-scale information in each layer. From the performance comparison on semi-supervised node classification tasks with different label percentage in Table 1, we can see that the proposed models consistently perform better than the state-of-the-art models, especially when there are less labeled nodes. See detailed experimental results in [44].



(a) Snowball **(b)** Truncated Block Krylov

Figure 2. Snowball and Truncated Krylov Architectures.

2.2. *Future Works on Over-Smoothing*

Weight Initialization for GNNs

Even without aggregation in each hidden layer, an NN with deep architecture still suffers from vanishing activation variances and back-propagated gradients variance problem [13], which make the training of deep NN hard. In last decade, designing new parameter initialization methods is proved to be effective [13,18] to address the variance reduction problem during feedforward and backpropagation process. This motivates us to investigate the variance propagation in GNNs and analyze if the current weight initialization methods are suitable for GNNs or not. To this end, we can show that the vanishing variance caused by aggregation operation in GNNs is more serious than NN. Designing a new parameter initialization scheme for GNNs is potentially a feasible way to address this problem and empirically achieves promising performance [45]. we will propose a new method in this subsection.

Table 1. Accuracy without Validation. For each (column), the greener the cell, the better the performance. The redder, the worse. If our methods achieve better performance than all others, the corresponding cell will be in bold.

Algorithms	Cora						CiteSeer						PubMed			
	0.5%	1%	2%	3%	4%	5%	0.5%	1%	2%	3%	4%	5%	0.03%	0.05%	0.1%	0.3%
LP	56.4	62.3	65.4	67.5	69.0	70.2	34.8	40.2	43.6	45.3	46.4	47.3	61.4	66.4	65.4	66.8
Cheby	38.0	52.0	62.4	70.8	74.1	77.6	31.7	42.8	59.9	66.2	68.3	69.3	40.4	47.3	51.2	72.8
Co-training	56.6	66.4	73.5	75.9	78.9	80.8	47.3	55.7	62.1	62.5	64.5	65.5	62.2	68.3	72.7	78.2
Self-training	53.7	66.1	73.8	77.2	79.4	80.0	43.3	58.1	68.2	69.8	70.4	71.0	51.9	58.7	66.8	77.0
Union	58.5	69.9	75.9	78.5	80.4	81.7	46.3	59.1	66.7	66.7	67.6	68.2	58.4	64.0	70.7	79.2
Intersection	49.7	65.0	72.9	77.1	79.4	80.2	42.9	59.1	68.6	70.1	70.8	71.2	52.0	59.3	69.7	77.6
MultiStage	61.1	63.7	74.4	76.1	77.2		53.0	57.8	63.8	68.0	69.0		57.4	64.3	70.2	
M3S	61.5	67.2	75.6	77.8	78.0		56.1	62.1	66.4	70.3	70.5		59.2	64.4	70.6	
GCN	42.6	56.9	67.8	74.9	77.6	79.3	33.4	46.5	62.6	66.9	68.7	69.6	46.4	49.7	56.3	76.6
GCN-SVAT	43.6	53.9	71.4	75.6	78.3	78.5	47.0	52.4	65.8	68.6	69.5	70.7	52.1	56.9	63.5	77.2
GCN-DVAT	49	61.8	71.9	75.9	78.4	78.6	51.5	58.5	67.4	69.2	70.8	71.3	53.3	58.6	66.3	77.3
linear Snowball	***67.6***	***74.6***	***78.9***	***80.9***	***82.3***	***82.9***	***56.0***	***63.4***	***69.3***	***70.6***	***72.5***	***72.6***	***65.5***	***68.5***	***73.6***	***79.7***
Snowball	***68.4***	***73.2***	***78.4***	***80.8***	***82.3***	***83.0***	***56.4***	***63.9***	***68.7***	***70.5***	***71.8***	***72.8***	***66.5***	***68.6***	***73.2***	***80.1***
truncated Krylov	***71.8***	***76.5***	***80.0***	***82.0***	***83.0***	***84.1***	***59.9***	***66.1***	***69.8***	***71.3***	***72.3***	***73.7***	***68.7***	***71.4***	***75.5***	***80.4***

The current initialization scheme of GNNs still follows the Xavier initialization [13], *i.e.* $W_i \sim U\left[-\frac{\sqrt{6}}{\sqrt{n_j+n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j+n_{j+1}}}\right]$, or He (or Kaiming) initialization [18], *i.e.* $W_i \sim N\left(0, \sqrt{2/n_i}\right)$, which is designed for traditional multilayer perceptron (MLP) , where W_i is the parameter matrix of layer i and n_i is the number of hidden units of layer i. These two initialization methods are derived by studying the variance propagation between layers during feedforward and backpropagation process. These two processes are different in GNNs by an extra multiplication of aggregation operator $\hat{A}$. To analyze the variance propagation, we use deep GCN as an example, use $\hat{A} = \hat{A}_{\text{rw}}$ and decompose it as follows,

$$\begin{aligned} &Y_0 = X,\ H_1 = \hat{A}_{\text{rw}} X W_0,\ Y_1 = f(H_1),\ H_{l+1} = \hat{A}_{\text{rw}} Y_l W_l,\ Y_{l+1} = f(H_{l+1}),\ l = 1,\dots,n \\ &Y = \text{softmax}(\hat{A}_{\text{rw}} Y_n W_n) \equiv \text{softmax}(H_{n+1}),\ \mathcal{L} = -\text{trace}(Z^T \log Y) \end{aligned} \tag{6}$$

where $H_l, Y_l \in \mathbb{R}^{N\times F_l}$, $W_l \in \mathbb{R}^{F_l \times F_{l+1}}$; $Z \in \mathbb{R}^{N\times C}$ is the ground truth matrix with one-hot label vector. Then the gradient propagates in the following way,

$$\frac{\partial \mathcal{L}}{\partial H_l} = \frac{\partial \mathcal{L}}{\partial Y_l} \odot f'(H_l),\ \frac{\partial \mathcal{L}}{\partial W_{l-1}} = Y_{l-1}^T \hat{A}_{\text{rw}} \frac{\partial \mathcal{L}}{\partial H_l},\ \frac{\partial \mathcal{L}}{\partial Y_{l-1}} = \hat{A}_{\text{rw}} \frac{\partial \mathcal{L}}{\partial H_l} W_{l-1}^T \tag{7}$$

Variance Analysis: Forward View

Consider element i, j in matrix H_{l+1} during the feed-forward process in (6),

$$(H_{l+1})_{ij} = (\hat{A}_{\text{rw}})_{i,:} Y_l (W_l)_{:,j} = \sum_{t=1}^{F_l} \sum_{k=1}^{N} (\hat{A}_{\text{rw}})_{ik} (Y_l)_{kt} (W_l)_{t,j},\ Y_{l+1} = f(H_{l+1}),\ l = 1,\dots,n \tag{8}$$

Suppose we have linear activation function such as that proposed in [60]; each element in W_l is *i.i.d.*initialized with $E\left((W_l)_{ij}\right) = 0$; $E\left((Y_l)_{kt}\right) = 0$ and all elements in Y_l are independent [2]. Then, $\text{Var}\left((Y_{l+1})_{ij}\right) = \text{Var}\left((Y_{l+1})\right)$ can be written as

[2] For simplicity, the independence assumption is directly borrowed from [13], but theoretically it is too strong for GNNs. We will try to relax this assumption in the future.

$$\mathrm{Var}\left(\sum_{t=1}^{F_l}\sum_{k=1}^{N}(\hat{A}_{\mathrm{rw}})_{ik}\,(Y_l)_{kt}\,(W_l)_{t,j}\right)=\sum_{t=1}^{F_l}\sum_{k=1}^{N}\mathrm{Var}\left((\hat{A}_{\mathrm{rw}})_{ik}\,(Y_l)_{kt}\,(W_l)_{t,j}\right)=\frac{F_l}{d_i+1}\mathrm{Var}\,(Y_l)\mathrm{Var}(W_l) \tag{9}$$

Suppose each element in Y_l shares the same variance denoted as $\mathrm{Var}\,(Y_l)$. To prevent variance vanishing between layers, *i.e.* $\mathrm{Var}\,(Y_{l+1})=\mathrm{Var}\,(Y_l)$, from (8) we can approximately have (see computation in Appendix A.2.1)

$$\mathrm{Var}(W_l)=\frac{d_i+1}{F_l} \tag{10}$$

This tells us that the variance of W_l depends on the degree of a node, but since the parameter matrix is shared by all nodes, we cannot design a node specified initialization scheme. Thus, we make a compromise between nodes as follows

$$\mathrm{Var}(W_l)\approx\frac{\sum\limits_{i=1}^{N}(d_i+1)}{NF_l}=\frac{1+\text{average node degree}}{F_l} \tag{11}$$

Another way is to use weighted average by considering the node degree as the weight of each node. Through this way, we have

$$\mathrm{Var}(W)=\sum_{i=1}^{N}\frac{d_i+1}{\sum\limits_{j=1}^{N}d_j+1}\,\frac{d_i+1}{F_l}=\frac{\sum\limits_{i=1}^{N}(d_i+1)^2}{(\sum\limits_{i=1}^{N}d_i+1)F_l} \tag{12}$$

Variance Analysis: Backward View

Under the same assumption as feedforward view and suppose each element in $\frac{\partial\mathcal{L}}{\partial H_l}$ and $\frac{\partial\mathcal{L}}{\partial W_{l-1}}$ are independent to each other and has zero mean, from (7) we can approximately (see computation in Appendix A.2.2)

$$\frac{\partial\mathcal{L}}{\partial H_l}=\frac{\partial\mathcal{L}}{\partial Y_l}=\hat{A}_{\mathrm{rw}}\frac{\partial\mathcal{L}}{\partial H_{l+1}}W_l^T,\;\frac{\partial\mathcal{L}}{\partial W_{l-1}}=Y_{l-1}^T\hat{A}_{\mathrm{rw}}\frac{\partial\mathcal{L}}{\partial H_l} \tag{13}$$

Then,

$$\begin{aligned}\left(\frac{\partial\mathcal{L}}{\partial H_l}\right)_{ij}&=\sum_{t=1}^{F_{l+1}}\sum_{k=1}^{N}(\hat{A}_{\mathrm{rw}})_{ik}\left(\frac{\partial\mathcal{L}}{\partial H_{l+1}}\right)_{kt}(W_l^T)_{t,j}\\ \left(\frac{\partial\mathcal{L}}{\partial W_{l-1}}\right)_{ij}&=(\hat{A}_{\mathrm{rw}}Y_{l-1})_{\cdot i}^T\left(\frac{\partial\mathcal{L}}{\partial H_l}\right)_{\cdot j}=\sum_{k=1}^{N}(\sum_{t=1}^{N}(\hat{A}_{\mathrm{rw}})_{kt}(Y_{l-1})_{ti})\left(\frac{\partial\mathcal{L}}{\partial H_l}\right)_{kj},\end{aligned} \tag{14}$$

