# From Local to Global: Spectral-Inspired Graph Neural Networks

**Anonymous Author(s)**
Affiliation
Address
email

## Abstract

Graph Neural Networks (GNNs) are powerful deep learning methods for Non-Euclidean data. Popular GNNs are message-passing algorithms (MPNNs) that aggregate and combine signals in a local graph neighborhood. However, shallow MPNNs tend to miss long-range signals and perform poorly on some heterophilous graphs, while deep MPNNs can suffer from issues like over-smoothing or over-squashing. To mitigate such issues, existing works typically borrow normalization techniques from training neural networks on Euclidean data or modify the graph structures. Yet these approaches are not well-understood theoretically and could increase the overall computational complexity. In this work, we draw inspirations from spectral graph embedding and propose `PowerEmbed` — a simple layer-wise normalization technique to boost MPNNs. We show `PowerEmbed` can provably express the top-$k$ leading eigenvectors of the graph operator, which prevents over-smoothing and is agnostic to the graph topology; meanwhile, it produces a list of representations ranging from local features to global signals, which avoids over-squashing. We apply `PowerEmbed` in a wide range of simulated and real graphs and demonstrate its competitive performance, particularly for heterophilous graphs.

## 1 Introduction

We consider the graph representation learning problem. Given the graph $G$ of $n$ nodes with adjacency matrix $A \in \mathbb{R}^{n\times n}$ and node features $X \in \mathbb{R}^{n\times p}$, a function $f$ embeds the graph $G$ by mapping $(A,X)$ to $\mathbf{h} \in \mathbb{R}^{n\times k}$, where each node is associated with an embedding vector $h_i \in \mathbb{R}^k$. The task is to find a permutation equivariant function $f$ (i.e. $f(\Pi A \Pi^\top, \Pi X) = \Pi f(A, X)$ for all permutation matrices $\Pi$), while preserving the graph structure: similar nodes in $G$ should be close to each other in the embedding space $\mathbb{R}^k$. The quality of the embedding $\mathbf{h}$ is typically evaluated in the subsequent statistical inference task, such as node classification or link prediction.

Spectral embeddings choose $f$ based on matrix factorization. For example, $f$ could output the top-$k$ eigenvectors of $A$. Spectral embeddings are well-established statistical methods that enjoy nice theoretical properties on random graphs, e.g. stochastic block models [1], latent position random graphs [2]. Nonetheless, spectral methods typically employ a global spectral decomposition, which can be computationally prohibitive on large-scale graphs; they are also brittle on sparse graphs due to the bias in graph spectra estimation [3],

Graph neural networks (GNNs) parameterize $f$ as a neural network, and typically optimize $f$ with label supervision from the subsequent inference task. Popular GNNs are message-passing algorithms (MPNNs) that aggregate and combine signals in a local graph neighborhood. Despite their computational efficiency and scalability, recent works have shown that shallow MPNNs tend to miss long-range signals or global information [4], while deep MPNNs can suffer from issues like over-smoothing (i.e., losing information for the inference task) [5, 6] or over-squashing (i.e.

Submitted to NeurIPS 2022 GLFrontiers Workshop. Do not distribute.

distortion of information from distant nodes) [4, 7]. Moreover, MPNNs seem to perform poorly on several *heterophilous* graphs (i.e., nodes from different classes are more likely to connect, as opposed *homophilous* graphs) [8–10]. To mitigate such issues, existing MPNNs typically borrow optimization techniques from training standard neural networks [11, 12] or modify the graph structures [13, 14] with increasingly complicated architectures [15, 16]. Nevertheless, these approaches do not fully exploit the nature of the graph learning problem. To encode the right kind of inductive bias, there are two major classes of theoretically-driven GNNs: (1) higher-order GNNs that are based on the Weisfeiler-Lehman hierarchy [17–26]; (2) spectral GNNs that are motivated from graph Fourier transform [27–31]. Despite their nice theoretical properties, they typically incur higher computational costs (e.g., due to operations on higher-order tensors in higher-order GNNs, or additional computations to parameterize the spectral filter functions in spectral GNNs).

### 1.1 Our Contributions

Inspired by spectral embeddings, we propose a simple normalization technique in MPNNs to encode global spectra information. Specifically:

1. We propose an unsupervised representation learning method `PowerEmbed` (Algorithm 1) by augmenting the message-passing layer with a simple normalization step; `PowerEmbed` can express the top-$k$ eigenvectors of the graph operator, which is agnostic to the graph topology and works well for graphs with homophily or heterophily.
2. We couple `PowerEmbed` with an inception network (Algorithm 2) to learn the rich representations that interpolate from local message-passing features to global spectral information, which provably avoids over-smoothing and over-squashing.
3. We demonstrate numerically that our simple techniques achieve competitive performance for node classification in a wide range of simulated and real-world graphs.

## 2 Related Work

**Normalization techniques in MPNNs.** Many normalization techniques have been proposed to design deeper MPNNs: [32, 33] proposed a layer-wise mean subtraction and rescaling to maintain pairwise node embedding distances; [34] proposed a layer-wise node normalization depending on their (predicted) clusters; [12, 35] adapted normalization techniques from deep learning and propose node-wise, batch-wise, and graph-wise normalization methods.

**Injecting global information in MPNNs.** Global properties of the graph can be encoded as inputs to MPNNs, such as using spectral embeddings as node features [16, 36], sampling anchor nodes [37], or using other low-pass geometric features [38]. On the other hand, the global information may be learned by using specific architectural choices, such as residual connections [11, 39], attention mechanisms [14, 40–42], or transformers [15, 16, 43]. Furthermore, to speed up the long-range information flow, the original graph can be modified, such as graph sparsification [44–46], graph sampling [47, 48], or localized subgraph extraction [49]. Finally, multiscale graph representation learning can be used, such as scattering transform [50, 29], or hierarchical approaches [51, 52].

**GNNs for community detection.** Several GNNs have been proposed to tackle community detection in stochastic block models. These include semi-supervised approaches such as line-GNNs [53], diff-pooling [52], and GNNs combined with Markov random field models [54]. There are also unsupervised approaches such as graph-pooling [55].

To the best of our knowledge, the statistical properties of the aforementioned MPNNs are not well-understood. Moreover, many of them incur high computational costs. This motivates our work to propose simple techniques that enjoy provable statistical guarantees and computational efficiency.

## 3 Preliminaries

**Notations.** Let the graph $G = (V, E, X, Y)$ with node set $V$, edge set $E$, node input features $X \in \mathbb{R}^{n\times p}$ and node labels $Y \in \mathbb{R}^n$, where $|V| = n$. Let $A, D$ be the adjacency matrix and the degree matrix of $G$. Let $\bar{A} = \tilde{D}^{-0.5}\tilde{A}\tilde{D}^{-0.5}$ be the symmetric graph Laplacian, where $\tilde{A} =$

$A + I, \tilde{D} = D + I$. Let $A_{rw} = \tilde{D}^{-1}\tilde{A}$ be the random walk graph Laplacian. Let $\frac{U}{\|U[:,k]\|}$ be the column normalized form of matrix $U$.

**Inference problem setup.** To evaluate the graph embedding $\mathbf{h} = f(A, X)$, we consider the node classification task: Given a graph $G$ of $n$ nodes with $n_t < n$ training labels, the goal is to predict the remaining $n - n_t$ test labels. Let $\Omega$ and $\Omega^{\perp}$ the indices corresponding to training nodes and test nodes respectively ($V = V_{\Omega} \cup V_{\Omega^{\perp}}$). Then the classifier is trained on the training set $(\mathbf{h}_{\Omega}, Y_{\Omega})$ and evaluated on the test set $(\mathbf{h}_{\Omega^{\perp}}, Y_{\Omega^{\perp}})$. If the graph embedding is a list of features $P = [X, \mathbf{h}^{(1)} \ldots, \mathbf{h}^{(L)}]$, then we let $P_{\Omega}, P_{\Omega^{\perp}}$ denote the training and test set features, where

$$P_{\Omega} = [X_{\Omega}, \mathbf{h}_{\Omega}^{(1)}, \ldots, \mathbf{h}_{\Omega}^{(L)}]. \tag{1}$$

**Definition 3.1** (Stochastic Block Model (SBM))**.** A graph $A$ with $n$ nodes is a random SBM graph if it is sampled as

$$A \sim Bernoulli(P),\ P = ZBZ^{\top}, \tag{2}$$

where $Z \in \mathbb{R}^{n \times K}$ is a membership matrix such that $Z_{i,k}$ is 1 if the $i$-th node belongs to the $k$-th class, $\|Z_{i,\cdot}\|_1 = \sum_{k=1}^{K} |Z_{i,k}| = 1$, and $B \in [0,1]^{K \times K}$ is a full-rank matrix representing the block connection probability.

**Definition 3.2** (2B-SBM with Gaussian node features)**.** A two-block symmetric SBM (2B-SBM) is given by:

$$Z_{i,\cdot} = \begin{cases} [1,0] & \text{if } i \in [n/2] \\ [0,1] & \text{otherwise.} \end{cases},\ B = \begin{bmatrix} p & q \\ q & p \end{bmatrix}, \tag{3}$$

where $p, q \in (0,1), p \neq q$. The node features in block $k \in \{0,1\}$ are sampled from a multivariate Gaussian $\mathcal{N}(\mu_k, \Sigma_k)$ and stored in a node feature matrix $X \in \mathbb{R}^{n \times k}$.

