From Local to Global: Spectral-Inspired Graph Neural Networks

Summer Research, AMS, JHU

Chengyue Huang

September 29, 2022

arXiv preprint(2022) arXiv:2209.12054
Submitted to NeurIPS 2022 GLFrontiers Workshop.

Outline

Notations & Concepts

- 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 $Y \in \mathbb{R}^{n}$.

Notations & Concepts

- 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 $Y \in \mathbb{R}^{n}$.
- A, D are the adjacency matrix and the degree matrix of G.

Notations & Concepts

- 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 $Y \in \mathbb{R}^{n}$.
- A, D are the adjacency matrix and the degree matrix of G.
- Self-loop: $\tilde{A} = A + I, \tilde{D} = D + I$.

Notations & Concepts

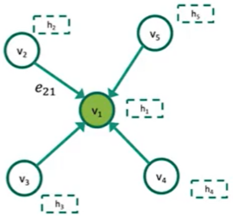
- Graph $G = (V, E, X, Y)$ with node set V, edge set E, node input features $X \in \mathbb{R}^{n \times p}$ and node labels $Y \in \mathbb{R}^{n}$.
- A, D are the adjacency matrix and the degree matrix of G.
- Self-loop: $\tilde{A} = A + I, \tilde{D} = D + I$.
- **Graph Laplacians**
  - Random Walk Graph Laplacian: $A_{rw} = \tilde{D}^{-1}\tilde{A}$
  - Symmetric Graph Laplacian: $\bar{A} = \tilde{D}^{-0.5}\tilde{A}\tilde{D}^{-0.5}$

Notations & Concepts

- 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 $Y \in \mathbb{R}^{n}$.
- A, D are the adjacency matrix and the degree matrix of G.
- Self-loop: $\tilde{A} = A + I, \tilde{D} = D + I$.
- **Graph Laplacians**
  - Random Walk Graph Laplacian: $A_{rw} = \tilde{D}^{-1}\tilde{A}$
  - Symmetric Graph Laplacian: $\bar{A} = \tilde{D}^{-0.5}\tilde{A}\tilde{D}^{-0.5}$
- **Graph Topology**
  - Homophily: nodes from same classes are more likely to connect.
  - Heterophily: nodes from different classes are more likely to connect.

Preliminaries

- **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

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

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

Preliminaries

- **Message-Passing Neural Network (MPNN)**

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

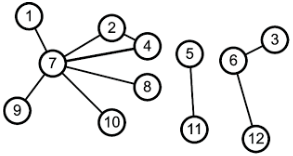
  - **Graph Convolution Network (GCN): $\mathbf{h}^{(l)} = \sigma(\bar{A}\mathbf{h}^{(l-1)}W^{(l-1)})$.**
  - **Simple Graph Convolution (SGC)**:

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

  where σ is nonlinear activaton function, and $\psi(i,j) = \frac{1}{\sqrt{\deg(i)\deg(j)}} W^{(l-1)}\mathbf{h}_j^{(l-1)}$.

Over-smoothing Problem

- **SGC:** $\mathbf{h}^{(L)} = \bar{A}\mathbf{h}^{(L-1)}W^{(L-1)} = \bar{A}^L X\big(W^{(L-1)} \dots W^{(0)}\big) = \bar{A}^L XW.$
- **Over-smoothing**



Assume the graph has one connected component. Then

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

- Only encodes connected component and degree information;
- Miss the community structure present in subsequent eigenvectors.

Over-squashing Problem

- **Over-squashing**
  Let $h_i^{(L)} = h_i^{(L)}(x_1, \ldots, x_n)$ 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 $\partial h_i^{(L)} / \partial x_s$.
  - The smaller the Jacobian value, the more node feature information is "squashed" out in the embedding.

Related Work

- **Borrow optimization techniques from training standard neural networks**
  Adapt normalization techniques from deep learning and propose node-wise, batch-wise, and graph-wise normalization methods.

Related Work

- **Borrow optimization techniques from training standard neural networks**
  Adapt normalization techniques from deep learning and propose node-wise, batch-wise, and graph-wise normalization methods.
- **Inject global information in MPNNs**
  - Encode global properties of the graph as inputs to MPNNs: using spectral embeddings as node features, sampling anchor nodes, or using other low-pass geometric features.
  - Use specific architectural choices: residual connections, attention mechanisms, or transformers.

Related Work

- **Borrow optimization techniques from training standard neural networks**
  Adapt normalization techniques from deep learning and propose node-wise, batch-wise, and graph-wise normalization methods.
- **Inject global information in MPNNs**
  - Encode global properties of the graph as inputs to MPNNs: using spectral embeddings as node features, sampling anchor nodes, or using other low-pass geometric features.
  - Use specific architectural choices: residual connections, attention mechanisms, or transformers.
- **Modify the graph structures with increasingly complicated architectures**
  To speed up the long-range information flow: graph sparsification, graph sampling, or localized subgraph extraction.