Thus,

$$\begin{aligned}\mathrm{Var}\left(\left(\frac{\partial\mathcal{L}}{\partial H_l}\right)_{ij}\right)&=\mathrm{Var}\left(\sum_{t=1}^{F_{l+1}}\sum_{k=1}^{N}\hat{A}_{\mathrm{rw}_{ik}}\left(\frac{\partial\mathcal{L}}{\partial H_{l+1}}\right)_{kt}(W_l^T)_{t,j}\right)=\frac{F_{l+1}}{d_i+1}\mathrm{Var}\left(\frac{\partial\mathcal{L}}{\partial H_{l+1}}\right)\mathrm{Var}\,(W_l)\\ \mathrm{Var}\left(\left(\frac{\partial\mathcal{L}}{\partial W_{l-1}}\right)_{ij}\right)&=\mathrm{Var}\left(\sum_{k=1}^{N}(\sum_{t=1}^{N}(\hat{A}_{\mathrm{rw}})_{kt}(Y_{l-1})_{ti})\left(\frac{\partial\mathcal{L}}{\partial H_l}\right)_{kj}\right)=\left(\sum_{k=1}^{N}\frac{1}{d_k+1}\right)\mathrm{Var}\,(Y_{l-1})\,\mathrm{Var}\left(\frac{\partial\mathcal{L}}{\partial H_l}\right)\end{aligned} \tag{15}$$

$$\mathrm{Var}(W_l)=\frac{\sum\limits_{i=1}^{N}(d_i+1)}{NF_{l+1}}\approx\frac{1+\text{average node degree}}{F_{l+1}} \tag{16}$$

From (9) and (15), Var (Y_{l-1}) can be approximately written as

$$\begin{aligned}
&\mathrm{Var}\left(\frac{\partial \mathcal{L}}{\partial H_l}\right) \approx \frac{NF_{l+1}}{\sum\limits_{i=1}^{N} d_i+1}\mathrm{Var}\left(\frac{\partial \mathcal{L}}{\partial H_{l+1}}\right)\mathrm{Var}\,(W_l) \approx \mathrm{Var}\left(\frac{\partial \mathcal{L}}{\partial H_{n+1}}\right)\prod_{l'=l+1}^{n+1}\frac{NF_{l'}}{\sum\limits_{i=1}^{N} d_i+1}\mathrm{Var}\,(W_{l'-1}) \\
&\mathrm{Var}\,(Y_{l-1}) \approx \frac{NF_{l-2}}{\sum_k d_k+1}\mathrm{Var}\,(\mathbf{Y_{l-2}})\mathrm{Var}(W_{l-2}) \approx \mathrm{Var}(\mathbf{Y_0})\prod_{l'=0}^{l-2}\frac{NF_{l'}}{\sum_k d_k+1}\mathrm{Var}(W_{l'})
\end{aligned} \tag{17}$$

From (15), if each $\mathrm{Var}(W_{l'})$ equals to $\mathrm{Var}(W)$ and each F_l equals to F, then

$$\mathrm{Var}\left(\left(\frac{\partial \mathcal{L}}{\partial W_{l-1}}\right)_{ij}\right) \approx \left(\sum_{k=1}^{N}\frac{1}{d_k+1}\right)\mathrm{Var}(\mathbf{Y_0})\mathrm{Var}\left(\frac{\partial \mathcal{L}}{\partial H_{n+1}}\right)\left(\frac{NF}{\sum_k d_k+1}\mathrm{Var}(W)\right)^n \tag{18}$$

Combined with (11), we can set the variance of the parameter matrix as

$$\mathrm{Var}(W_l) \approx \frac{2\sum\limits_{i=1}^{N}(d_i+1)}{N(F_l+F_{l+1})} = \frac{2(1+\text{average node degree})}{(F_l+F_{l+1})} \tag{19}$$

Thus, each element in W_i can be drawn from $N(0, \sqrt{\frac{2(1+\text{average node degree})}{(F_l+F_{l+1})}})$.

Adaptive ReLU (AdaReLU) Activation Function

To satisfy the assumption that the activation function is linear at the beginning of training process and to still learn a nonlinear function during training, we design the following adaptive ReLU (AdaReLU) activation function

$$f(\boldsymbol{x}_i) = \begin{cases} \beta_i \boldsymbol{x}_i, & \text{if } \boldsymbol{x}_i > 0 \\ \alpha_i \boldsymbol{x}_i, & \text{if } \boldsymbol{x}_i \leq 0 \end{cases}$$

where α_i and β_i are learnable parameters and are initialized to be 1. If $\alpha_i = \alpha$ and $\beta_i = \beta$ for all i, we have channel-shared AdaReLU, otherwise we have channel-wise AdaReLU [3]. From the preliminary experimental results, channel-wise works better than channel-shared AdaReLU.

There exist some experimental evidence [45] that controlling variance flow by initialization like (19) can relieve the performance decrease of deep GCN. But more tests and hyperparameter tunning still needs to be done. More theoretical analysis on variance propagation needs to be done.

3. GNNs on Heterophily Graphs

GNNs are considered as an extension of basic Neural Networks (NNs) by additionally making use of graph structure based on the relational inductive bias (homophily assumption), rather than treating the nodes as collections of independent and identically distributed (*i.i.d.*) samples. Though GNNs are believed to outperform basic NNs in real-world tasks, it is found that in some cases, the graph-aware models have little performance gain or even underperform graph-agnostic models [6,46,51,69,71]. One of the main reasons for the performance degradation is believed to be heterophily, *i.e.* when the connected nodes tend to have different labels [69,71]. Heterophily challenge has received attention recently and there are increasing number of models being put forward to analyze [39,43] and address this problem [6,36,41,42,64,69,70].

In this section , we first introduce the most commently used homophily metrics in Section 3.1. Then, we show that not all cases of heterophily are harmful for GNNs and propose new metrics based on a similarity matrix which considers the influence of both graph structure and input features on GNNs in

[3] The words "channel-shared" and "channel-wise" are borrowed from [18], which indicate if we share the same learning parameter between each feature dimension or not.

Section 3.2. The metrics demonstrate advantages over the commonly used homophily metrics by tests on synthetic graphs. From the metrics and the observations, we find some cases of harmful heterophily can be addressed by diversification operation and its effectiveness can be proved in Section 3.3. With this fact and knowledge of filterbanks, we propose the Adaptive Channel Mixing (ACM) framework in Section 3.4 to adaptively exploit aggregation, diversification and identity channels in each GNN layer to address harmful heterophily. We validate the ACM-augmented baselines with real-world node classification tasks. They consistently achieve significant performance gain and exceed the state-of-the-art GNNs on most of the tasks without incurring significant computational burden. In Section 3.5, we introduce some prior work on addressing heterophily problems and explain their differences with ACM framework. The limitation of diversification operation and remaining challenges of heterophily problems are discussed in Section 3.6

3.1. Metrics of Homophily

The metrics of homophily are defined by considering different relations between node labels and graph structures defined by adjacency matrix. There are three commonly used homophily metrics: edge homophily [1,70], node homophily [51], and class homophily [35] [4] defined as follows:

$$\begin{gathered}
H_{\text{edge}}(\mathcal{G}) = \frac{\left|\{e_{uv} \mid e_{uv} \in \mathcal{E}, Z_{u,:} = Z_{v,:}\}\right|}{|\mathcal{E}|}, \quad H_{\text{node}}(\mathcal{G}) = \frac{1}{|\mathcal{V}|} \sum_{v \in \mathcal{V}} \frac{\left|\{u \mid u \in \mathcal{N}_v, Z_{u,:} = Z_{v,:}\}\right|}{d_v}, \\
H_{\text{class}}(\mathcal{G}) = \frac{1}{C-1} \sum_{k=1}^{C} \left[h_k - \frac{\left|\{v \mid Z_{v,k} = 1\}\right|}{N} \right]_+, \quad h_k = \frac{\sum_{v \in \mathcal{V}} \left|\{u \mid Z_{v,k} = 1, u \in \mathcal{N}_v, Z_{u,:} = Z_{v,:}\}\right|}{\sum_{v \in \{v \mid Z_{v,k}=1\}} d_v}
\end{gathered} \tag{20}$$

where $[a]_+ = \max(a, 0)$; h_k is the class-wise homophily metric [35]. They are all in the range of $[0, 1]$ and a value close to 1 corresponds to strong homophily while a value close to 0 indicates strong heterophily. $H_{\text{edge}}(\mathcal{G})$ measures the proportion of edges that connect two nodes in the same class; $H_{\text{node}}(\mathcal{G})$ evaluates the average proportion of edge-label consistency of all nodes; $H_{\text{class}}(\mathcal{G})$ tries to avoid the sensitivity to imbalanced class, which can cause H_{edge} misleadingly large. The above definitions are all based on the graph-label consistency and imply that the inconsistency will cause harmful effect to the performance of GNNs. With this in mind, we will show a counter example to illustrate the insufficiency of the above metrics and propose new metrics in the following subsection.

3.2. Analysis of Heterophily and Aggregation Homophily Metric

Heterophily is believed to be harmful for message-passing based GNNs [6,51,70] because intuitively features of nodes in different classes will be falsely mixed and this will lead nodes indistinguishable [70]. Nevertheless, it is not always the case, *e.g.* the bipartite graph shown in Figure 3 is highly heterophilous according to the homophily metrics in (20), but after mean aggregation, the nodes in classes 1 and 2 only exchange colors and are still distinguishable. Authors in [6] also point out the insufficiency of H_{node} by examples to show that different graph typologies with the same H_{node} can carry different label information.

[4] The authors in [35] did not name this homophily metric. We name it class homophily based on its definition.