The SBM model is a canonical random graph model with planted clusters, which has been widely studied in the context of community detection [3, 2, 1, 56]. In particular, the leading eigenvectors of the graph adjacency $A$ or the graph Laplacians $A_{rw}, \bar{A}$ provably encode community membership, provided that the graphs are sufficiently dense [57, 1]. To see this, consider the 2B-SBM model in (3) without node features, then the matrix $P = \mathbb{E}(A)$ has three eigenvalues, ranked by magnitude as $(p+q)/2, (p-q)/2, 0$, where 0 has multiplicity $n-2$. The leading eigenvector is a constant vector, whereas the second eigenvector $u_2(P) = [\mathbf{1}_{n/2}; -\mathbf{1}_{n/2}]$ reveals the community structure. Now, the random graph $A$ can be viewed as a perturbation of its expectation $P$. Standard results in concentration of measure show that the eigenvalues and eigenvectors of $A$ are close to those of $P$ for sufficiently dense graphs (the average node degree needs to be of the order $\Omega(\frac{n}{\log n})$). However, spectral methods fail to detect communities when the graphs are very sparse ($p = q = O(1/n)$), and there are known information-theoretical thresholds for certain stochastic block models (see [3] for a recent survey).

In this work, we consider classical spectral embedding methods defined as follows: Let $A = USU^{\top}, XX^{\top} = \tilde{U}\Sigma\tilde{U}^{\top}$ be the spectral decomposition of the graph $A$ and the covariance matrix $XX^{\top}$. Then

$$\mathbf{h}^{ASE} = U_k, \quad \mathbf{h}^{\mathrm{cov}(X)} = \tilde{U}_k, \quad \mathbf{h}^{A_X} = [\mathbf{h}^{ASE}; \mathbf{h}^{\mathrm{cov}(X)}]. \tag{4}$$

Crucially, spectral methods that are based on the top-$k$ eigenvectors ranked by absolute *magnitude* have no assumption on the graph being homophilous (e.g., $p > q$) or heterophilous (e.g., $p < q$); They perform consistently well on both cases by "looking globally" at the graph structure. In contrast, message-passing methods are local algorithms that work well in homophilous graphs, but they tend to overlook global signals and struggle to learn in heterophilous graphs.

**Definition 3.3** (Message-Passing Neural Network (MPNN))**.** A $L$-layer MPNN initializes the embedding $\mathbf{h}^{(0)} = X$. At each iteration $l$, the embedding of node $i$ is updated as

$$\mathbf{h}_i^{(l)} = \phi\left(\mathbf{h}_i^{(l-1)}, \sum_{j \in \mathcal{N}(i)} \psi(\mathbf{h}_i^{(l-1)}, \mathbf{h}_j^{(l-1)})\right), \tag{5}$$

where $\phi, \psi$ are the update and message functions, and $\mathcal{N}(i)$ denotes the neighbors of node $i$.

For example, graph convolutional networks (GCN [58]) choose $\phi$ as a nonlinear activation function $\sigma$ and $\psi(i,j) = \frac{1}{\sqrt{\deg(i)\deg(j)}} W^{(l-1)} \mathbf{h}_j^{(l-1)}$, where $W^{(l-1)}$ is a learnable weight matrix. In other words:

$$\mathbf{h}^{(l)} = \sigma(\bar{A}\mathbf{h}^{(l-1)} W^{(l-1)}). \tag{6}$$

GCN can be further simplified by removing the nonlinearity $\sigma$, which is known as simple graph convolution (SGC [59]):

$$\mathbf{h}^{(L)} = \bar{A}\mathbf{h}^{(L-1)} W^{(L-1)} = \bar{A}^L X \big(W^{(L-1)} \dots W^{(0)}\big). \tag{7}$$

To train a classifier on top of the SGC embedding in (7), we can absorb the weight matrices $W^{(l-1)}$ to the downstream classifier. In other words, SGC can be viewed as producing an unsupervised representation $\bar{A}^L X$ followed by learning a classifier. However, such representation fails to distinguish nodes from different classes when $L$ is sufficiently large, known as the over-smoothing issue of deep GNNs.

**Definition 3.4** (Over-smoothing [5])**.** Assume the graph has one connected component. Then

$$\lim_{L\to\infty} A_{rw}^L X = [\mathbf{1}_n; , \dots, \mathbf{1}_n]; \quad \lim_{L\to\infty} \bar{A}^L X = D^{-1}[\mathbf{1}_n; , \dots, \mathbf{1}_n]. \tag{8}$$

Over-smoothing describes that for a sufficiently deep SGC model, its node embedding converges to the largest eigenvector which only encodes connected component and degree information, but tends to miss the community structure present in subsequent eigenvectors. It has also been extended and defined for nonlinear GCNs [6, 60] and empirically observed in deep MPNNs [32, 44, 39].

On the other hand, over-squashing refers to the distortion of node features in the learned graph representation in a deep GNN model [4, 7], which can also hurt subsequent inference when the node features carry useful signals.

**Definition 3.5** (Over-squashing [7])**.** Let $h_i^{(L)} = h_i^{(L)}(x_1, \dots, x_n)$ be the output for node $i$ of a $L$-layer MPNN with input features $\{x_i\}_{i=1}^n$. Then the over-squashing effect (for node $i$ with respect to node $s$) is measured by the Jacobian $\partial h_i^{(L)} / \partial x_s$.

Thus, the smaller the Jacobian value, the more node feature information is "squashed" out in the embedding.

# 4 PowerEmbed

In this section, we first show how to express the top-$k$ eigenvectors of a graph operator using message-passing power iteration (`PowerEmbed` in Algorithm 1). Next, we describe how to exploit the rich representations produced by `PowerEmbed` by using the inception neural network architecture (Algorithm 2). Our framework is summarized in Figure 1.

**Algorithm 1** PowerEmbed

**Require:** a graph operator $S \in \mathbb{R}^{n\times n}$, node features $X \in \mathbb{R}^{n\times k}$, a list $P = [X]$.
  **Initialize:**
    $U(t) = X$
  **for** t = 0 to L-1 **do**
    $\tilde{U}(t+1) = S\,U(t)$ [message-passing]
    $U(t+1) = \tilde{U}(t+1)[\tilde{U}(t+1)^\top \tilde{U}(t+1)]^{-1}$ [Normalization]
    Append $\frac{U(t+1)}{\|U(t+1)[:,k]\|}$ to $P$
  **end for**
  **return** $P$

Algorithm 1 is motivated from the natural power iteration [61], which provably returns the top-$k$ eigenvectors of a symmetric matrix $S$ (up to an orthogonal transformation) upon convergence, with mild assumptions that the input features $X$ has full column rank and the $S$ has an eigen-gap, i.e., $\lambda_k \neq \lambda_{k+1}$ where $\lambda_k$ denotes the $k$-th eigenvalue of $S$ (ranked by magnitude). Crucially, the

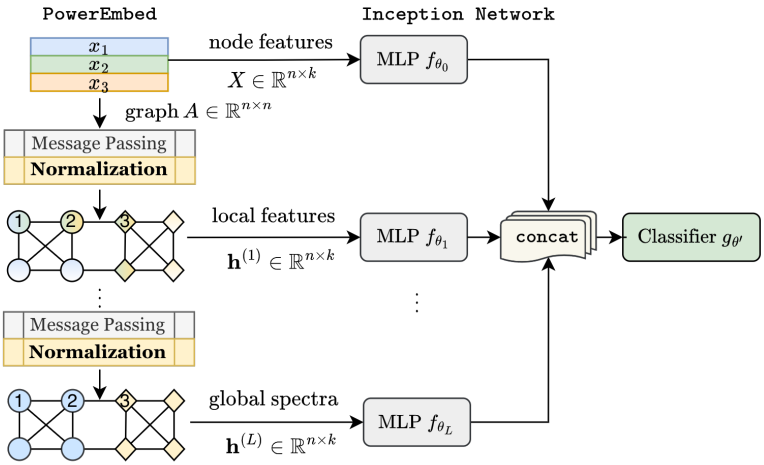


Figure 1: `PowerEmbed` produces a list of embeddings ranging from local neighbor-averaged features to global spectral information, which is then jointly learned by using an inception neural network.

normalization step ensures orthogonality of the output at each iteration and only requires inverting a $k \times k$ matrix (which may be further improved using variants of QR factorization). The convergence rate of the $k$-th eigenvector depends on $\frac{\lambda_{k+1}}{\lambda_k}$, which is fast provided with a large eigen-gap $|\lambda_k| \gg |\lambda_{k+1}|$. In practice, if the original node features live in high-dimensional space (e.g., more features than the number of nodes), we first perform a dimensionality reduction to obtain the best rank-$k$ approximation of $X$ before running Algorithm 1 to speed up convergence.

Unlike existing work that requires pre-computed eigenvectors as input [16, 36, 62], our approach implicitly computes eigenvectors using power iteration to improve efficiency while maintaining expressivity. `PowerEmbed` extracts both *local* features from the first few iterations and *global* information from the last few iterations (i.e., the top-$k$ eigenvectors). These rich features are jointly learned using the inception network (Algorithm 2). Therefore, the presence of local features (including the input features) avoids over-squashing (Definition 3.5); the ability to express top-$k$ eigenvectors of a graph operator also avoids over-smoothing (Definition 3.4) and makes it agnostic to the graph topology.