PowerEmbed

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 $P$ [Column normalization]
  end for
  return $P$
```

*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 $O(k)$).*

PowerEmbed

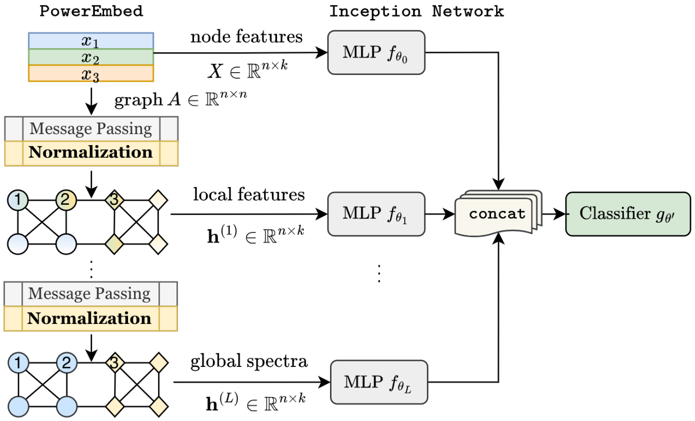


Figure 1: `PowerEmbed` extracts both *local* features from the first few iterations and *global* information from the last few iterations (i.e., the top-k eigenvectors), which are jointly learned using the inception network.

Numerical Experiments - Baselines

- **Unnormalized counterparts**: using intermediate representations
  - SIGN (scalable inception GNN): A_{rw}
  - SGC(Incep): $\bar{A}$

Numerical Experiments - Baselines

- **Unnormalized counterparts**: using intermediate representations
  - SIGN (scalable inception GNN): A_{rw}
  - SGC(Incep): $\bar{A}$
- **Spectral methods**
  Spectral decomposition of the graph A and the covariance matrix $XX^\top$:
  $A = USU^\top, XX^\top = \tilde{U}\Sigma\tilde{U}^\top$
  - ASE: $\mathbf{h}^{ASE} = U_k$
  - Cov(X): $\mathbf{h}^{\text{cov}(X)} = \tilde{U}_k$
  - A_X: $\mathbf{h}^{A\_X} = [\mathbf{h}^{ASE}; \mathbf{h}^{\text{cov}(X)}]$

Numerical Experiments - Baselines

- **Unnormalized counterparts**: using intermediate representations
  - SIGN (scalable inception GNN): A_{rw}
  - SGC(Incep): $\bar{A}$
- **Spectral methods**
  Spectral decomposition of the graph A and the covariance matrix $XX^\top$:
  $A = USU^\top, XX^\top = \tilde{U}\Sigma\tilde{U}^\top$
  - ASE: $\mathbf{h}^{ASE} = U_k$
  - Cov(X): $\mathbf{h}^{\mathrm{cov}(X)} = \tilde{U}_k$
  - A_X: $\mathbf{h}^{A\_X} = [\mathbf{h}^{ASE}; \mathbf{h}^{\mathrm{cov}(X)}]$
- **Semi-supervised MPNNs**
  - Benchmark: GCN, GAT (graph attention network)
  - Long-range spatial information: GEOM-GCN (geometric GCN), GPR-GNN (generalized page-rank GNN), and GCNII (GCN via initial residual and identity mapping)

Numerical Experiments - Synthetic Graphs

- **Stochastic Block Model (SBM)**
  A graph A with n nodes is a random SBM graph if it is sampled as

$$A \sim Bernoulli(P),\ P = ZBZ^\top,$$

  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.

Numerical Experiments - Synthetic Graphs

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

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