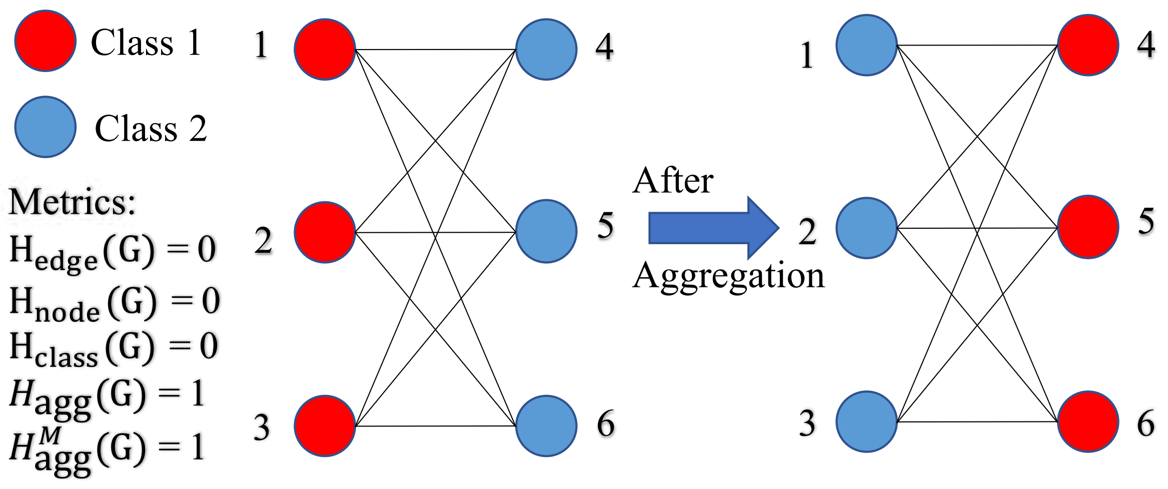


Figure 3. Example of harmless heterophily

To analyze to what extent the graph structure can affect the output of a GNN, we first simplify the GCN by removing its nonlinearity as [60]. Let $\hat{A} \in \mathbb{R}^{N\times N}$ denote a general aggregation operator. Then, Equation (1) can be simplified as,

$$Y = \text{softmax}(\hat{A}XW) = \text{softmax}(Y') \tag{21}$$

After each gradient decent step $\Delta W = \gamma \frac{d\mathcal{L}}{dW}$, where γ is the learning rate, the update of Y' will be,

$$\Delta Y' = \hat{A}X\Delta W = \gamma \hat{A}X\frac{d\mathcal{L}}{dW} \propto \hat{A}X\frac{d\mathcal{L}}{dW} = \hat{A}XX^T\hat{A}^T(Z-Y) = S(\hat{A},X)(Z-Y) \tag{22}$$

where $S(\hat{A},X) \equiv \hat{A}X(\hat{A}X)^T$ is a post-aggregation node similarity matrix, $Z-Y$ is the prediction error matrix. The update direction of node i is essentially a weighted sum of the prediction error, *i.e.* $\Delta(Y')_{i,:} = \sum_{j\in\mathcal{V}}\left[S(\hat{A},X)\right]_{i,j}(Z-Y)_{j,:}$.

To study the effect of heterophily, we first define the *aggregation similarity score* as follows.

Definition 1. *Aggregation similarity score*

$$S_{agg}\left(S(\hat{A},X)\right) = \frac{\left|\left\{v \,\middle|\, \text{Mean}_u\left(\{S(\hat{A},X)_{v,u}|Z_{u,:} = Z_{v,:}\}\right) \geq \text{Mean}_u\left(\{S(\hat{A},X)_{v,u}|Z_{u,:} \neq Z_{v,:}\}\right)\right\}\right|}{|\mathcal{V}|} \tag{23}$$

where Mean_u $(\{\cdot\})$ *takes the average over u of a given multiset of values or variables.*

$S_{\text{agg}}(S(\hat{A},X))$ measures the proportion of nodes $v \in \mathcal{V}$ that will put relatively larger similarity weights on nodes in the same class than in other classes after aggregation. It is easy to see that $S_{\text{agg}}(S(\hat{A},X)) \in [0,1]$. But in practice, we observe that in most datasets, we will have $S_{\text{agg}}(S(\hat{A},X)) \geq 0.5$. Based on this observation, we rescale (23) to the following modified aggregation similarity for practical usage,

$$S^M_{\text{agg}}\left(S(\hat{A},X)\right) = \left[2S_{\text{agg}}\left(S(\hat{A},X)\right) - 1\right]_+ \tag{24}$$

In order to measure the consistency between labels and graph structures without considering node features and make a fair comparison with the existing homophily metrics in (20), we define the graph ($\mathcal{G}$) aggregation ($\hat{A}$) homophily and its modified version as

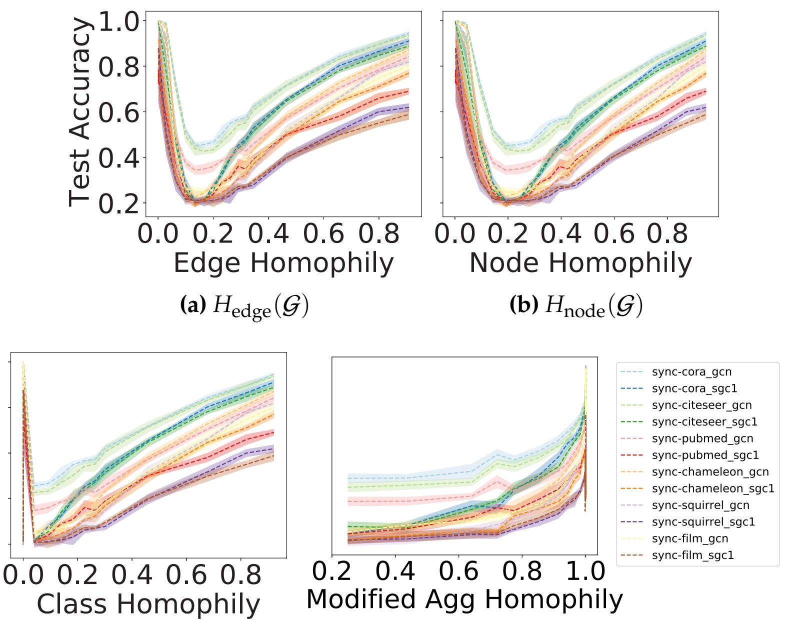
$$H_{\text{agg}}(\mathcal{G}) = S_{\text{agg}}\left(S(\hat{A},Z)\right),\ H^M_{\text{agg}}(\mathcal{G}) = S^M_{\text{agg}}\left(S(\hat{A},Z)\right) \tag{25}$$

In practice, we will only check $H_{\text{agg}}(\mathcal{G})$ when $H^M_{\text{agg}}(\mathcal{G}) = 0$. As Figure 3 shows, when $\hat{A} = \hat{A}_{\text{rw}}$, $H_{\text{agg}}(\mathcal{G}) = H^M_{\text{agg}}(\mathcal{G}) = 1$. Thus, this new metric reflects the fact that nodes in classes 1 and 2 are still highly distinguishable after aggregation, while other metrics mentioned before fail to capture the information and misleadingly give value 0. This shows the advantage of $H_{\text{agg}}(\mathcal{G})$ and $H^M_{\text{agg}}(\mathcal{G})$ by additionally considering information from aggregation operator $\hat{A}$ and the similarity matrix.

Comparison of Homophily Metrics on Synthetic Graphs

To comprehensively compare $H^M_{\text{agg}}(\mathcal{G})$ with the metrics in (20) in terms of how they reveal the influence of graph structure on the GNN performance, we generate synthetic graphs (d-regular graphs with edge homophily varied from 0.005 to 0.95) and evaluate SGC with 1-hop aggregation (SGC-1) [60] and GCN [24] on them.

The performance of SGC-1 and GCN are expected to be monotonically increasing with a proper and informative homophily metric. However, Figure 4a–c show that the performance curves under $H_{\text{edge}}(\mathcal{G}), H_{\text{node}}(\mathcal{G})$ and $H_{\text{class}}(\mathcal{G})$ are U-shaped [5], while Figure 4d reveals a nearly monotonic curve only with a little numerical perturbation around 1. This indicates that $H^M_{\text{agg}}(\mathcal{G})$ can describe how the graph structure affects the performance of SGC-1 and GCN more appropriately and adequately than the existing metrics.



(a) $H_{\text{edge}}(\mathcal{G})$ **(b)** $H_{\text{node}}(\mathcal{G})$

(c) $H_{\text{class}}(\mathcal{G})$ **(d)** $H^M_{\text{agg}}(\mathcal{G})$

Figure 4. Comparison of baseline performance under different homophily metrics.

3.3. How Diversification Operation Helps with Harmful Heterophily

We first consider the example shown in Figure 5. From $S(\hat{A}, X)$, nodes 1,3 assign relatively large positive weights to nodes in class 2 after aggregation, which will make node 1,3 hard to be distinguished from nodes in class 2. Despite the fact, we can still distinguish between nodes 1,3 and 4,5,6,7 by considering their neighborhood difference: nodes 1,3 are different from most of their neighbors while nodes 4,5,6,7 are similar to most of their neighbors. This indicates, in some cases, although some nodes become similar after aggregation, they are still distinguishable via their surrounding dissimilarities. This leads us to use *diversification operation*, *i.e.* high-pass (HP) filter $I - \hat{A}$ [10] (will be introduced in the next

[5] A similar J-shaped curve is found in [70], though using different data generation processes. It does not mention the insufficiency of edge homophily.

subsection) to extract the information of neighborhood differences and address harmful heterophily. As $S(I-\hat{A},X)$ in Figure 5 shows, nodes 1,3 will assign negative weights to nodes 4,5,6,7 after diversification operation, *i.e.* nodes 1,3 treat nodes 4,5,6,7 as negative samples and will move away from them during backpropagation. Base on this example, we first propose diversification distinguishability as follows to measures the proportion of nodes that diversification operation is potentially helpful for,

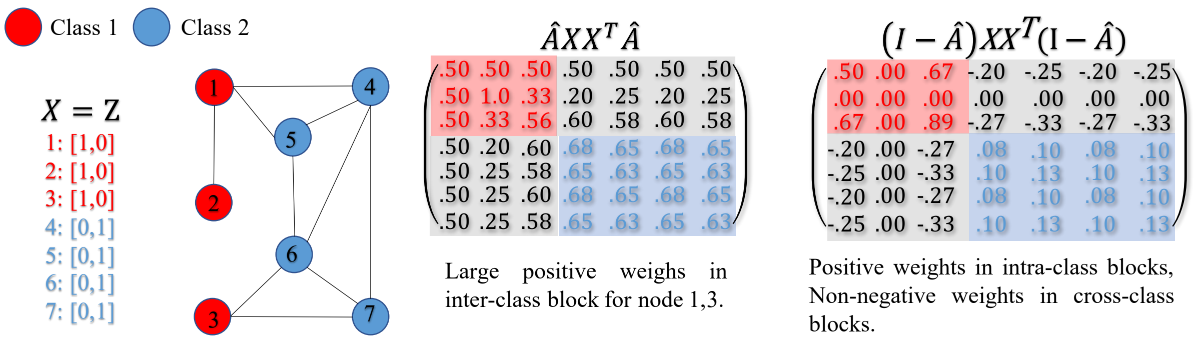


Figure 5. Example of how HP filter addresses harmful heterophily.

Definition 2. *Diversification Distinguishability (DD) based on $S(I-\hat{A},X)$.*

Given $S(I-\hat{A},X)$, a node v is diversification distinguishable if the following two conditions are satisfied at the same time,

$$\begin{aligned} &\textbf{1. } \mathrm{Mean}_u\left(\{S(I-\hat{A},X)_{v,u} | u\in\mathcal{V}\wedge Z_{u,:}=Z_{v,:}\}\right)>0;\\ &\textbf{2. } \mathrm{Mean}_u\left(\{S(I-\hat{A},X)_{v,u} | u\in\mathcal{V}\wedge Z_{u,:}\neq Z_{v,:}\}\right)\leq 0 \end{aligned} \tag{26}$$

Then, graph diversification distinguishability value is defined as

$$\mathrm{DD}_{\hat{A},X}(\mathcal{G})=\frac{1}{|\mathcal{V}|}\Big|\{v|v \textit{ is diversification distinguishable}\}\Big| \tag{27}$$

We can see that $\mathrm{DD}_{\hat{A},X}(\mathcal{G})\in[0,1]$. The effectiveness of diversification operation can be proved for binary classification problems under certain conditions based on definition 2, leading us to:

Theorem 3. Suppose $X=Z,\hat{A}=\hat{A}_{\mathrm{rw}}$. Then, for a binary classification problem, *i.e.* $C=2$, all nodes are diversification distinguishable, *i.e.* $\mathrm{DD}_{\hat{A},Z}(\mathcal{G})=1$.