**Algorithm 2** Node Classification with Inception Networks on `PowerEmbed` Representations

**Require:** Training features $P_\Omega$ (1), training labels $Y_\Omega$, loss function $\ell$, step size $\eta$, hidden dimension $k'$, number of classes $K$, epochs $T$.
  **Initialize:**
    A list of Multilayer Perceptions $\texttt{MLPs} = [f_{\theta_0}, \ldots, f_{\theta_L}]$ where $f_{\theta_i} : \mathbb{R}^k \to \mathbb{R}^{k'}$,
    a classifier $g_{\theta'} : \mathbb{R}^{(L+1)k'} \to \mathbb{R}^K$.
  **for** t = 0 to T **do**
    $H = \left[ f_{\theta_0}\left(P_\Omega[0]\right), \ldots, f_{\theta_L}\left(P_\Omega[L]\right) \right]$
    $H' = \texttt{concat}(H) \in \mathbb{R}^{m\times(L+1)k'}$
    $\hat{Y} = g_{\theta'}(H')$
    $L = \ell(\hat{Y}, Y_\Omega)$
    $\theta = \theta - \eta \nabla_\theta L$ for all $\theta \in \{\theta_0, \ldots, \theta_L, \theta'\}$
  **end for**
  **return** $g_{\theta'} \circ \texttt{MLPs}$

The following propositions, proven in Appendix A are basic facts of graph theory. Proposition 4.1 applies the convergence results from [61] to the graph operators used in Algorithm 1, and it shows that `PowerEmbed` can express the top-$k$ eigenvectors of the common graph operators. Proposition 4.2 shows that our graph representation learning approach is permutation equivariant [63].

**Proposition 4.1.** *Consider a graph $G$ with graph operator $S \in \{A, \bar{A}, A_{rw}\}$ and feature matrix $X \in \mathbb{R}^{n\times k}$. If $X$ is full (column) rank and the $k$-th and $(k+1)$-th eigenvalues of $S$ are distinct, then the last iterate $U(t+1)$ from* `PowerEmbed` *converges to the top $k$ eigenvectors of $S$ when $t \to \infty$ (up to an orthogonal transformation in $O(k)$).*

**Proposition 4.2.** *The method* `PowerEmbed` *(Algorithm 1) followed by the inception neural network $g_{\theta'} \circ$* `MLPs` *(from Algorithm 2) is permutation equivariant.*

# 5 Numerical Experiments

## 5.1 Baselines

To evaluate the benefits of injecting global spectra information into MPNNs, We compare `PowerEmbed` with three classes of baselines.

*Unnormalized counterparts.* Unsupervised MPNNs without layer-wise normalization using the same inception network architecture (Algorithm 2) : SIGN [64] that uses $A_{rw}$, and a modified version of SGC [59] that uses $\bar{A}$ which we call "SGC(Incep)". These baselines illustrate the effects of adding the proposed layer-wise normalization in Algorithm 1.

*Spectral methods.* Standard spectral embedding methods described in (4) that use purely global (spectral) information. These global baselines measure the effects of adding local information (from neighbor-aggregated features) in `PowerEmbed`.

*Semi-supervised MPNNs.* Popular models including graph convolutional networks (GCN) [58] and graph attention networks (GAT) [65]) for benchmark purpose; MPNNs that encode long-range information including geometric GCN (GEOM-GCN) [8], generalized page-rank GNN (GPR-GNN) [66], and GCN via initial residual and identity mapping (GCNII) [67]; These baselines are chosen to gauge the effects of injecting *spectral* information as opposed to long-range spatial information.

We remark that all unsupervised methods (i.e., `PowerEmbed`, unnormalized counterparts, spectral methods) produce an embedding matrix $\mathbf{h} \in \mathbb{R}^{n\times k}$ (or a list of features $P$) without label supervision, followed by fitting a neural network classifier: For a single embedding matrix $\mathbf{h}$, we fit a 2-layer MLP with ReLU activation; for the list $P = [X, \mathbf{h}_1, \ldots, \mathbf{h}_L]$, we fit the inception network (Algorithm 2). On the other hand, all semi-supervised MPNNs optimize the network end-to-end.

All the simulations and code are available in this repository [1].

## 5.2 Synthetic graphs

In this section, we simulate random graphs following the stochastic block models defined in (3), with node features $\mu_0 = [1,1], \mu_1 = -\mu_0$. We consider the (binary) node classification setting described in Section 3 with $n = 500$ nodes and a 10/90 train/test split.

Figure 2 illustrates the convergence of the last iterate of `PowerEmbed` to the top-2 eigenvectors of a 2B-SBM expected adjacency matrix, with different choices of the parameters $p$,$q$ that controls the block probability matrix $B$ defined in (3). The convergence is fast in the dense case where $\lambda_i(A) \approx \lambda_i(P), i \in \{1,2\}$ and are well-separated from the remaining $n-2$ eigenvalues. In the sparse case, the eigen-gap $\lambda_2(A) - \lambda_3(A)$ is smaller due to higher (random) noise in the sparse random graph $A$, and thus the convergence of $\hat{u}_2$ is slower.

Figure 3 shows that `PowerEmbed` is able to recover the spectral embeddings in dense graphs (left column), and outperforms spectral embedding in sparse graphs (right column). This can be explained by `PowerEmbed`'s ability to leverage both local node features and global structural information. Moreover, `PowerEmbed` and spectral methods perform consistently well on graphs with homophily (top row) and heterophily (bottom row). This is in contrast to standard MPNNs that suffer from over-smoothing (e.g., "GCN-10") and fail in sparse heterophilous graphs (e.g., "GCN-5" performs much worse in heterophilous graphs in (d) compared to homophilous graph in (b) while "Power(Lap)-10" performs steadily well). Comprehensive simulation results including other baselines can be found in Appendix C.1.

[1] https://anonymous.4open.science/r/spectral-inspired-gnn-BE9D

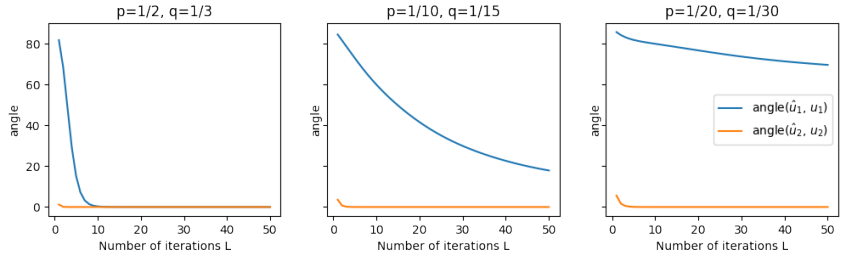


Figure 2: Convergence of the estimated eigenvectors $\hat{u}_i$ from Algorithm 1 is faster for denser graphs and slower for sparse graphs. We report the angles between the true top-2 eigenspaces and estimated top-2 eigenspaces, averaged over 30 random runs.

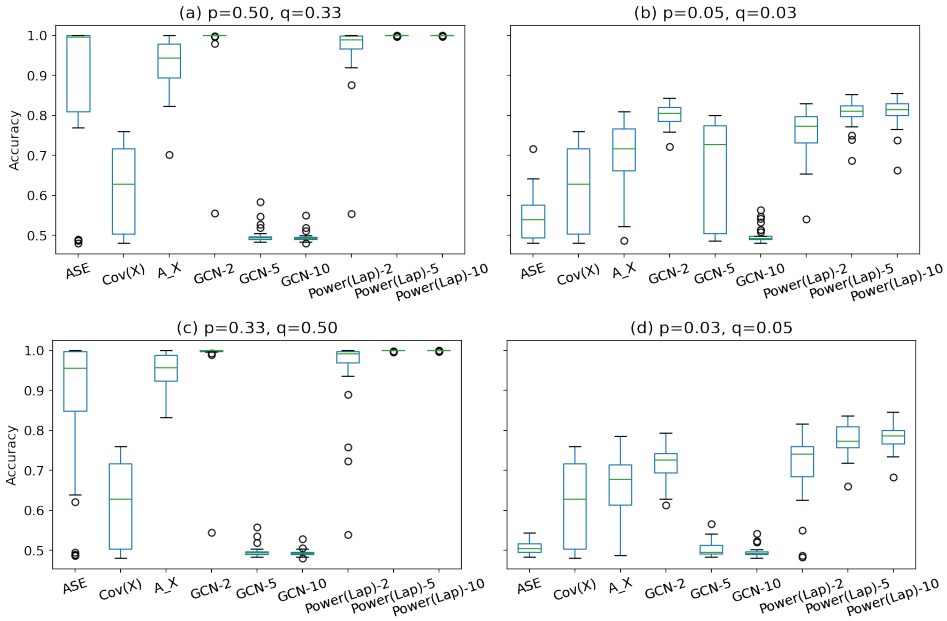


Figure 3: `PowerEmbed` enjoys the same performance guarantee as spectral embedding methods in dense graphs (left) with homophily (top-left) and heterophily (bottom-left); It outperforms spectral embedding methods in sparse graphs (right). "ASE", "Cov(X)", "A_X" are global spectral methods defined in (4). Standard MPNNs suffer from over-smoothing (e.g., "GCN-10") and perform much worse in sparse heterophilous graphs (e.g., "GCN-5"). "Method-$k$" denotes the corresponding method with $k$ iterations. Boxplots are shown for each method, with the mean indicated by green bars and outliers displayed as circles.

### 5.3 Real-world graphs

We report node classification results of `PowerEmbed` on 10 graph benchmark datasets, including 5 heterophily graphs and 5 homophily graphs with varying densities. We summarize key statistics of each graph in Table 1, Table 2 and describe their details in Appendix B.2.