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

Numerical Experiments - Synthetic Graphs

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

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

  where $p, q \in (0,1), p \neq q$. The node features in block $k \in \{0,1\}$ are sampled from a m-dimensional multivariate Gaussian $\mathcal{N}(\mu_k, \Sigma_k)$ and stored in a node feature matrix $X \in \mathbb{R}^{n \times m}$.
  - **2B-SBM model without node features**
    1. $P = \mathbb{E}(A)$ has three eigenvalues, ranked by magnitude as $(p+q)/2, (p-q)/2, 0$, where 0 has multiplicity $n-2$.
    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**.

Numerical Experiments - Synthetic Graphs

- **Convergence of estimated eigenvectors**
  Settings: 2 Gaussians with mean $\mu_0 = [1, 1]$, $\mu_1 = -\mu_0$, binary node classification, $n = 500$ nodes, 10/90 train/test split, $k = 2$.

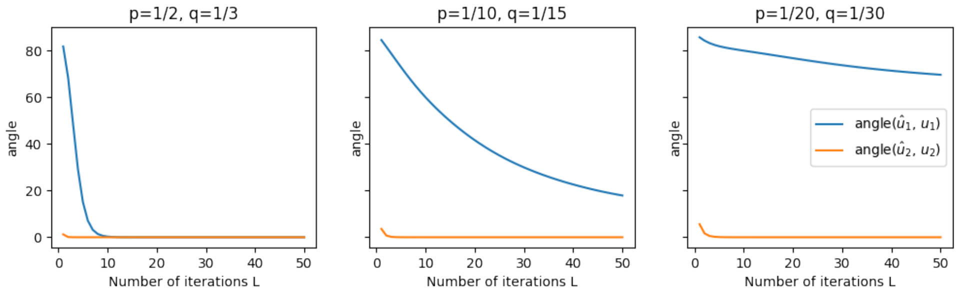


Figure 2: Convergence of the estimated eigenvectors $\hat{u}_i$ from Algorithm 1 is faster for denser graphs and slower for sparse graphs. We report the angles between the true top-2 eigenspaces and estimated top-2 eigenspaces, averaged over 30 random runs.

Numerical Experiments - Synthetic Graphs

- **Comparison with baselines**

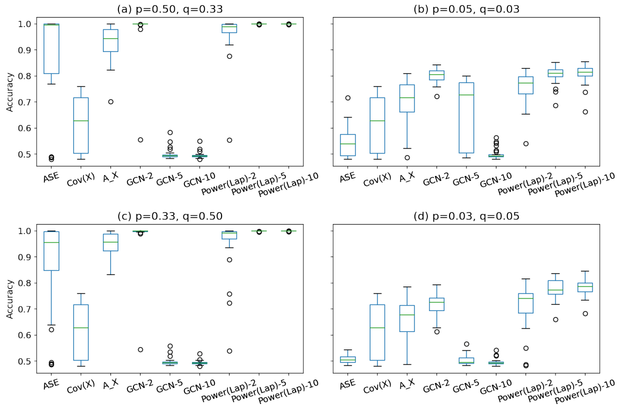


Figure 3: `PowerEmbed` enjoys the same performance guarantee as spectral embedding methods in dense graphs (left), and outperforms spectral methods in sparse graphs (right). Standard MPNNs suffer from over-smoothing (e.g., "GCN-10") and perform much worse in sparse heterophilous graphs (e.g., "GCN-5").

Numerical Experiments - Synthetic Graphs

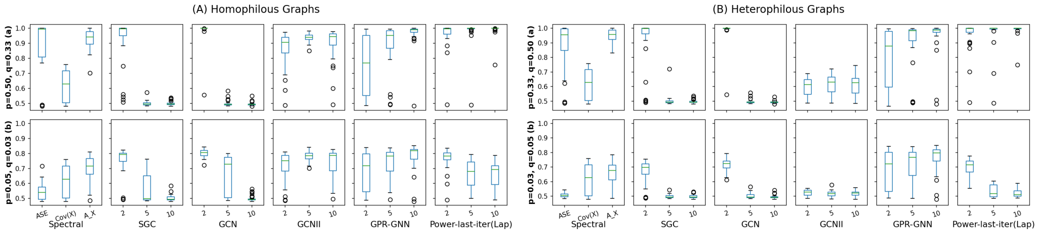


Figure 4: Supplementary to Figure 3 on other baselines: the last iterate of `PowerEmbed` perform well in dense graphs (both homophily and heterophily), but fail in sparse graphs, similar to spectral embedding ("ASE"). Standard MPNNs suffer from over-smoothing ("SGC-10", "GCN-10"). MPNNs encode long-range information can go deeper ("GCNII", "GPR-GNN"), albeit with higher variance.

Numerical Experiments - Synthetic Graphs

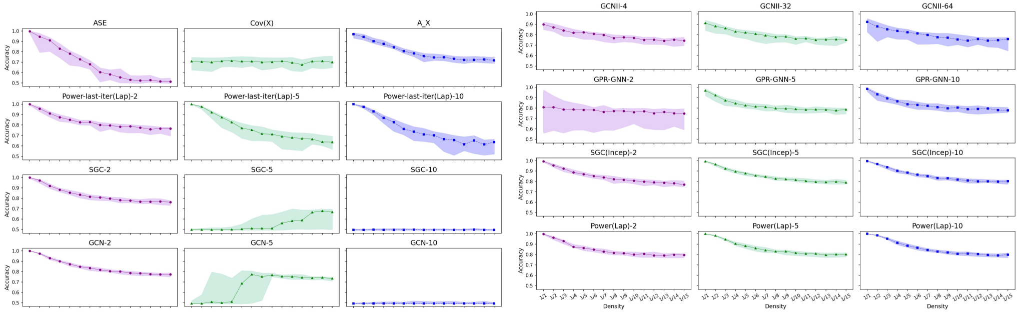


Figure 5: Performance w.r.t graph density changes in 2B-SBM model: $p = 1/2 \times$ X-axis, $q = 1/3 \times$ X-axis.