Theorem 3 theoretically demonstrates the importance of diversification operation to extract high-frequency information of graph signal [10]. Combined with aggregation operation, which is a low-pass filter [10,48], we can get a filterbank which uses both aggregation and diversification operations to distinctively extract the low- and high-frequency information from graph signals. We will introduce filterbank in the next subsection.

3.4. Filterbank and Adaptive Channel Mixing(ACM) GNN Framework

Filterbank

For the graph signal $\boldsymbol{x}$ defined on $\mathcal{G}$, a 2-channel linear (analysis) filterbank [10] [6] includes a pair of low-pass(LP) and high-pass(HP) filters $H_{\mathrm{LP}}, H_{\mathrm{HP}}$, where H_{LP} and H_{HP} retain the low-frequency and high-frequency content of $\boldsymbol{x}$, respectively. Filterbanks with $H_{\mathrm{LP}}+H_{\mathrm{HP}}=I$ will not lose any information of the input signal, *i.e.* perfect reconstruction property [10].

However, most existing GNNs are under uni-channel filtering architecture [17,24,58] with either H_{LP} or H_{HP} channel that only partially preserves the input information. Generally, the Laplacian matrices

[6] In graph signal processing, an additional synthesis filter [10] is required to form the 2-channel filterbank. But synthesis filter is not needed in our framework, so we do not introduce it in our paper.

$(L_{\text{sym}}, L_{\text{rw}}, \hat{L}_{\text{sym}}, \hat{L}_{\text{rw}})$ can be regarded as HP filters [10] and affinity matrices $(A_{\text{sym}}, A_{\text{rw}}, \hat{A}_{\text{sym}}, \hat{A}_{\text{rw}})$ can be treated as LP filters [16,48]. Moreover, we consider MLPs as owing a special identity filterbank with matrix I that satisfies $H_{\text{LP}} + H_{\text{HP}} = I + 0 = I$.

Filterbank in Spatial Form

Filterbank methods can also be extended to spatial GNNs. Formally, on the node level, left multiplying H_{LP} and H_{HP} on $\boldsymbol{x}$ performs as aggregation and diversification operations, respectively. For example, suppose $H_{\text{LP}} = \hat{A}$ and $H_{\text{HP}} = I - \hat{A}$, then for node i we have

$$(H_{\text{LP}}\boldsymbol{x})_i = \sum_{j\in\{\mathcal{N}_i \cup i\}} \hat{A}_{i,j}\boldsymbol{x_j},\ (H_{\text{HP}}\boldsymbol{x})_i = \boldsymbol{x_i} - \sum_{j\in\{\mathcal{N}_i \cup i\}} \hat{A}_{i,j}\boldsymbol{x_j} \tag{28}$$

where $\hat{A}_{i,j}$ is the connection weight between two nodes. To leverage HP and identity channels in GNNs, we propose the Adaptive Channel Mixing (ACM) framework which can be applied to lots of baseline GNN. We use GCN as an example and introduce ACM framework in matrix form. We use H_{LP} and H_{HP} to represent general LP and HP filters. The ACM framework includes 3 steps as follows,

$$\begin{aligned}
&\textbf{Step 1. Feature Extraction for Each Channel:}\\
&\text{Option 1: } H_L^l = \text{ReLU}\left(H_{\text{LP}}H^{l-1}W_L^{l-1}\right),\ H_H^l = \text{ReLU}\left(H_{\text{HP}}H^{l-1}W_H^{l-1}\right), H_I^l = \text{ ReLU}\left(IH^{l-1}W_I^{l-1}\right);\\
&\text{Option 2: } H_L^l = H_{\text{LP}}\text{ReLU}\left(H^{l-1}W_L^{l-1}\right),\ H_H^l = H_{\text{HP}}\text{ReLU}\left(H^{l-1}W_H^{l-1}\right), H_I^l = I\ \text{ReLU}\left(H^{l-1}W_I^{l-1}\right);\\
&W_L^{l-1},\ W_H^{l-1},\ W_I^{l-1} \in \mathbb{R}^{F_{l-1}\times F_l};\\
&\textbf{Step 2. Feature-based Weight Learning}\\
&\tilde{\alpha}_L^l = \sigma\left(H_L^l\tilde{W}_L^l\right),\ \tilde{\alpha}_H^l = \sigma\left(H_H^l\tilde{W}_H^l\right),\ \tilde{\alpha}_I^l = \sigma\left(H_I^l\tilde{W}_I^l\right),\ \tilde{W}_L^{l-1},\ \tilde{W}_H^{l-1},\ \tilde{W}_I^{l-1} \in \mathbb{R}^{F_l\times 1}\\
&\left[\alpha_L^l, \alpha_H^l, \alpha_I^l\right] = \text{Softmax}\left(\left[\tilde{\alpha}_L^l, \tilde{\alpha}_H^l, \tilde{\alpha}_I^l\right]W_{\text{Mix}}^l/T,\right),\ W_{\text{Mix}}^l \in \mathbb{R}^{3\times 3}, T \in \mathbb{R} \text{ is the temperature;}\\
&\textbf{Step 3. Node-wise Channel Mixing:}\\
&H^l = \left(\text{diag}(\alpha_L^l)H_L^l + \text{diag}(\alpha_H^l)H_H^l + \text{diag}(\alpha_I^l)H_I^l\right).
\end{aligned} \tag{29}$$

The framework with option 1 in step 1 is ACM framework and with option 2 is ACMII framework. ACM(II)-GCN first implement distinct feature extractions for 3 channels, respectively. After processed by a set of filterbanks, 3 filtered components H_L^l, H_H^l, H_I^l are obtained. Different nodes may have different needs for the information in the 3 channels, *e.g.* in Figure 5, nodes 1,3 demand high-frequency information while node 2 only needs low-frequency information. To adaptively exploit information from different channels, ACM(II)-GCN learns row-wise (node-wise) feature-conditioned weights to combine the 3 channels. ACM(II) can be easily plugged into spatial GNNs by replacing H_{LP} and H_{HP} by aggregation and diversification operations as shown in (28).

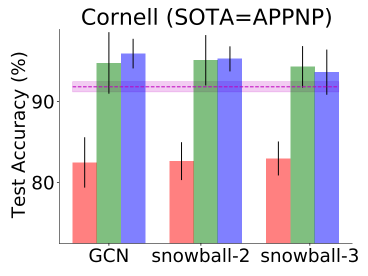
Complexity

Number of learnable parameters in layer l of ACM(II)-GCN is $3F_{l-1}(F_l + 1) + 9$, while it is $F_{l-1}F_l$ in GCN. The computation of step 1-3 takes $NF_l(8 + 6F_{l-1}) + 2F_l(\text{nnz}(H_{\text{LP}}) + \text{nnz}(H_{\text{HP}})) + 18N$ flops, while GCN layer takes $2NF_{l-1}F_l + 2F_l(\text{nnz}(H_{\text{LP}}))$ flops, where $\text{nnz}(\cdot)$ is the number of non-zero elements.

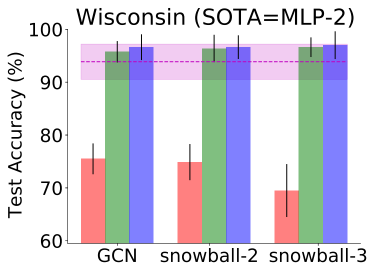
Performance Comparison

We implement SGC [60] with 1 hop and 2 hops (SGC-1, SGC-2), GCNII [5], GCNII* [5], GCN [24] and snowball networks with 2 and 3 layers (snowball-2, snowball-3) and apply them in ACM or ACMII framework: we use $\hat{A}_{\text{rw}}$ as LP filter and the corresponding HP filter is $I - \hat{A}_{\text{rw}}$. We compare them with several baseline and SOTA GNN models: MLP with 2 layers (MLP-2), GAT [58], APPNP [25], GPRGNN [6], H$_2$GCN [70], MixHop [1], GCN+JK [24,35,63], GAT+JK [35,58,63], FAGCN [2] GraphSAGE [17] and Geom-GCN [51]. Besides the 9 benchmark datasets *Cornell*, *Wisconsin*, *Texas*, *Film*, *Chameleon*, *Squirrel*, *Cora*, *Citeseer* and *Pubmed* used in [51], we further test the above models on a new benchmark dataset, *Deezer-Europe*, that is proposed in [35]. On each dataset used in [51], we test the models 10 times following the same early stopping strategy, the same random data splitting method and Adam [23] optimizer as used in GPRGNN [6]. For *Deezer-Europe*, we test the above models 5 times with the same early stopping strategy, the same fixed splits and AdamW [37] used in [35].

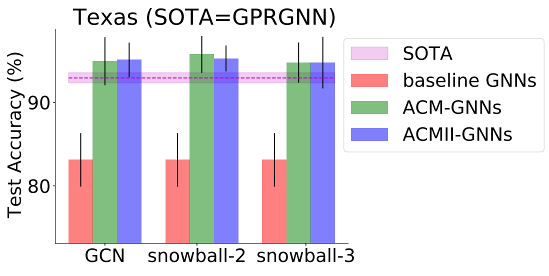
To better visualize the performance boost and the comparison with SOTA models, in Figure 6, we plot the bar charts of the test accuracy of SOTA models, 3 selected baselines (GCN, snowball-2, snowball-3) and their ACM and ACMII augmented models on 6 most commonly used benchmark heterophily datasets (See [40] for the full results and comparison). We can see that after being applied in ACM or ACMII framework, the performance of the 3 baseline models are significantly boosted on all tasks and can achieve SOTA performance. Especially on *Cornell*, *Texas*, *Film* and *Squirrel*, the augmented models significantly outperform the current SOTA models. Overall, It suggests that ACM or ACMII framework can help GNNs to generalize better on node classification tasks on heterophilous graphs.