For most graphs except Computers and Photo, we use the same data splits (48/32/20 for train/validation/test) from [8] released in Pytorch Geometric [68]; for Computers and Photo that were not studied in [8], we use the same data split (60/20/20) as in [69]. Note that most of the node features are high-dimensional bag-of-words vectors. Thus, for unsupervised embedding methods, we use the top-$k$ eigenvector of the feature covariance as the input feature, where $k = 10$ for small-size graphs including Wisconsin, Texas, and Cornell, and $k = 100$ elsewhere; for semi-supervised methods, we use the original node features following the original model architectures. Self-loops are used for all graphs. We train all models using ADAM optimizer [70] and full-batch gradient descent with the

Table 1: Heterophily graph experiments: `PowerEmbed` and spectral methods (third block) significantly outperform unsupervised MPNN counterparts (second block) and semi-supervised MPNNs (last block) in dense heterophilous graphs; `PowerEmbed` works better than spectral methods on sparse heterophilous graphs. We report the mean accuracy $\pm$ stderr over 10 data splits. "Method-$k$" indicates the corresponding method with $k$ iterations. The $*$ results are obtained from [8].

| Graph | Squirrel | Chameleon | Wisconsin | Texas | Cornell |
|---|---|---|---|---|---|
| Density | 38.16 | 13.8 | 1.86 | 1.61 | 1.53 |
| Homophily | 0.22 | 0.23 | 0.21 | 0.11 | 0.3 |
| #Nodes | 5,201 | 2,277 | 251 | 183 | 183 |
| #Edges | 198,493 | 31,421 | 466 | 295 | 280 |
| #Features | 2,089 | 2,325 | 1,703 | 1703 | 1,703 |
| #Classes | 5 | 5 | 5 | 5 | 5 |
| Power-10 | **53.53 $\pm$ 0.41** | **64.98 $\pm$ 0.55** | 74.71 $\pm$ 1.74 | 73.51 $\pm$ 2.05 | 75.14 $\pm$ 2.50 |
| Power(RW)-10 | 44.58 $\pm$ 0.52 | 61.64 $\pm$ 0.43 | 75.49 $\pm$ 1.71 | 75.68 $\pm$ 1.21 | 72.97 $\pm$ 1.58 |
| Power(Lap)-10 | 42.32 $\pm$ 0.37 | 62.17 $\pm$ 0.41 | 74.71 $\pm$ 1.74 | 74.05 $\pm$ 2.10 | 77.03 $\pm$ 1.54 |
| Power-2 | 52.13 $\pm$ 0.55 | 64.47 $\pm$ 0.76 | 75.29 $\pm$ 1.47 | **79.19 $\pm$ 1.33** | 76.76 $\pm$ 1.63 |
| Power(RW)-2 | 45.92 $\pm$ 0.48 | 59.67 $\pm$ 0.62 | 77.45 $\pm$ 0.89 | 76.22 $\pm$ 1.31 | 75.41 $\pm$ 1.85 |
| Power(Lap)-2 | 43.06 $\pm$ 0.56 | 60.00 $\pm$ 0.62 | **78.43 $\pm$ 1.59** | 77.03 $\pm$ 1.54 | **78.30 $\pm$ 1.58** |
| SGC(Incep)-10 | 37.07 $\pm$ 0.55 | 55.11 $\pm$ 0.82 | 75.29 $\pm$ 1.04 | 75.68 $\pm$ 1.95 | 75.68 $\pm$ 1.83 |
| SIGN-10 | 38.47 $\pm$ 0.42 | 60.22 $\pm$ 0.72 | 75.29 $\pm$ 1.45 | 73.51 $\pm$ 2.02 | 75.68 $\pm$ 1.21 |
| SGC(Incep)-2 | 35.33 $\pm$ 0.35 | 54.19 $\pm$ 0.65 | 77.45 $\pm$ 0.89 | 76.76 $\pm$ 1.34 | 76.22 $\pm$ 2.07 |
| SIGN-2 | 40.97 $\pm$ 0.35 | 60.11 $\pm$ 0.97 | **78.43 $\pm$ 1.41** | 75.14 $\pm$ 2.02 | 76.76 $\pm$ 1.34 |
| Cov(X) | 33.12 $\pm$ 0.53 | 44.74 $\pm$ 1.00 | 75.69 $\pm$ 1.25 | 77.30 $\pm$ 1.12 | 77.03 $\pm$ 2.27 |
| ASE | 41.46 $\pm$ 0.62 | 57.92 $\pm$ 0.77 | 49.41 $\pm$ 2.09 | 58.65 $\pm$ 1.79 | 56.76 $\pm$ 0.66 |
| A_X | 49.11 $\pm$ 0.37 | 61.97 $\pm$ 0.76 | 77.84 $\pm$ 1.24 | 76.76 $\pm$ 1.22 | 75.95 $\pm$ 2.28 |
| GCN$*$ | 23.96 | 28.18 | 45.88 | 52.16 | 52.7 |
| GAT$*$ | 30.03 | 42.93 | 49.41 | 58.38 | 54.32 |
| Geom-GCN$*$ | 38.14 | 60.9 | 64.12 | 67.57 | 60.81 |
| GCNII-10 | 35.23 $\pm$ 0.50 | 49.96 $\pm$ 0.46 | 59.02 $\pm$ 1.60 | 61.08 $\pm$ 1.49 | 48.38 $\pm$ 1.64 |
| GPR-GNN-10 | 34.51 $\pm$ 1.45 | 52.37 $\pm$ 3.43 | 59.41 $\pm$ 2.75 | 58.92 $\pm$ 2.98 | 52.97 $\pm$ 3.11 |

same hyperparameters: 100 epochs, 0.01 learning rate, 0.5 dropout rate, and no weight decay. Further details can be found in Appendix B.1.

Table 1 summarizes the experimental results on *heterophilous* graph benchmarks: `PowerEmbed` outperforms other baselines on graphs with heterophily, particularly on dense heterophilous (i.e., Squirrel and Chameleon). Figure 4 visualize the effect of increasing the number of message-passing layers: `PowerEmbed` exhibits consistent performance in deeper models and avoids over-smoothing; The ability to express global spectra information via simple normalization significantly boosts performance over unnormalized MPNN counterparts in heterophilous graphs. We also observe that even simple spectral method A_X concatenating leading eigenvectors of graph spectra and node covariance achieves competitive performance, drastically outperforming unnormalized MPNNs and semi-supervised MPNNs. This highlights the utility of learning graph representation from first principle. Further ablation studies on real benchmarks are shown in Appendix C.2.

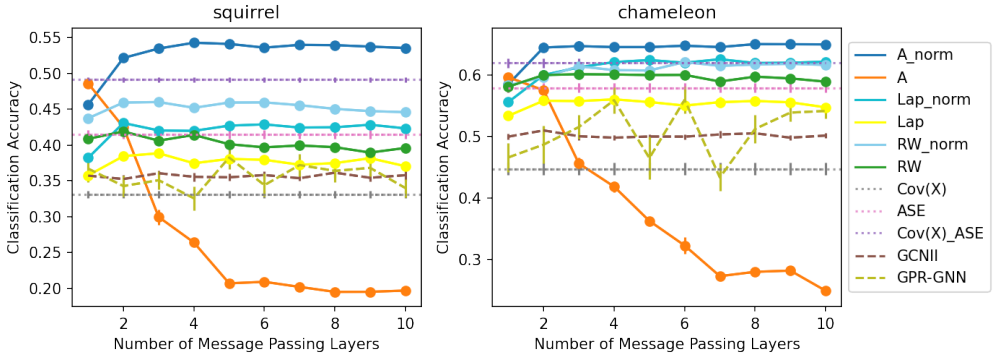


Figure 4: `PowerEmbed` (annotated with "_norm") that adds the normalization step for orthogonality can expressive top-$k$ eigenvectors, which avoids over-smoothing and outperforms other baselines, particularly in heterophilous graphs. Baselines include unnormalized counterparts (SIGN denoted as "RW", SGC(Incep) denoted as "Lap"); spectral methods defined in (4), and semi-supervised MPNNs.