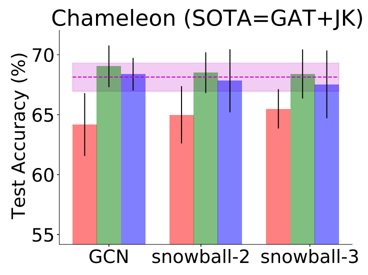
(a) $H_{\text{agg}}^{M}(\mathcal{G}) = 0.8032$, $\uparrow$ 4.1 %



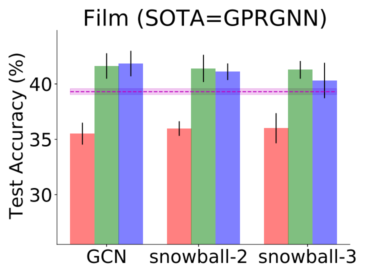
(b) $H_{\text{agg}}^{M}(\mathcal{G}) = 0.7768$, $\uparrow$3.13 %



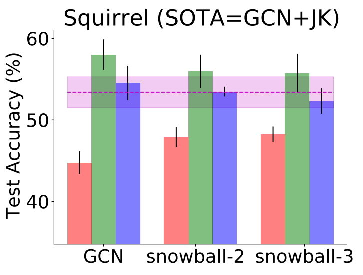
(c) $H_{\text{agg}}^{M}(\mathcal{G}) = 0.694$, $\uparrow$2.82 %



(d) $H_{\text{agg}}^{M}(\mathcal{G}) = 0.61$, $\uparrow$ 0.9 %



(e) $H_{\text{agg}}^{M}(\mathcal{G}) = 0.6822$, $\uparrow$2.54%



(f) $H_{\text{agg}}^{M}(\mathcal{G}) = 0.3566$, $\uparrow$4.62 %

Figure 6. Comparison of SOTA models (magenta), selected baseline GNNs (red) and their ACM (green) and ACMII (blue) augmented models on 6 selected datasets. The black line and the error bar indicate the standard deviation. The symbol "↑" means the amount of improvement of the best ACM-baseline and ACM-baseline over the SOTA models.

3.5. Prior Work

We discuss relevant work of GNNs on addressing heterophily challenge in this part. Authors in [1] acknowledge the difficulty of learning on graphs with weak homophily and propose MixHop to extract features from multi-hop neighborhood to get more information. Geom-GCN [51] precomputes unsupervised node embeddings and uses graph structure defined by geometric relationships in the embedding space to define the bi-level aggregation process. Authors in [20] propose measurements based on feature smoothness and label smoothness that are potentially helpful to guide GNNs on dealing with heterophilous graphs. H$_2$GCN [70] combines 3 key designs to address heterophily: (1) ego- and neighbor-embedding separation; (2) higher-order neighborhoods; (3) combination of intermediate representations. CPGNN [69] models label correlations by the compatibility matrix, which is beneficial for heterophily settings, and propagates a prior belief estimation into GNNs by the compatibility matrix. FBGNN [47] first proposes to use filterbank to address heterophily problem, but it does not fully explain the insights behind HP filters and does not contain identity channel and node-wise channel mixing mechanism. FAGCN [2] learns edge-level aggregation weights as GAT [58] but allows the weights to be negative which enables the network to capture the high-frequency components in graph signals.

GPRGNN [6] uses learnable weights that can be both positive and negative for feature propagation, it allows GRPGNN to adapt heterophily structure of graph and is able to handle both high- and low-frequency parts of the graph signals.

3.6. Future Work

Limitation of Diversification Operation

Diversification operation does not work well in all harmful heterophily cases. For example, consider an imbalanced dataset where several small clusters with distinctive labels are densely connected to a large cluster. In this case, the surrounding differences of nodes in small clusters are similar, *i.e.* the neighborhood differences are mainly from their connection to the same large cluster, and this possibly makes diversification operation fail to discriminate them. Thus, it is obvious that ACM framework is not able to handle all heterophily cases.

From Figure 4, we can see that GNNs consistently perform well in the high homophily area. This reveals the fact that all homophily cases are helpful. This reminds us that instead of using a fixed adjacency matrix, we can learn a new adjacency matrix with different homophily level. With this in mind, we design an architecture with additional adjacency learner as shown in Figure 7: instead of using a fixed predefined adjacency matrix, we will learn an adjacency matrix with edges that can reveal the label similarity between nodes, *i.e.* homophily. . This adjacency learner should ideally be trained end-to end. From some preliminary experimental results (not included in this report) of a GCN with a pretrained adjacency learner, this method is promising although there are some stability issues need to be fixed.

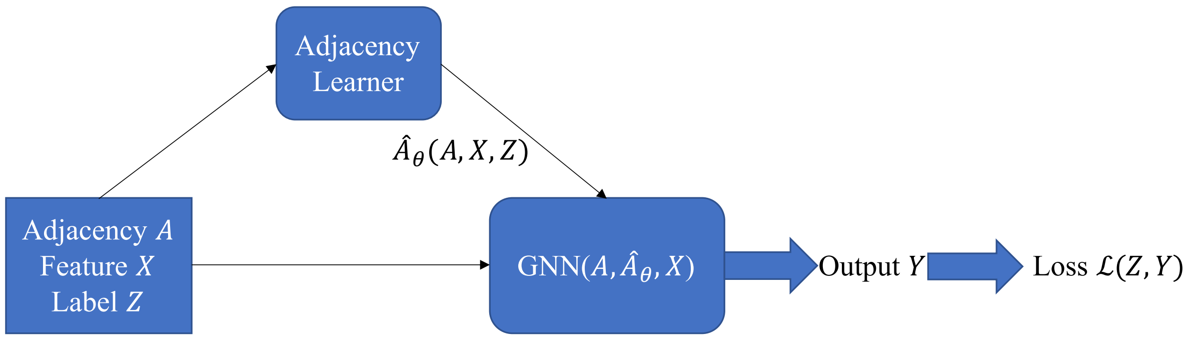


Figure 7. GNN with adjacency learner.

Exploring Different Ways for Adjacency Candidate Selection

Some tricks can be explored when we are selecting the adjacency candidates for the adjacency learner:

- Sample or select (top-k_1) nodes from complementary graph, put them together with the pre-defined neighborhood set to form adjacency candidate set, then sample or select (top-k_2) adjacency candidates for training. Try to train it end-to-end.
- Consider modeling the candidate selection process as a multi-armed bandit problem. Find an efficient way to learn to select good candidates from complementary graph. Can use pseudo count to prevent selecting the same nodes repeatedly.

4. Graph Representation Learning for Reinforcement Learning

4.1. Markov Decision Process (MDP)

MDP is a framework to model the learning process that the agent learns from the interaction with the environment[56,67,68]. The interaction happens in discrete time steps, $t = 0, 1, 2, 3, \cdots$. At step t, given a state $S_t = s_t \in \mathcal{S}$, the agent picks an action $a_t \in \mathcal{A}(s_t)$ according to a policy $\pi(\cdot|s_t)$, which is a rule of choosing actions given a state. Then, at time $t+1$, the environmental dynamics $p : \mathcal{S} \times \mathcal{R} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ take the agent to a new state $S_{t+1} = s_{t+1} \in \mathcal{S}$ and provide a numerical reward $R_{t+1} = r_{t+1}(s_t, a_t, s_{t+1}) \in \mathbb{R}$. Such a sequence of interaction gives us a trajectory $\tau = \{S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \cdots\}$. The objective is to find an optimal policy to maximize the expected long-term discounted cumulative reward $V_\pi(s) = E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s]$ for each state s, where γ is the discount factor.

For a given policy π, solving its value function $\boldsymbol{V}_\pi$ is equivalent to solving the following linear system,

$$\boldsymbol{V}_\pi = \boldsymbol{r}_\pi + \gamma P_\pi \boldsymbol{V}_\pi \tag{30}$$

where $\boldsymbol{V}_\pi = [V_\pi(s)]^T_{s\in\mathcal{S}} \in \mathbb{R}^{|\mathcal{S}|}, \boldsymbol{r}_\pi = [r_\pi(s)]^T_{s\in\mathcal{S}} \in \mathbb{R}^{|\mathcal{S}|}, P_\pi = [P_\pi(s'|s)]_{s',s\in\mathcal{S}} \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{S}|}$. The state transition matrix P_π essentially defines a graph structure over states and the reward vector $\boldsymbol{r}_\pi$ is a signal defines on graph. Thus, solving value function can be considered as a (supervised or semi-supervised) node regression tasks over graph. Besides solving $\boldsymbol{V}_\pi$, the graph structure can also be used for reward propagation and representation learning in Reinforcement Learning (RL) [26–28].

4.2. Graph Representation Learning for MDP

Treating MDP as a graph is an old but never outdated idea. Traditional methods use graph Laplacian for a fixed policy to estimate $\boldsymbol{V}_\pi$, *e.g.* proto-value function [49]. In addition to value function estimation, [28] proposes to use GCN to learn potential-based reward shaping, which can accelerate the learning process of the agent.

The above methods both construct the graph from the sampled trajectory data. With modern Graph Representation Learning (GRL) methods *e.g.* node embedding methods [3], link prediction methods [52,54,57], we can learn to reconstruct the underlying graph (adjacency matrix) from sampled data more efficiently. And label propagation [53], which is a commonly used algorithm for graph semi-supervised learning, can be helpful for efficient reward propagation. In Section 4.3, we will introduce the potential of using GRL for reward propagation and representation learning in reinforcement learning.

4.3. Reinforcement Learning with Graph Representation Learning

In this section, we will draw how to represent Markov Decision Process (MDP) with graph and introduce two possible ways of using graph representation learning to address the problems defined on MDP.

Each state can be treated as a node on a graph, the transition probability between each pair of nodes (an element in state transition matrix) can be represented by the edge (or weight) between them and value function is a function defined on each node of the graph. The details (for finite MDP) are introduced in matrix form as follows [38,59]:

- Denote $|S||A| \times |S|$ environment transition matrix as P, where

$$P_{(sa,s')} = \sum_r p\left(s', r|s, a\right) \tag{31}$$

  and $P_{(sa,s')} \geq 0$, $\sum_{s'} P_{(sa,s')} = 1$, for all s, a. Note that P is not a square matrix.
- We rewrite the policy π by an $|S| \times |S||A|$ matrix Π, where $\Pi_{(s,s'a)} = \pi(a|s)$ if $s' = s$, otherwise 0:

$$\Pi = \mathrm{diag}(\pi(\cdot|s_1)^T, \cdots, \pi(\cdot|s_{|S|})^T) \tag{32}$$

  where $\pi(\cdot|s_i)^T$ is an $|A|$-dimensional row vector. From this definition, one can quickly verify that the matrix product ΠP gives the $|S| \times |S|$ state-to-state transition matrix P_π (asymmetric) induced by the policy π in the environment P, and the $|S||A| \times |S||A|$ matrix product $P\Pi$ gives the state-action-to-state-action transition matrix P'_π (asymmetric) induced by policy π in the environment P.
- We denote the $|S||A| \times 1$ reward vector as $\boldsymbol{r}$, whose entry $r_{(sa)}$ specifies the reward obtained when taking action a in state s, i.e.

$$\boldsymbol{r}_{(sa)} = E[r|s, a] = \sum_{s'\in\mathcal{S}} P_{sa,s'} \cdot r(s, a, s'). \tag{33}$$

- The state value function and state-action value function can be represented by

$$\boldsymbol{V}_{\boldsymbol{\pi}} = \sum_{i=0}^{\infty} \gamma^i (\Pi P)^i \Pi \boldsymbol{r} = \Pi \boldsymbol{r} + \gamma \Pi P \boldsymbol{V}_{\boldsymbol{\pi}} \in \mathbb{R}^{|S|\times 1},\ \boldsymbol{Q}_{\boldsymbol{\pi}} = \sum_{i=0}^{\infty} \gamma^i (P\Pi)^i \boldsymbol{r} = \boldsymbol{r} + \gamma P \Pi \boldsymbol{Q}_{\boldsymbol{\pi}} \in \mathbb{R}^{|S||A|\times 1} \quad (34)$$

4.3.1. Learn Reward Propagation as Label Propagation

The sampling process from an MDP can be considered as a random walk defined on a graph, because the relation (edge) between each pair of states is essentially a transition probability [7]. Discovering the underlying graph of a MDP can help us to leverage the correlation between states to learn value function or do to efficient exploration in sparse reward environment.