Table 2: Homophilous graph experiments: `PowerEmbed` shows competitive performance as other MPNN baselines. We report the mean accuracy $\pm$ stderr over 10 data splits. "Method-k" indicates the corresponding method with $k$ iterations. The $*$ results are obtained in [8], from [71] for Computers and Photo.

| Graph | Computers | Photo | Coauthor(CS) | Cora | Citeseer |
|---|---|---|---|---|---|
| Density | 35.76 | 31.13 | 8.93 | 1.95 | 1.41 |
| Homophily | 0.8 | 0.85 | 0.83 | 0.81 | 0.74 |
| #Nodes | 13,752 | 7,650 | 18,333 | 2,708 | 3327 |
| #Edges | 491,722 | 238,162 | 163,788 | 5,278 | 4676 |
| #Features | 767 | 745 | 6,805 | 1,433 | 3703 |
| #Classes | 10 | 8 | 15 | 7 | 6 |
| Power-10 | $90.34 \pm 0.22$ | $93.84 \pm 0.17$ | $93.93 \pm 0.11$ | $81.69 \pm 0.50$ | $69.67 \pm 0.63$ |
| Power(RW)-10 | $91.14 \pm 0.19$ | $94.16 \pm 0.20$ | $93.88 \pm 0.13$ | $85.03 \pm 0.44$ | $73.15 \pm 0.54$ |
| Power(Lap)-10 | $91.20 \pm 0.14$ | $93.97 \pm 0.19$ | $94.26 \pm 0.09$ | $84.95 \pm 0.40$ | $72.61 \pm 0.51$ |
| Power-2 | $90.85 \pm 0.15$ | $94.04 \pm 0.21$ | $94.32 \pm 0.11$ | $81.23 \pm 0.52$ | $72.03 \pm 0.41$ |
| Power(RW)-2 | $\mathbf{91.43 \pm 0.13}$ | $94.56 \pm 0.19$ | $94.30 \pm 0.08$ | $83.56 \pm 0.44$ | $72.62 \pm 0.48$ |
| Power(Lap)-2 | $91.33 \pm 0.15$ | $94.58 \pm 0.21$ | $94.75 \pm 0.09$ | $83.52 \pm 0.27$ | $73.27 \pm 0.75$ |
| SGC(Incep)-10 | $90.61 \pm 0.16$ | $94.65 \pm 0.18$ | $94.44 \pm 0.10$ | $84.89 \pm 0.71$ | $73.39 \pm 0.62$ |
| SIGN-10 | $90.65 \pm 0.18$ | $94.63 \pm 0.25$ | $94.05 \pm 0.13$ | $85.45 \pm 0.32$ | $72.78 \pm 0.51$ |
| SGC(Incep)-2 | $90.97 \pm 0.17$ | $94.47 \pm 0.22$ | $94.54 \pm 0.08$ | $83.74 \pm 0.53$ | $72.47 \pm 0.61$ |
| SIGN-2 | $90.89 \pm 0.20$ | $\mathbf{94.59 \pm 0.19}$ | $94.09 \pm 0.12$ | $83.92 \pm 0.43$ | $73.27 \pm 0.53$ |
| Cov(X) | $82.42 \pm 0.14$ | $89.45 \pm 0.26$ | $91.70 \pm 0.15$ | $69.24 \pm 0.56$ | $66.79 \pm 0.62$ |
| ASE | $77.61 \pm 0.20$ | $85.84 \pm 0.22$ | $75.24 \pm 0.21$ | $72.84 \pm 0.48$ | $51.73 \pm 1.67$ |
| A_X | $89.97 \pm 0.21$ | $94.22 \pm 0.22$ | $93.69 \pm 0.13$ | $80.89 \pm 0.56$ | $69.84 \pm 0.71$ |
| GCN$*$ | 90.49 | 93.91 | 93.32 | 85.77 | 73.68 |
| GAT$*$ | - | - | - | **86.37** | 74.32 |
| Geom-GCN$*$ | - | - | - | 84.93 | **75.14** |
| GCNII-10 | $90.75 \pm 0.16$ | $93.86 \pm 0.18$ | $94.32 \pm 0.26$ | $84.14 \pm 0.47$ | $72.17 \pm 0.66$ |
| GPR-GNN-10 | $87.62 \pm 0.85$ | $93.52 \pm 0.39$ | $\mathbf{94.81 \pm 0.27}$ | $85.77 \pm 0.67$ | $73.22 \pm 0.73$ |

Table 2 summarizes the results on *homophilous* graph benchmarks: `PowerEmbed` achieves competitive performance as other MPNN baselines. As discussed in Section 3, MPNNs tend to work well in graphs with homophily due to the inductive bias of aggregating local neighbors' information, while spectral methods could perform sub-optimally, especially in sparse graphs. Together with Table 1, `PowerEmbed` is shown to be agnostic to the graph topology and can flexibly combine local and global information.

# 6 Discussion

In this work, we propose a simple technique `PowerEmbed` that exploits the advantages of global spectral methods and local message-passing algorithms for graph representation learning. Specifically, `PowerEmbed` performs a layer-wise normalization that allows MPNNs to express the top-$k$ eigenvectors of a graph and capture global spectral information. We demonstrate the advantages of our techniques theoretically and empirically on both synthetic and real-world graphs.

Our spectral-inspired GNN opens the door to many interesting future directions. Firstly, we can extend `PowerEmbed` to semi-supervised graph learning tasks allowing learnable weights in the message-passing layers: While `PowerEmbed` can still express the top-$k$ eigenvectors of the graph operators, it may converge to a function of eigenvectors driven by label supervision. We conjecture that the semi-supervised versions may be helpful in certain sparse graphs, where the graph eigenvectors are suboptimal in estimating community structure and the label signals can improve the inference performance (e.g., partially-labeled SBM [72, 73]). Furthermore, since `PowerEmbed` combines global spectral information with local MPNNs, it is natural to consider more powerful versions of `PowerEmbed` that are based on higher-order MPNNs. Yet it remains open to fully understand the relations between graph spatial information (i.e., symmetries) and graph spectral information (e.g., eigenvalues and eigenvectors) [74, 75]. We hope that our approach will inspire future research on principled combinations of global and local methods for graph representation learning.

# References


[1] Jing Lei and Alessandro Rinaldo. Consistency of spectral clustering in stochastic block models. *The Annals of Statistics*, 43(1), feb 2015. doi: 10.1214/14-aos1274. URL https://doi.org/10.1214%2F14-aos1274.

[2] Avanti Athreya, Donniell E Fishkind, Minh Tang, Carey E Priebe, Youngser Park, Joshua T Vogelstein, Keith Levin, Vince Lyzinski, and Yichen Qin. Statistical inference on random dot product graphs: a survey. *The Journal of Machine Learning Research*, 18(1):8393–8484, 2017.

[3] Emmanuel Abbe. Community detection and stochastic block models: recent developments. *The Journal of Machine Learning Research*, 18(1):6446–6531, 2017.

[4] Uri Alon and Eran Yahav. On the bottleneck of graph neural networks and its practical implications. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=i80OPhOCVH2.

[5] Q. Li, Z. Han, and X.-M. Wu. Deeper Insights into Graph Convolutional Networks for Semi-Supervised Learning. In *The Thirty-Second AAAI Conference on Artificial Intelligence*. AAAI, 2018.

[6] Kenta Oono and Taiji Suzuki. Graph neural networks exponentially lose expressive power for node classification. In *International Conference on Learning Representations*, 2019.

[7] Jake Topping, Francesco Di Giovanni, Benjamin Paul Chamberlain, Xiaowen Dong, and Michael M. Bronstein. Understanding over-squashing and bottlenecks on graphs via curvature, 2021.

[8] Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. Geom-gcn: Geometric graph convolutional networks. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=S1e2agrFvS.

[9] Jiong Zhu, Yujun Yan, Lingxiao Zhao, Mark Heimann, Leman Akoglu, and Danai Koutra. Beyond homophily in graph neural networks: Current limitations and effective designs. *Advances in Neural Information Processing Systems*, 33:7793–7804, 2020.

[10] Yao Ma, Xiaorui Liu, Neil Shah, and Jiliang Tang. Is homophily a necessity for graph neural networks? In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=ucASPPD9GKN.

[11] Guohao Li, Matthias Muller, Ali Thabet, and Bernard Ghanem. Deepgcns: Can gcns go as deep as cnns? In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9267–9276, 2019.

[12] Tianle Cai, Shengjie Luo, Keyulu Xu, Di He, Tie-yan Liu, and Liwei Wang. Graphnorm: A principled approach to accelerating graph neural network training. In *International Conference on Machine Learning*, pages 1204–1215. PMLR, 2021.

[13] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2018.

[14] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In Jennifer Dy and Andreas 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. URL https://proceedings.mlr.press/v80/xu18c.html.

[15] Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. Do transformers really perform bad for graph representation? *arXiv preprint arXiv:2106.05234*, 2021.

[16] Devin Kreuzer, Dominique Beaini, Will Hamilton, Vincent Létourneau, and Prudencio Tossou. Rethinking graph transformers with spectral attention. *Advances in Neural Information Processing Systems*, 34, 2021.

[17] Ningyuan Huang and Soledad Villar. A short tutorial on the weisfeiler-lehman test and its variants. In *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8533–8537, 2021. doi: 10.1109/ICASSP39728.2021.9413523.

[18] Ryoma Sato. A survey on the expressive power of graph neural networks. *arXiv preprint arXiv:2003.04078*, 2020.

[19] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

[20] Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 4602–4609, 2019.

[21] Christopher Morris, Gaurav Rattan, and Petra Mutzel. Weisfeiler and leman go sparse: Towards scalable higher-order graph embeddings. *Advances in Neural Information Processing Systems*, 33:21824–21840, 2020.

[22] Cristian Bodnar, Fabrizio Frasca, Yuguang Wang, Nina Otter, Guido F Montufar, Pietro Lio, and Michael Bronstein. Weisfeiler and lehman go topological: Message passing simplicial networks. In *International Conference on Machine Learning*, pages 1026–1037. PMLR, 2021.

[23] Leonardo Cotta, Christopher Morris, and Bruno Ribeiro. Reconstruction for powerful graph representations. *Advances in Neural Information Processing Systems*, 34, 2021.

[24] Zhengdao Chen, Soledad Villar, Lei Chen, and Joan Bruna. On the equivalence between graph isomorphism testing and function approximation with gnns. In *Advances in Neural Information Processing Systems*, pages 15868–15876, 2019.

[25] Zhengdao Chen, Lei Chen, Soledad Villar, and Joan Bruna. Can graph neural networks count substructures? *arXiv preprint arXiv:2002.04025*, 2020.