Usually, the graph is constructed from the trajectory data, *i.e.* the pairwise state transition data. But once we update the policy, we need to reconstruct the graph. With graph embedding methods for link prediction tasks, *e.g.*, Deepwalk [52], node2vec [14], Line[57], we are able to learn graph reconstruction by inferring some unobserved transition. To be more specifically, instead of learning $P_\pi(s'|s)$ for a fixed policy π, we can learn the state-action transition probability $P(s'|s,a)$, which is independent of π. In this way, we can take use of trajectory data in all history no matter the policy changes or not. And once we are given a policy, we can infer the graph by combining $\pi(a|s)$ and $P(s'|s,a)$.

4.3.2. Graph Embedding as Auxiliary Task for Representation Learning

Learning auxiliary tasks is showed to be helpful for state representation learning [22], which is critical to learn a good policy for agents. Among the methods, successor representation is showed to be theoretically and empirically important for learning a good state representations [8,29]. Modeling the successor triplet (s,a,s') for MDP is essentially equivalent to modeling the triplet (head, relation, tail) in knowledge graph. And there exist a lot of algorithms in knowledge graph embedding community to address triplet embedding problem, *e.g.*, TransE [3], RotateE [55], QuatE [65] and DihEdral [62]. These methods can be borrowed to learn richer representation for RL tasks.

Appendix A. Calculation of Variances

Appendix A.1. Background

We first decompose the deep GCN architecture as follows

$$\begin{aligned}
&\boldsymbol{Y_0} = \boldsymbol{X},\ \boldsymbol{H_1} = \hat{A}\boldsymbol{X}W_0,\ \boldsymbol{Y_1} = f(\boldsymbol{H_1}) \\
&\boldsymbol{H_{l+1}} = \hat{A}\boldsymbol{Y_l}W_l,\ \boldsymbol{Y_{l+1}} = f(\boldsymbol{H_{l+1}}),\ l = 1,\dots,n \\
&\boldsymbol{Y} = \text{softmax}(\hat{A}\boldsymbol{Y_n}W_n) \equiv \text{softmax}(\boldsymbol{H_{n+1}}) \\
&L = -\text{trace}(\boldsymbol{Z}^T \log \boldsymbol{Y})
\end{aligned} \quad (A1)$$

[7] From this perspective, we should not treat the trajectory as sequential data, because we do not necessarily have an ordered relation between states on a graph, even for directed graph. Although the observation seems to have an order in it, we actually only have transition relation.

where $\boldsymbol{H_l}, \boldsymbol{Y_l} \in \mathbb{R}^{N \times F_l}$, $W_l \in \mathbb{R}^{F_l \times F_{l+1}}$; $\boldsymbol{Z} \in \mathbb{R}^{N \times C}$ is the ground truth matrix with one-hot label vector $\boldsymbol{Z}_{i,:}$ in each row, C is number of classes; L is the scalar loss. Then the gradient propagates in the following way

$$\begin{aligned}
\text{Output } \frac{\partial L}{\partial \boldsymbol{H_{n+1}}} &= \text{softmax}(\boldsymbol{H_{n+1}}) - \boldsymbol{Z} \\
\frac{\partial L}{\partial W_n} &= \boldsymbol{Y_n}^T \hat{A} \frac{\partial L}{\partial \boldsymbol{H_{n+1}}}, \ \frac{\partial L}{\partial \boldsymbol{Y_n}} = \hat{A} \frac{\partial L}{\partial \boldsymbol{H_{n+1}}} W_n^T \\
\text{Hidden } \frac{\partial L}{\partial \boldsymbol{H_l}} &= \frac{\partial L}{\partial \boldsymbol{Y_l}} \odot f'(\boldsymbol{H_l}), \ \frac{\partial L}{\partial W_{l-1}} = \boldsymbol{Y_{l-1}}^T \hat{A} \frac{\partial L}{\partial \boldsymbol{H_l}}, \\
\frac{\partial L}{\partial \boldsymbol{Y_{l-1}}} &= \hat{A} \frac{\partial L}{\partial \boldsymbol{H_l}} W_{l-1}^T
\end{aligned} \tag{A2}$$

where $\odot$ is the Hadamard product. The gradient propagation of GCN differs from that of multi-layer perceptron (MLP) by an extra multiplication of $\hat{A}$ when the gradient signal flows through $\boldsymbol{Y_l}$.

Appendix A.2. Variance Analysis

Appendix A.2.1. Forward View

$$\begin{aligned}
&\boldsymbol{H_{l+1}} = \hat{A}\boldsymbol{Y_l}W_l, \ (\boldsymbol{H_{l+1}})_{ij} = \hat{A}_{i,:}\boldsymbol{Y_l}(W_l)_{:,j} = \sum_{t=1}^{F_l}\sum_{k=1}^{N} \hat{A}_{ik}\,(\boldsymbol{Y_l})_{kt}\,(W_l)_{t,j} \\
&\boldsymbol{Y_{l+1}} = f(\boldsymbol{H_{l+1}}), \ l = 1, \ldots, n
\end{aligned} \tag{A3}$$

Suppose the activation function is identity function such as [60], all element in W share the same variance and each element in $\boldsymbol{X}$ are independent and share the same variance, $E\left((\boldsymbol{Y_l})_{kt}\right) = 0$ and $E\Big((W_l)_{ij}\Big) = 0$, $\text{Var}\Big((\boldsymbol{Y_{l+1}})_{ij}\Big)$ can be written as

$$\begin{aligned}
&\text{Var}\left(\sum_{t=1}^{F_l}\sum_{k=1}^{N}\hat{A}_{ik}\,(\boldsymbol{Y_l})_{kt}\,W_{t,j}\right) = \sum_{t=1}^{F_l}\sum_{k=1}^{N}\text{Var}\left(\hat{A}_{ik}\,(\boldsymbol{Y_l})_{kt}\,W_{t,j}\right) \\
&= F_l(d_i+1)\cdot\frac{1}{(d_i+1)^2}\text{Var}\left(\boldsymbol{Y_l}\right)\text{Var}\left(W\right) \\
&= \frac{F_l}{d_i+1}\text{Var}\left(\boldsymbol{Y_l}\right)\text{Var}(W) = \text{Var}\left(\boldsymbol{Y_{l+1}}\right)
\end{aligned} \tag{A4}$$

Then, if we want $\text{Var}\left(\boldsymbol{Y_{l+1}}\right) = \text{Var}\left(\boldsymbol{Y_l}\right)$ we will have

$$\text{Var}(W) = \frac{d_i+1}{F_l} \tag{A5}$$

Since the parameter matrix is shared by all nodes, we cannot design a node specified initialization scheme for the W. Thus, we initialize each element in W by the average values as follows

$$\text{Var}(W) = \frac{\sum\limits_{i=1}^{N}(d_i+1)}{NF_l} = \frac{1 + \text{average node degree}}{F_l} \tag{A6}$$

If we relax the assumption $E\left((\boldsymbol{Y_l})_{kt}\right) = 0$ and assume it nonzero, as shown in [18], if we use ReLU activation function, we will have

$$\text{Var}(W) = \frac{2\sum\limits_{i=1}^{N}(d_i+1)}{NF_l} = \frac{2(1 + \text{average node degree})}{F_l} \tag{A7}$$

If we consider the correlation between nodes and keep the assumption that each dimension of the feature (columns in $\boldsymbol{X}$) is independent and $E(X) \neq 0$, we will have

$$
\begin{aligned}
\operatorname{Var}\left((\boldsymbol{H}_{l+1})_{ij}\right) &= \operatorname{Var}\left(\sum_{t=1}^{F_l}\sum_{k=1}^{N} \hat{A}_{ik}\,(\boldsymbol{Y}_l)_{kt}\,W_{t,j}\right) \\
&= F_l \cdot \operatorname{Var}\left(\left(\sum_{k=1}^{N} A_{ik}\,(\boldsymbol{Y}_l)_k\right) W\right) = F_l \cdot E\left(\sum_{k=1}^{N} A_{ik}\,(\boldsymbol{Y}_l)_k\right)^2 \operatorname{Var}(W) \\
&= \frac{F_l}{(d_i+1)^2} E\left(\sum_{k\in\mathcal{N}_i} (\boldsymbol{Y}_l)_k\right)^2 \operatorname{Var}(W) \\
&= \frac{F_l}{(d_i+1)^2} E\left(\sum_{k\in\mathcal{N}_i} (\boldsymbol{Y}_l)_k^2 + \sum_{k,j\in\mathcal{N}_i,k\neq j} (\boldsymbol{Y}_l)_k\,(\boldsymbol{Y}_l)_j\right) \operatorname{Var}(W)
\end{aligned}
\tag{A8}
$$

Since

$$
E\left((\boldsymbol{Y}_l)_k\,(\boldsymbol{Y}_l)_j\right) = \operatorname{Cov}\left((\boldsymbol{Y}_l)_k, (\boldsymbol{Y}_l)_j\right) + E\left((\boldsymbol{Y}_l)_k\right) E\left((\boldsymbol{Y}_l)_j\right)
$$

we can have several reasonable assumptions over $\operatorname{Cov}\left((\boldsymbol{Y}_l)_k, (\boldsymbol{Y}_l)_j\right)$ to get different results

- The adjacency matrix with self-loop can be considered as a prior covariance matrix and thus a reasonable assumption is $\operatorname{Cov}\left((\boldsymbol{Y}_l)_k, (\boldsymbol{Y}_l)_j\right) = \operatorname{Var}\left((\boldsymbol{Y}_l)_k\right) = \operatorname{Var}\left((\boldsymbol{Y}_l)_j\right)$.
- Consider symmetric normalized $\hat{A}$ as a prior covariance matrix and we have $\operatorname{Cov}\left((\boldsymbol{Y}_l)_k, (\boldsymbol{Y}_l)_j\right) = \sqrt{\operatorname{Var}\left((\boldsymbol{Y}_l)_k\right)\operatorname{Var}\left((\boldsymbol{Y}_l)_j\right)} = \operatorname{Var}\left((\boldsymbol{Y}_l)_k\right)$

These assumptions all lead us to

$$
E\left((\boldsymbol{Y}_l)_k\,(\boldsymbol{Y}_l)_j\right) = \operatorname{Var}\left((\boldsymbol{Y}_l)_k\right) + E^2\left((\boldsymbol{Y}_l)_k\right) = E\,(\boldsymbol{Y}_l)_k^2
$$

Thus we have

$$
\begin{aligned}
\operatorname{Var}\left((\boldsymbol{H}_{l+1})_{ij}\right) &= F_l \cdot \frac{1}{(d_i+1)^2} E\left((d_i+1)^2\,(\boldsymbol{Y}_l)_k^2\right) \operatorname{Var}(W) \\
&= F_l \cdot E\left((\boldsymbol{Y}_l)_k^2\right) \operatorname{Var}(W) = F_l \cdot \frac{1}{2} \operatorname{Var}\left(\boldsymbol{H}_l\right) \operatorname{Var}(W)
\end{aligned}
\tag{A9}
$$

Thus

$$
\operatorname{Var}(W) = \frac{2}{F_l}
\tag{A10}
$$

Suppose $E((\boldsymbol{Y}_l)_k\,(\boldsymbol{Y}_l)_j) \geq 0$ and $Cov((\boldsymbol{Y}_l)_k, (\boldsymbol{Y}_l)_j) \geq 0$, since $Cov((\boldsymbol{Y}_l)_k, (\boldsymbol{Y}_l)_j) \leq \sqrt{\operatorname{Var}((\boldsymbol{Y}_l)_k)\operatorname{Var}((\boldsymbol{Y}_l)_j)} = \operatorname{Var}((\boldsymbol{Y}_l)_k)$, we have $E((\boldsymbol{Y}_l)_k\,(\boldsymbol{Y}_l)_j) \leq E\,(\boldsymbol{Y}_l)_k^2$ and we assume $E((\boldsymbol{Y}_l)_k\,(\boldsymbol{Y}_l)_j) = \alpha E\,(\boldsymbol{Y}_l)_k^2, \alpha \in [0,1]$. Then,

$$
\begin{aligned}
\operatorname{Var}\left((\boldsymbol{H}_{l+1})_{ij}\right) &= F_l \cdot \frac{1}{(d_i+1)^2}(d_i+1)(\alpha d_i+1) E\,(\boldsymbol{Y}_l)_k^2 \operatorname{Var}(W) \\
&= F_l \cdot \frac{\alpha d_i+1}{d_i+1} E\,(\boldsymbol{Y}_l)_k^2 \operatorname{Var}(W) \\
&= F_l \cdot \frac{\alpha d_i+1}{2(d_i+1)} \operatorname{Var}\left(\boldsymbol{H}_l\right) \operatorname{Var}(W)
\end{aligned}
\tag{A11}
$$

Thus,

$$
\operatorname{Var}(W) = \frac{2(d_i+1)}{F_l(\alpha d_i+1)}
\tag{A12}
$$

$(d_i+1)/(\alpha d_i+1)$ can be considered as the effective degree of node i and an estimation is

$$\begin{aligned}\hat{\mathrm{Var}}(W) &= \frac{\sum_i 2(d_i+1)/(\alpha d_i+1)}{NF_l} \\ &= \frac{2\times \text{average effective node degree}}{F_l}\end{aligned} \tag{A13}$$

Appendix A.2.2. Backward View

Under linear assumption, we have $\boldsymbol{H_l}=\boldsymbol{Y_l}$ and

$$\frac{\partial L}{\partial \boldsymbol{H_l}} = \frac{\partial L}{\partial \boldsymbol{Y_l}} = \hat{A}\frac{\partial L}{\partial \boldsymbol{H_{l+1}}}W_l^T, \ \frac{\partial L}{\partial W_{l-1}} = \boldsymbol{Y_{l-1}^T}\hat{A}\frac{\partial L}{\partial \boldsymbol{H_l}}, \tag{A14}$$

We have

$$\begin{aligned}\left(\frac{\partial L}{\partial \boldsymbol{H_l}}\right)_{ij} &= \sum_{t=1}^{F_{l+1}}\sum_{k=1}^{N}\hat{A}_{ik}\left(\frac{\partial L}{\partial \boldsymbol{H_{l+1}}}\right)_{kt}(W_l^T)_{t,j} \\ \left(\frac{\partial L}{\partial W_{l-1}}\right)_{ij} &= (\hat{A}\boldsymbol{Y_{l-1}})_{\cdot i}^T\left(\frac{\partial L}{\partial \boldsymbol{H_l}}\right)_{\cdot j} = \sum_{k=1}^{N}(\sum_{t=1}^{N}\hat{A}_{kt}(\boldsymbol{Y_{l-1}})_{ti})\left(\frac{\partial L}{\partial \boldsymbol{H_l}}\right)_{kj},\end{aligned} \tag{A15}$$

And

$$\begin{aligned}\mathrm{Var}\left(\left(\frac{\partial L}{\partial \boldsymbol{H_l}}\right)_{ij}\right) &= \mathrm{Var}\left(\sum_{t=1}^{F_{l+1}}\sum_{k=1}^{N}\hat{A}_{ik}\left(\frac{\partial L}{\partial \boldsymbol{H_{l+1}}}\right)_{kt}(W_l^T)_{t,j}\right) \\ &= \frac{F_{l+1}}{d_i+1}\mathrm{Var}\left(\left(\frac{\partial L}{\partial \boldsymbol{H_{l+1}}}\right)_{kt}(W_l^T)_{t,j}\right) \\ &= \frac{F_{l+1}}{d_i+1}\mathrm{Var}\left(\frac{\partial L}{\partial \boldsymbol{H_{l+1}}}\right)\mathrm{Var}\left(W_l\right) \\ &\approx \frac{NF_{l+1}}{\sum_i(d_i+1)}\mathrm{Var}\left(\frac{\partial L}{\partial \boldsymbol{H_{l+1}}}\right)\mathrm{Var}\left(W_l\right) \\ &\approx \mathrm{Var}\left(\frac{\partial L}{\partial \boldsymbol{H_{n+1}}}\right)\prod_{l'=l}^{n}\frac{NF_{l+1}}{\sum_i(d_i+1)}\mathrm{Var}\left(W_l\right) \\ \mathrm{Var}\left(\left(\frac{\partial L}{\partial W_{l-1}}\right)_{ij}\right) &= \mathrm{Var}\left(\sum_{k=1}^{N}(\sum_{t=1}^{N}\hat{A}_{kt}(\boldsymbol{Y_{l-1}})_{ti})\left(\frac{\partial L}{\partial \boldsymbol{H_l}}\right)_{kj}\right) \\ &= \sum_{k=1}^{N}\mathrm{Var}\left(\sum_{t=1}^{N}\hat{A}_{kt}(\boldsymbol{Y_{l-1}})_{ti}\right)\mathrm{Var}\left(\left(\frac{\partial L}{\partial \boldsymbol{H_l}}\right)_{kj}\right) \\ &= \left(\sum_{k=1}^{N}\frac{1}{d_k+1}\right)\mathrm{Var}\left(\boldsymbol{Y_{l-1}}\right)\mathrm{Var}\left(\frac{\partial L}{\partial \boldsymbol{H_l}}\right)\end{aligned} \tag{A16}$$

From (A4), Var $(\boldsymbol{Y_{l-1}})$ can be approximately written as

$$\begin{aligned}&\mathrm{Var}\left(\boldsymbol{Y_{l-1}}\right) = \mathrm{Var}\left(\boldsymbol{H_{l-1}}\right) \approx \frac{NF_{l-2}}{\sum_k d_k+1}\mathrm{Var}\left(\boldsymbol{Y_{l-2}}\right)\mathrm{Var}(W) \\ &\approx \mathrm{Var}(\boldsymbol{Y_0})\prod_{l'=0}^{l-2}\frac{NF_{l'}}{\sum_k d_k+1}\mathrm{Var}(W)\end{aligned} \tag{A17}$$

Then

$$
\begin{aligned}
&\mathrm{Var}\left(\left(\frac{\partial L}{\partial W_{l-1}}\right)_{ij}\right)\\
&\approx\left(\sum_{k=1}^{N}\frac{1}{d_k+1}\right)\mathrm{Var}(\boldsymbol{Y_0})\mathrm{Var}\left(\frac{\partial L}{\partial \boldsymbol{H_{n+1}}}\right)\prod_{l'=0,l'\neq l-1}^{n}\frac{NF_{l'}}{\sum_k d_k+1}\mathrm{Var}(W)
\end{aligned}
\tag{A18}
$$

Appendix A.3. Energy Analysis

Another way to design weight initialization is from the flow of energy perspective. Under the linear assumption and suppose we can do QR factorization of the weight matrices to make them orthogonal and $W^TW = WW^T = \alpha I$, then we have

$$
\begin{aligned}
&\mathrm{trace}(\boldsymbol{H}_{l+1}^T\boldsymbol{H}_{l+1}) = \mathrm{trace}\left((\hat{A}\boldsymbol{Y_l}W_l)^T\hat{A}\boldsymbol{Y_l}W_l\right)\\
&= \mathrm{trace}\left(W_l^T\boldsymbol{Y}_l^T\hat{A}^T\hat{A}\boldsymbol{Y_l}W_l\right) = \mathrm{trace}\left(W_lW_l^T\boldsymbol{Y}_l^T\hat{A}^T\hat{A}\boldsymbol{Y_l}\right)\\
&= \mathrm{trace}\left(\alpha\sum_{i=1}^{N}\lambda_i^2\boldsymbol{Y}_l^T\boldsymbol{u_i}\boldsymbol{u}_i^T\boldsymbol{Y}_l\right) = \mathrm{trace}\left(\alpha\sum_{i=1}^{N}\lambda_i^2\boldsymbol{u}_i^T\boldsymbol{Y}_l(\boldsymbol{u}_i^T\boldsymbol{Y}_l)^T\right)\\
&= \mathrm{trace}\left(\alpha\sum_{i=1}^{N}\lambda_i^2\left\|\boldsymbol{u}_i^T\boldsymbol{Y}_l\right\|_2^2\right) = \mathrm{trace}\left(\sum_{i=1}^{N}\left\|\boldsymbol{u}_i^T\boldsymbol{H}_{l+1}\right\|_2^2\right)
\end{aligned}
\tag{A19}
$$

Suppose all $\left\|\boldsymbol{u}_i^T\boldsymbol{Y}_l\right\|_2^2$ and $\boldsymbol{u}_i^T\boldsymbol{H}_{l+1}$ are equal, then

$$
\alpha = \frac{N}{\sum\limits_{i=1}^{N}\lambda_i^2} = \frac{N}{\|\hat{A}\|_F^2} = \frac{N}{\sum\limits_{i=1}^{N}\frac{1}{d_i+1}}
\tag{A20}
$$

If we use ReLU activation function, we have

$$
\alpha = \frac{2N}{\sum\limits_{i=1}^{N}\lambda_i^2} = \frac{2N}{\|\hat{A}\|_F^2} = \frac{2N}{\sum\limits_{i=1}^{N}\frac{1}{d_i+1}}
\tag{A21}
$$

References

1. S. Abu-El-Haija, B. Perozzi, A. Kapoor, N. Alipourfard, K. Lerman, H. Harutyunyan, G. Ver Steeg, and A. Galstyan. Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing. In *international conference on machine learning*, pages 21–29. PMLR, 2019.
2. D. Bo, X. Wang, C. Shi, and H. Shen. Beyond low-frequency information in graph convolutional networks. *arXiv preprint arXiv:2101.00797*, 2021.
3. A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko. Translating embeddings for modeling multi-relational data. In *Advances in neural information processing systems*, pages 2787–2795, 2013.
4. M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst. Geometric deep learning: Going beyond euclidean data. *arXiv*, abs/1611.08097, 2016.
5. M. Chen, Z. Wei, Z. Huang, B. Ding, and Y. Li. Simple and deep graph convolutional networks. *arXiv preprint arXiv:2007.02133*, 2020.
6. E. Chien, J. Peng, P. Li, and O. Milenkovic. Adaptive universal generalized pagerank graph neural network. In *International Conference on Learning Representations. https://openreview. net/forum*, 2021.
7. F. R. Chung and F. C. Graham. *Spectral graph theory*. Number 92. American Mathematical Soc., 1997.
8. P. Dayan. Improving generalization for temporal difference learning: The successor representation. *Neural Computation*, 5(4):613–624, 1993.
9. M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *arXiv*, abs/1606.09375, 2016.