[26] Christopher Morris, Yaron Lipman, Haggai Maron, Bastian Rieck, Nils M. Kriege, Martin Grohe, Matthias Fey, and Karsten Borgwardt. Weisfeiler and leman go machine learning: The story so far, 2021. URL `https://arxiv.org/abs/2112.09992`.

[27] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.

[28] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in neural information processing systems*, 29, 2016.

[29] Michael Perlmutter, Feng Gao, Guy Wolf, and Matthew Hirn. Understanding graph neural networks with asymmetric geometric scattering transforms. *arXiv preprint arXiv:1911.06253*, 2019.

[30] Renjie Liao, Zhizhen Zhao, Raquel Urtasun, and Richard S Zemel. Lanczosnet: Multi-scale deep graph convolutional networks. 2019.

[31] Luana Ruiz, Fernando Gama, and Alejandro Ribeiro. Graph neural networks: Architectures, stability, and transferability. *Proceedings of the IEEE*, 109(5):660–682, 2021. doi: 10.1109/JPROC.2021.3055400.

[32] Lingxiao Zhao and Leman Akoglu. Pairnorm: Tackling oversmoothing in gnns. *CoRR*, abs/1909.12223, 2019.

[33] Chaoqi Yang, Ruijie Wang, Shuochao Yao, Shengzhong Liu, and Tarek Abdelzaher. Revisiting over-smoothing in deep gcns, 2020.

[34] Kaixiong Zhou, Xiao Huang, Yuening Li, Daochen Zha, Rui Chen, and Xia Hu. Towards deeper graph neural networks with differentiable group normalization. *Advances in Neural Information Processing Systems*, 33:4917–4928, 2020.

[35] Yihao Chen, Xin Tang, Xianbiao Qi, Chun-Guang Li, and Rong Xiao. Learning graph normalization for graph neural networks. *Neurocomputing*, 2022.

[36] Vijay Prakash Dwivedi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Graph neural networks with learnable structural and positional representations. 2021. doi: 10.48550/ARXIV.2110.07875. URL `https://arxiv.org/abs/2110.07875`.

[37] Jiaxuan You, Rex Ying, and Jure Leskovec. Position-aware graph neural networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 7134–7143. PMLR, 09–15 Jun 2019. URL `https://proceedings.mlr.press/v97/you19b.html`.

[38] Frederik Wenkel, Yimeng Min, Matthew Hirn, Michael Perlmutter, and Guy Wolf. Overcoming oversmoothness in graph convolutional networks via hybrid scattering networks. *arXiv preprint arXiv:2201.08932*, 2022.

[39] Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 1725–1735. PMLR, 13–18 Jul 2020. URL `http://proceedings.mlr.press/v119/chen20v.html`.

[40] Ziqi Liu, Chaochao Chen, Longfei Li, Jun Zhou, Xiaolong Li, and Le Song. Geniepath: Graph neural networks with adaptive receptive paths. *CoRR*, abs/1802.00910, 2018.

[41] Meng Liu, Zhengyang Wang, and Shuiwang Ji. Non-local graph neural networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[42] Zhiwen Xie, Guangyou Zhou, Jin Liu, and Xiangji Huang. Reinceptione: relation-aware inception network with joint local-global structural information for knowledge graph embedding. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 5929–5939, 2020.

[43] Zhanghao Wu, Paras Jain, Matthew Wright, Azalia Mirhoseini, Joseph E Gonzalez, and Ion Stoica. Representing long-range context for graph neural networks with global attention. *Advances in Neural Information Processing Systems*, 34:13266–13279, 2021.

[44] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. Dropedge: Towards deep graph convolutional networks on node classification, 2020.

[45] Jiayu Li, Tianyun Zhang, Hao Tian, Shengmin Jin, Makan Fardad, and Reza Zafarani. Sgcn: A graph sparsifier based on graph convolutional networks. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 275–287. Springer, 2020.

[46] Tianlong Chen, Yongduo Sui, Xuxi Chen, Aston Zhang, and Zhangyang Wang. A unified lottery ticket hypothesis for graph neural networks. In *International Conference on Machine Learning*, pages 1695–1706. PMLR, 2021.

[47] Hanqing Zeng, Muhan Zhang, Yinglong Xia, Ajitesh Srivastava, Andrey Malevich, Rajgopal Kannan, Viktor Prasanna, Long Jin, and Ren Chen. Deep graph neural networks with shallow subgraph samplers, 2021.

[48] Minji Yoon, Théophile Gervet, Baoxu Shi, Sufeng Niu, Qi He, and Jaewon Yang. Performance-adaptive sampling strategy towards fast and accurate graph neural networks. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 2046–2056, 2021.

[49] Hanqing Zeng, Muhan Zhang, Yinglong Xia, Ajitesh Srivastava, Andrey Malevich, Rajgopal Kannan, Viktor Prasanna, Long Jin, and Ren Chen. Decoupling the depth and scope of graph neural networks. *Advances in Neural Information Processing Systems*, 34, 2021.

[50] Fernando Gama, Alejandro Ribeiro, and Joan Bruna. Diffusion scattering transforms on graphs. *arXiv preprint arXiv:1806.08829*, 2018.

[51] Seyedeh Fatemeh Mousavi, Mehran Safayani, Abdolreza Mirzaei, and Hoda Bahonar. Hierarchical graph embedding in vector space by graph pyramid. *Pattern Recognition*, 61:245–254, 2017.

[52] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. *Advances in neural information processing systems*, 31, 2018.

[53] Zhengdao Chen, Xiang Li, and Joan Bruna. Supervised community detection with line graph neural networks. *arXiv preprint arXiv:1705.08415*, 2017.

[54] Di Jin, Ziyang Liu, Weihao Li, Dongxiao He, and Weixiong Zhang. Graph convolutional networks meet markov random fields: Semi-supervised community detection in attribute networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 152–159, 2019.

[55] Filippo Maria Bianchi, Daniele Grattarola, and Cesare Alippi. Spectral clustering with graph neural networks for graph pooling. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 874–883. PMLR, 13–18 Jul 2020. URL `https://proceedings.mlr.press/v119/bianchi20a.html`.

[56] Vince Lyzinski, Daniel L Sussman, Minh Tang, Avanti Athreya, and Carey E Priebe. Perfect clustering for stochastic blockmodel graphs via adjacency spectral embedding. *Electronic journal of statistics*, 8(2):2905–2922, 2014.

[57] Ulrike Von Luxburg, Mikhail Belkin, and Olivier Bousquet. Consistency of spectral clustering. *The Annals of Statistics*, pages 555–586, 2008.

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

[59] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In *International Conference on Machine Learning*, pages 6861–6871, 2019.

[60] Chen Cai and Yusu Wang. A note on over-smoothing for graph neural networks, 2020.

[61] Yingbo Hua, Yong Xiang, Tianping Chen, Karim Abed-Meraim, and Yongfeng Miao. A new look at the power method for fast subspace tracking. *Digital Signal Processing*, 9(4):297–314, 1999.

[62] Derek Lim, Joshua Robinson, Lingxiao Zhao, Tess Smidt, Suvrit Sra, Haggai Maron, and Stefanie Jegelka. Sign and basis invariant networks for spectral graph representation learning. *arXiv preprint arXiv:2202.13013*, 2022.

[63] Haggai Maron, Heli Ben-Hamu, Nadav Shamir, and Yaron Lipman. Invariant and equivariant graph networks. *arXiv preprint arXiv:1812.09902*, 2018.

[64] Fabrizio Frasca, Emanuele Rossi, Davide Eynard, Ben Chamberlain, Michael Bronstein, and Federico Monti. Sign: Scalable inception graph neural networks, 2020. URL `https://arxiv.org/abs/2004.11198`.

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

[66] Eli Chien, Jianhao Peng, Pan Li, and Olgica Milenkovic. Adaptive universal generalized pagerank graph neural network. *arXiv preprint arXiv:2006.07988*, 2020.

[67] Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks. In *International Conference on Machine Learning*, pages 1725–1735. PMLR, 2020.

[68] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[69] Yi Luo, Guangchun Luo, Ke Yan, and Aiguo Chen. Inferring from references with differences for semi-supervised node classification on graphs. *Mathematics*, 10(8):1262, 2022.

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

[71] Johannes Gasteiger, Stefan Weißenberger, and Stephan Günnemann. Diffusion improves graph learning. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2019.

[72] T Tony Cai, Tengyuan Liang, and Alexander Rakhlin. Inference via message passing on partially labeled stochastic block models. *arXiv preprint arXiv:1603.06923*, 2016.

[73] Cencheng Shen, Qizhe Wang, and Carey E. Priebe. One-hot graph encoder embedding, 2021. URL `https://arxiv.org/abs/2109.13098`.

[74] Martin Fürer. On the power of combinatorial and spectral invariants. *Linear algebra and its applications*, 432(9):2373–2380, 2010.

[75] Gaurav Rattan and Tim Seppelt. Weisfeiler–leman and graph spectra, 2021. URL `https://arxiv.org/abs/2103.02972`.

[76] Ulrike Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416, 2007.

[77] Benedek Rozemberczki, Carl Allen, and Rik Sarkar. Multi-scale attributed node embedding. *Journal of Complex Networks*, 9(2):cnab014, 2021.

[78] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*, 2018.

[79] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.