10. V. N. Ekambaram. *Graph structured data viewed through a fourier lens*. University of California, Berkeley, 2014.
11. A. Frommer, K. Lund, and D. B. Szyld. Block Krylov subspace methods for functions of matrices. *Electronic Transactions on Numerical Analysis*, 47:100–126, 2017.
12. J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1263–1272. JMLR. org, 2017.
13. X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.
14. A. Grover and J. Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864. ACM, 2016.
15. M. H. Gutknecht and T. Schmelzer. The block grade of a block krylov space. *Linear Algebra and its Applications*, 430(1):174–185, 2009.
16. W. L. Hamilton. Graph representation learning. *Synthesis Lectures on Artifical Intelligence and Machine Learning*, 14(3):1–159, 2020.
17. W. L. Hamilton, R. Ying, and J. Leskovec. Inductive representation learning on large graphs. *arXiv*, abs/1706.02216, 2017.
18. K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.
19. G. E. Hinton, S. Osindero, and Y.-W. Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.
20. Y. Hou, J. Zhang, J. Cheng, K. Ma, R. T. Ma, H. Chen, and M.-C. Yang. Measuring and improving the use of graph information in graph neural networks. In *International Conference on Learning Representations*, 2019.
21. C. Hua, S. Luan, Q. Zhang, and J. Fu. Graph neural networks intersect probabilistic graphical models: A survey. *arXiv preprint arXiv:2206.06089*, 2022.
22. M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397*, 2016.
23. D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
24. T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *arXiv*, abs/1609.02907, 2016.
25. J. Klicpera, A. Bojchevski, and S. Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. *arXiv preprint arXiv:1810.05997*, 2018.
26. M. Klissarov and D. Precup. Diffusion-based approximate value functions. In *the 35th international conference on Machine learning ECA Workshop*, 2018.
27. M. Klissarov and D. Precup. Graph convolutional networks as reward shaping functions. In *ICLR 2019, Representation Learning on Graphs and Manifolds Workshop*, 2019.
28. M. Klissarov and D. Precup. Reward propagation using graph convolutional networks. *arXiv preprint arXiv:2010.02474*, 2020.
29. T. D. Kulkarni, A. Saeedi, S. Gautam, and S. J. Gershman. Deep successor reinforcement learning. *arXiv preprint arXiv:1606.02396*, 2016.
30. Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *nature*, 521(7553):436, 2015.
31. Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
32. Q. Li, Z. Han, and X. Wu. Deeper insights into graph convolutional networks for semi-supervised learning. *arXiv*, abs/1801.07606, 2018.
33. R. Li, S. Wang, F. Zhu, and J. Huang. Adaptive graph convolutional neural networks. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.
34. R. Liao, Z. Zhao, R. Urtasun, and R. S. Zemel. Lanczosnet: Multi-scale deep graph convolutional networks. *arXiv*, abs/1901.01484, 2019.
35. D. Lim, X. Li, F. Hohne, and S.-N. Lim. New benchmarks for learning on non-homophilous graphs. *arXiv preprint arXiv:2104.01404*, 2021.

36. M. Liu, Z. Wang, and S. Ji. Non-local graph neural networks. *arXiv preprint arXiv:2005.14612*, 2020.
37. I. Loshchilov and F. Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.
38. S. Luan, X.-W. Chang, and D. Precup. Revisit policy optimization in matrix form. *arXiv preprint arXiv:1909.09186*, 2019.
39. S. Luan, C. Hua, Q. Lu, J. Zhu, X.-W. Chang, and D. Precup. When do we need gnn for node classification? *arXiv preprint arXiv:2210.16979*, 2022.
40. S. Luan, C. Hua, Q. Lu, J. Zhu, M. Zhao, S. Zhang, X.-W. Chang, and D. Precup. Is heterophily a real nightmare for graph neural networks on performing node classification?
41. S. Luan, C. Hua, Q. Lu, J. Zhu, M. Zhao, S. Zhang, X.-W. Chang, and D. Precup. Is heterophily a real nightmare for graph neural networks to do node classification? *arXiv preprint arXiv:2109.05641*, 2021.
42. S. Luan, C. Hua, Q. Lu, J. Zhu, M. Zhao, S. Zhang, X.-W. Chang, and D. Precup. Revisiting heterophily for graph neural networks. *Advances in neural information processing systems*, 35:1362–1375, 2022.
43. S. Luan, C. Hua, M. Xu, Q. Lu, J. Zhu, X.-W. Chang, J. Fu, J. Leskovec, and D. Precup. When do graph neural networks help with node classification: Investigating the homophily principle on node distinguishability. *arXiv preprint arXiv:2304.14274*, 2023.
44. S. Luan, M. Zhao, X.-W. Chang, and D. Precup. Break the ceiling: Stronger multi-scale deep graph convolutional networks. *Advances in neural information processing systems*, 32, 2019.
45. S. Luan, M. Zhao, X.-W. Chang, and D. Precup. Training matters: Unlocking potentials of deeper graph convolutional neural networks. *arXiv preprint arXiv:2008.08838*, 2020.
46. S. Luan, M. Zhao, C. Hua, X.-W. Chang, and D. Precup. Complete the missing half: Augmenting aggregation filtering with diversification for graph convolutional networks. *arXiv preprint arXiv:2008.08844*, 2020.
47. S. Luan, M. Zhao, C. Hua, X.-W. Chang, and D. Precup. Complete the missing half: Augmenting aggregation filtering with diversification for graph convolutional neural networks. *arXiv preprint arXiv:2212.10822*, 2022.
48. T. Maehara. Revisiting graph neural networks: All we have is low-pass filters. *arXiv preprint arXiv:1905.09550*, 2019.
49. S. Mahadevan. Proto-value functions: Developmental reinforcement learning. In *Proceedings of the 22nd international conference on Machine learning*, pages 553–560. ACM, 2005.
50. F. Monti, D. Boscaini, J. Masci, E. Rodola, J. Svoboda, and M. M. Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5115–5124, 2017.
51. H. Pei, B. Wei, K. C.-C. Chang, Y. Lei, and B. Yang. Geom-gcn: Geometric graph convolutional networks. *arXiv preprint arXiv:2002.05287*, 2020.
52. B. Perozzi, R. Al-Rfou, and S. Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710. ACM, 2014.
53. U. N. Raghavan, R. Albert, and S. Kumara. Near linear time algorithm to detect community structures in large-scale networks. *Physical review E*, 76(3):036106, 2007.
54. L. F. Ribeiro, P. H. Saverese, and D. R. Figueiredo. struc2vec: Learning node representations from structural identity. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 385–394, 2017.
55. Z. Sun, Z.-H. Deng, J.-Y. Nie, and J. Tang. Rotate: Knowledge graph embedding by relational rotation in complex space. *arXiv preprint arXiv:1902.10197*, 2019.
56. R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.
57. J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei. Line: Large-scale information network embedding. In *Proceedings of the 24th international conference on world wide web*, pages 1067–1077. International World Wide Web Conferences Steering Committee, 2015.
58. P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio. Graph attention networks. *arXiv*, abs/1710.10903, 2017.
59. T. Wang, M. Bowling, and D. Schuurmans. Dual representations for dynamic programming and reinforcement learning. In *2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, pages 44–51. IEEE, 2007.

60. F. Wu, T. Zhang, A. H. d. Souza Jr, C. Fifty, T. Yu, and K. Q. Weinberger. Simplifying graph convolutional networks. *arXiv preprint arXiv:1902.07153*, 2019.
61. Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu. A comprehensive survey on graph neural networks. *arXiv*, abs/1901.00596, 2019.
62. C. Xu and R. Li. Relation embedding with dihedral group in knowledge graph. *arXiv preprint arXiv:1906.00687*, 2019.
63. K. Xu, C. Li, Y. Tian, T. Sonobe, K.-i. Kawarabayashi, and S. Jegelka. Representation learning on graphs with jumping knowledge networks. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 5453–5462. PMLR, 10–15 Jul 2018.
64. Y. Yan, M. Hashemi, K. Swersky, Y. Yang, and D. Koutra. Two sides of the same coin: Heterophily and oversmoothing in graph convolutional neural networks. *arXiv preprint arXiv:2102.06462*, 2021.
65. S. Zhang, Y. Tay, L. Yao, and Q. Liu. Quaternion knowledge graph embedding. *arXiv preprint arXiv:1904.10281*, 2019.
66. S. Zhang, H. Tong, J. Xu, and R. Maciejewski. Graph convolutional networks: Algorithms, applications and open challenges. In *International Conference on Computational Social Networks*, pages 79–91. Springer, 2018.
67. M. Zhao, Z. Liu, S. Luan, S. Zhang, D. Precup, and Y. Bengio. A consciousness-inspired planning agent for model-based reinforcement learning. *Advances in neural information processing systems*, 34:1569–1581, 2021.
68. M. Zhao, S. Luan, I. Porada, X.-W. Chang, and D. Precup. Meta-learning state-based eligibility traces for more sample-efficient policy evaluation. *arXiv preprint arXiv:1904.11439*, 2019.
69. J. Zhu, R. A. Rossi, A. Rao, T. Mai, N. Lipka, N. K. Ahmed, and D. Koutra. Graph neural networks with heterophily. *arXiv preprint arXiv:2009.13566*, 2020.
70. J. Zhu, Y. Yan, L. Zhao, M. Heimann, L. Akoglu, and D. Koutra. Beyond homophily in graph neural networks: Current limitations and effective designs. *Advances in Neural Information Processing Systems*, 33, 2020.
71. J. Zhu, Y. Yan, L. Zhao, M. Heimann, L. Akoglu, and D. Koutra. Generalizing graph neural networks beyond homophily. *arXiv preprint arXiv:2006.11468*, 2020.