# A Proofs of `PowerEmbed` properties

*Proof of Proposition 4.1.* Recall $S \in \mathbb{R}^{n\times n}$ and $U(t) \in \mathbb{R}^{n\times k}$ for all $t \geq 0$. Since $S \in \{A, \bar{A}, A_{rw}\}$, $S$ admits a spectral decomposition. This is obvious for the symmetric matrices $A, \bar{A}$. For non-symmetric $A_{rw}$, if $\lambda$ is an eigenvalue $\bar{A}$ with eigenvector $w$, then $\lambda$ is also an eigenvalue of $A_{rw}$ with eigenvector $\tilde{D}^{-0.5}w$ [76, Proposition.3.3].

Assume the initialization $U_0$ is full rank, and the $k$-th and $(k+1)$-th eigenvalue of $S$ are distinct. Following the argument from Appendix A in [61] we observe

$$U(t+1) = SU(t)\left(U(t)^\top S^2 U(t)\right)^{-1} \tag{9}$$

where $U(t) \in \mathbb{R}^{n\times k}$. Consider the spectral decomposition of $S$:

$$S = [V_1\ V_2]\begin{bmatrix}\Lambda_1 & 0\\ 0 & \Lambda_2\end{bmatrix}[V_1\ V_2]^\top, \tag{10}$$

where $V_1 \in \mathbb{R}^{n\times k}$, $V_2 \in \mathbb{R}^{n\times(n-k)}$, and $\Lambda_1 = \mathrm{diag}(\lambda_1, \ldots, \lambda_k)$, $\Lambda_2 = \mathrm{diag}(\lambda_{k+1}, \ldots, \lambda_n)$ and

Let $\Phi(t) = V_1^\top U(t) \in \mathbb{R}^{k\times k}$ and $\Omega(t) = V_2^\top U(t) \in \mathbb{R}^{(n-k)\times k}$ for $t \geq 0$. Multiplying $[V_1\ V_2]^\top$ on both sides of (9) and using (10), we have

$$\begin{bmatrix}\Phi(t+1)\\ \Omega(t+1)\end{bmatrix} = \begin{bmatrix}\Lambda_1 & 0\\ 0 & \Lambda_2\end{bmatrix}\begin{bmatrix}\Phi(t)\\ \Omega(t)\end{bmatrix} Z(t), \tag{11}$$

where $Z(t) = \{\Phi(t)^\top\Lambda_1^2\Phi(t) + \Omega(t)^\top\Lambda_2^2\Omega(t)\}^{-1}$.

Since $\Phi(0) = V_1^\top U(0)$ is full rank, there exists $L$ such that $\Omega(0) = L\Phi(0)$. A simple induction argument shows that

$$\Omega(t) = \Lambda_2^t L \Lambda_1^{-t}\Phi(t). \tag{12}$$

Using the assumption that $\lambda_k > \lambda_{k+1}$, (12) implies that $\lim_{t\to\infty}\Omega(t) \to 0$. Therefore $\lim_{t\to\infty} Z(t) = \{\Phi(t)^\top\Lambda_1^2\Phi(t)\}^{-1}$. Then (11) implies

$$\Phi(t+1) = \Lambda_1\Phi(t)Z(t) = \Lambda_1^{-1}\left(\Phi(t)^\top\right)^{-1}. \tag{13}$$

Thus, upon convergence at $t = L$, $\Phi(L)\Phi(L)^\top = \Lambda_1$ which implies $U(L) = WV_1\Lambda_1^{-0.5}$ where $W$ is a unitary matrix. This proves that Algorithm 1 returns the top-$k$ eigenvectors of $S$ up to orthogonal transformation in $O(k)$. □

*Proof of Proposition 4.2.* Algorithm 1 is permutation equivariant. Namely, for all $\Pi \in S_n$ permutation matrix and all $A \in \mathbb{R}^{n\times n}$ and all $X \in \mathbb{R}^{n\times k}$ we have that

$$\texttt{PowerEmbed}(\Pi\, A\, \Pi^\top, \Pi\, X) = \Pi\, \texttt{PowerEmbed}(A, X). \tag{14}$$

It suffices to observe that $\Pi A \Pi^\top \Pi X = \Pi A\, X$ and $(\Pi A\, X)^\top(\Pi A\, X) = (A\, X)^\top(A\, X)$, making $\tilde{U}(t{+}1)^\top\tilde{U}(t{+}1)$ permutation invariant, and therefore $[\mathcal{T}_u(\tilde{U}(t{+}1)^\top\tilde{U}(t{+}1))]^{-1}$ is also permutation invariant. Therefore at each step $\tilde{U}(t{+}1)(\Pi\, A\, \Pi^\top, \Pi\, X) = \Pi\tilde{U}(t{+}1)(A, X)$, making `PowerEmbed` permutation equivariant.

The inception networks $g_{\theta'} \circ \texttt{MLPs}$ are permutation equivariant because the same function is applied to all the nodes. It implements the equivariance as weight sharing (similar to message-passing methods). □

# B Experiment details

## B.1 Baselines

We perform `PowerEmbed` with different graph operators: $A$ (indicated by "Power"), $A_{rw}$ (indicated by "Power(RW)") and $\bar{A}$ (indicated by "Power(Lap)").

For semi-supervised MPNNs including GCNII and GPR-GNN, we use hyperparameters that are are comparable with `PowerEmbed`: epochs $= 100$, learning rate $= 0.01$, dropout rate $= 0.5$, weight decay $= 0$, number of layers $= 10$. Specifically for GCNII, we use $\alpha = 0.1$.

### B.2 Datasets

To evaluate PowerEmbed, we consider 5 dense graphs and 5 sparse graphs with different sizes and homophily ratios (see Tables 1, 2 for summary statistics).

*Webpage networks.* In these networks, nodes represent webpages and edges are hyperlinks between them. Squirrel and Chameleon are created based on specific topics from Wikipedia [77], where node features are informative nouns in the Wikipedia pages and the node labels are created in [8] which reflect the average monthly traffic of the webpage. Wisconsin, Texas and Cornell are introduced in [8], where node features are bag-of-words representation of the webpages, and node labels are the webpage category. All webpage networks we considered exhibit heterophily structure, where node with different class labels tend to be connected.

*Co-purchase networks.* Computers and Photo are Amazon co-purchase networks introduced in [78], where nodes represent goods, edges represent tendency of being bought together, node features are bag-of-words vector of product reviews, and node labels are product category.

*Citation networks.* Cora and Citeseer are standard citation network benchmarks [79] where nodes represent documents, edges represent citation links, node features are bag-of-words vector from the dictionary, and node labels are the document category.

*Co-authorship network.* Coauthor(CS) is introduced in [78] where nodes represent researchers, edges represent coauthorship, node features are bag-of-words vector from the paper keyword dictionary, and node labels are the research fields.

## C Ablation study

### C.1 Synthetic graphs

We report comprehensive simulation results on 2B-SBM models. Graph Laplacian operator $\bar{A}$ is used in all message-passing methods including PowerEmbed, SGC, GCN, GCNII, GPR-GNN. All experiments are done using $n = 500$ nodes and repeated for 30 random runs. Figure 5 supplements Figure 3 with other baselines, and Figure 6 shows a more fine-grained picture as the graph density decreases. We make the following observations.

*Performance in sparse graphs.* In sparse graphs (e.g., Fig 5 (b), Fig 6), spectral methods and the last iterate of PowerEmbed perform poorly whereas PowerEmbed using all intermediate representations (including initial node features) performs well. Shallow MPNNs achieve strong performance (e.g., 'SGC-2", "GCN-2"), including heterophilous graphs (Fig 5 (B)): as analyzed in [10], when nodes from the same class exhibit similar associativity pattern with other classes (as in the case of SBM graphs), heterophilous graphs are not necessarily hard to learn for shallows MPNNs.

*Over-smoothing.* Fig 6 shows that Standard MPNNs (e.g., 'SGC-10", "GCN-10") suffer from over-smoothing: increasing the number of layers degrades the performance. Moreover, the denser the graph, the worse of the performance degradation (e.g., 'SGC-5", "GCN-5' perform better in sparse graphs than dense graphs), a phenomenon discussed in [6, 44]. On the other hand, the last iterate of PowerEmbed (right-most columns in Fig 5) prevents over-smoothing in dense graphs. MPNNs that are specifically designed to avoid over-smoothing (e.g., GCNII, GPR-GNN) perform more stable as the number of layers increases, but suffer from high variance issues. PowerEmbed also prevents over-smoothing while achieving smaller variance. This illustrates the benefits of expressing global *spectral* information over long-range spatial information under the SBM models, which we also empirically observe in some real-world graphs (see Table 1, Table 2).

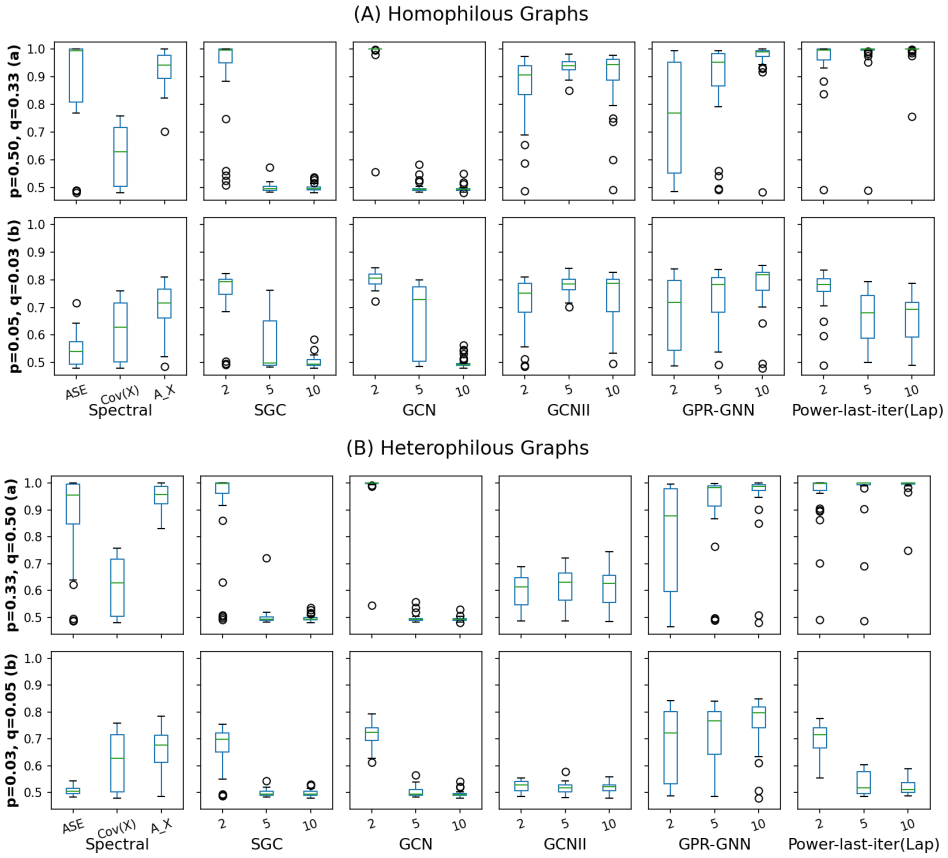


Figure 5: Supplementary to Figure 3 on other baselines: the last iterate of `PowerEmbed` perform well in dense graphs (both homophily and heterophily), but fail in sparse graphs, similar to spectral embedding ("ASE"). Standard MPNNs suffer from over-smoothing ("SGC-10", "GCN-10"). MPNNs encode long-range information can go deeper ("GCNII", "GPR-GNN"), albeit with higher variance.

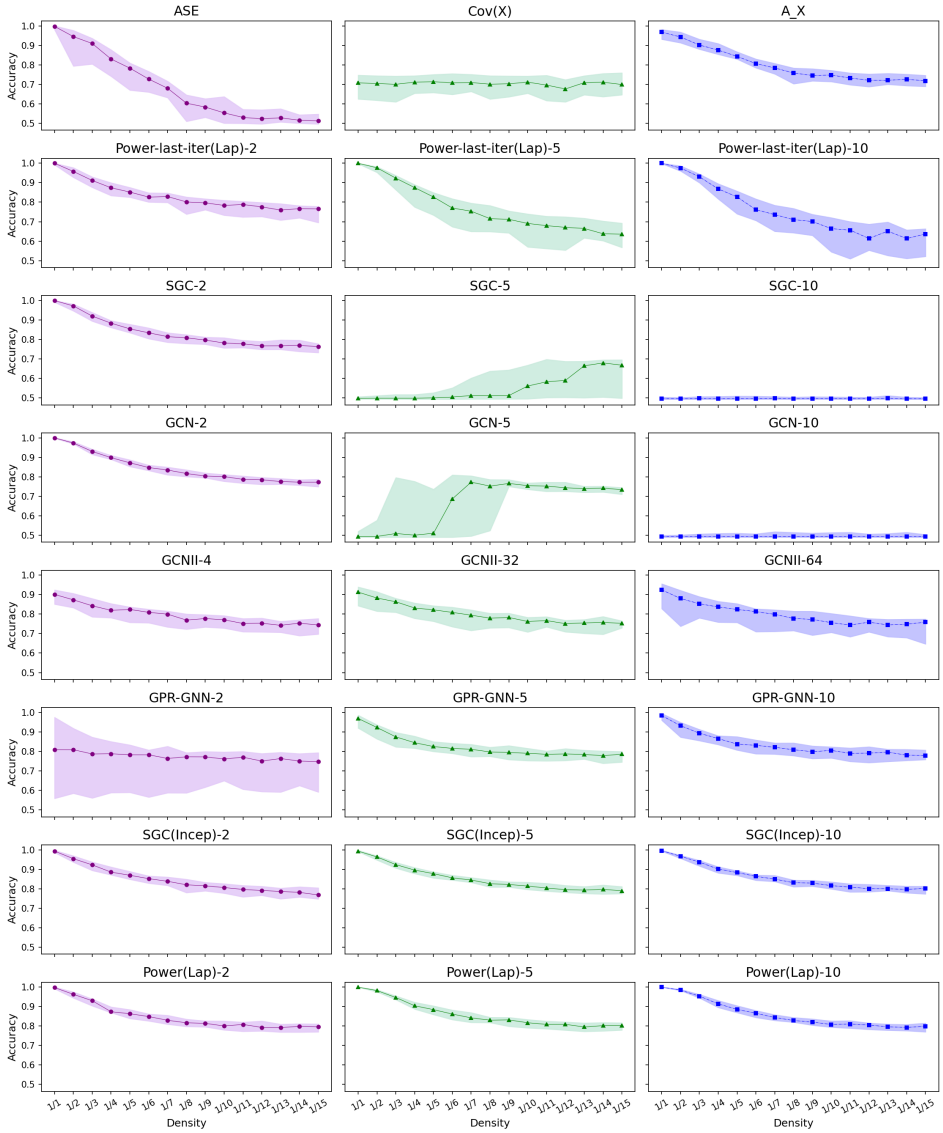


Figure 6: Performance w.r.t graph density changes in 2B-SBM model: $p = 1/2 \times$ X-axis, $q = 1/3 \times$ X-axis where $p$,$q$ defines the block connection probability matrix $B$ in eqn (3). "Method-$k$" denotes the corresponding method with $k$ number of layers. As density decreases from left to right, the performance of spectral embedding "ASE" and the last iterate of `PowerEmbed` (second row) degrade significantly, while shallow MPNNs degrade more gracefully (e.g.,"SGC-2", "GCN-2"). On the other hand, deep MPNNs (e.g.,"SGC-10", "GCN-10") completely fail due to oversmoothing, while deep (last iterate of) `PowerEmbed` are more resilient. Finally, `PowerEmbed` and SGC(Incep) (bottom two rows) that use a list of intermediate embeddings perform consistently well in sparse graphs, robust to the choice of number of layers.

## C.2 Real-world graphs

We compare the performance of `PowerEmbed` on real benchmarks using (1) all iterations; (2) last iteration; (3) the input feature and the last iteration. Figure 7 shows that on dense graphs, using only the input feature and the last iteration (3), `PowerEmbed` substantially outperforms its message-passing counterparts without normalization step.

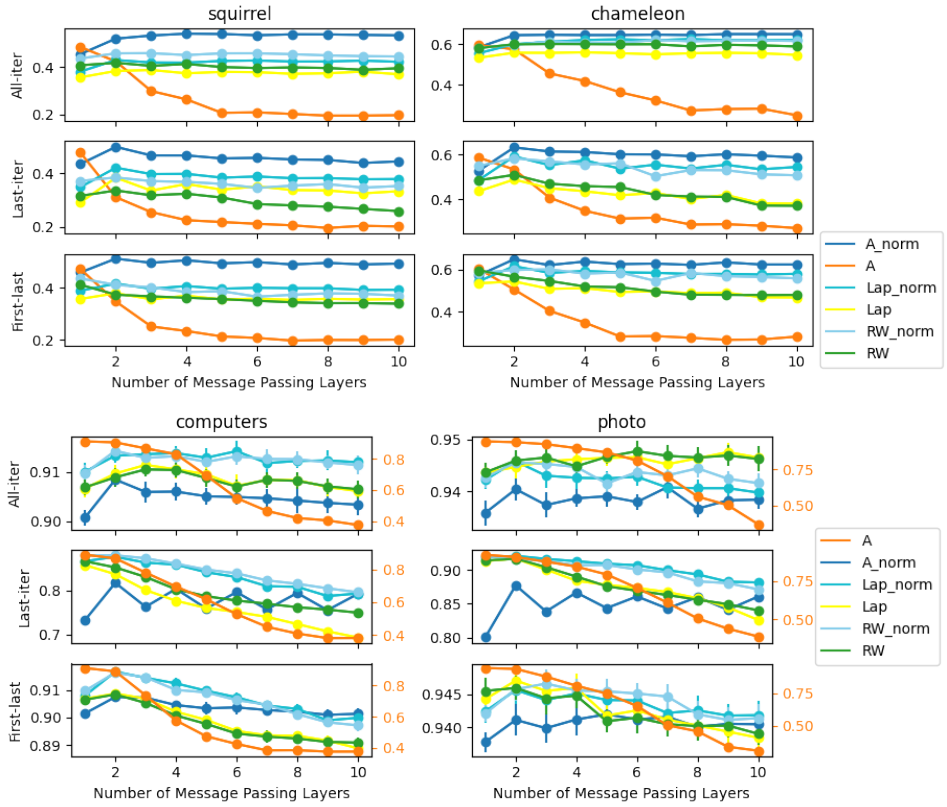


Figure 7: In dense graphs, `PowerEmbed` (blue lines, annotated with "_norm") outperforms its message-passing counterparts; We compare different graph operators $A$, $\bar{A}$ ("Lap"), and $A_{rw}$ ("RW"). We also study the effects of the number of iterations. Bottom row: The secondary Y-axis with orange labels correspond to performance of $A$.