- As density decreases, the performance of "ASE" and Power-last-iter (second row) degrade significantly, while shallow MPNNs degrade more gracefully.
- Deep MPNNs completely fail, while deep Power-last-iter are more resilient.
- `PowerEmbed` and SGC(Incep) that use a list of intermediate embeddings perform consistently well in sparse graphs, robust to the choice of number of layers.

Numerical Experiments - Real-world Graphs - Heterophily

Table 1: `PowerEmbed` outperforms other baselines on graphs with heterophily, particularly on dense heterophilous graphs.

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 ± 0.41**	**64.98 ± 0.55**	74.71 ± 1.74	73.51 ± 2.05	75.14 ± 2.50
Power(RW)-10	44.58 ± 0.52	61.64 ± 0.43	75.49 ± 1.71	75.68 ± 1.21	72.97 ± 1.58
Power(Lap)-10	42.32 ± 0.37	62.17 ± 0.41	74.71 ± 1.74	74.05 ± 2.10	77.03 ± 1.54
Power-2	52.13 ± 0.55	64.47 ± 0.76	75.29 ± 1.47	**79.19 ± 1.33**	76.76 ± 1.63
Power(RW)-2	45.92 ± 0.48	59.67 ± 0.62	77.45 ± 0.89	76.22 ± 1.31	75.41 ± 1.85
Power(Lap)-2	43.06 ± 0.56	60.00 ± 0.62	**78.43 ± 1.59**	77.03 ± 1.54	**78.30 ± 1.58**
SGC(Incep)-10	37.07 ± 0.55	55.11 ± 0.82	75.29 ± 1.04	75.68 ± 1.95	75.68 ± 1.83
SIGN-10	38.47 ± 0.42	60.22 ± 0.72	75.29 ± 1.45	73.51 ± 2.02	75.68 ± 1.21
SGC(Incep)-2	35.33 ± 0.35	54.19 ± 0.65	77.45 ± 0.89	76.76 ± 1.34	76.22 ± 2.07
SIGN-2	40.97 ± 0.35	60.11 ± 0.97	**78.43 ± 1.41**	75.14 ± 2.02	76.76 ± 1.34
Cov(X)	33.12 ± 0.53	44.74 ± 1.00	75.69 ± 1.25	77.30 ± 1.12	77.03 ± 2.27
ASE	41.46 ± 0.62	57.92 ± 0.77	49.41 ± 2.09	58.65 ± 1.79	56.76 ± 0.66
A_X	49.11 ± 0.37	61.97 ± 0.76	77.84 ± 1.24	76.76 ± 1.22	75.95 ± 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 ± 0.50	49.96 ± 0.46	59.02 ± 1.60	61.08 ± 1.49	48.38 ± 1.64
GPR-GNN-10	34.51 ± 1.45	52.37 ± 3.43	59.41 ± 2.75	58.92 ± 2.98	52.97 ± 3.11

Numerical Experiments - Real-world Graphs - Heterophily

- **Increasing the number of message-passing layers**

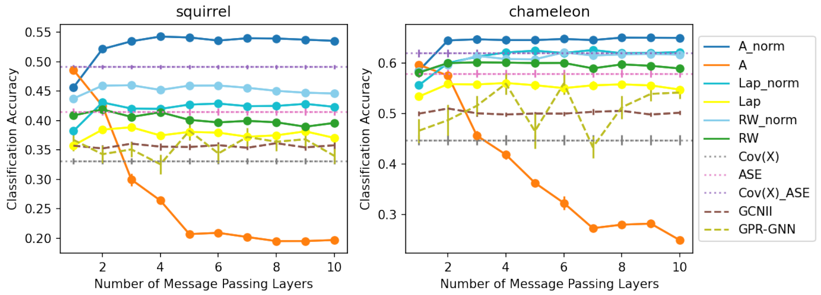


Figure 6: `PowerEmbed` (annotated with "_norm") that adds the normalization step for orthogonality can expressive top-k eigenvectors, which avoids over-smoothing and outperforms other baselines, particularly in heterophilous graphs. Baselines include unnormalized counterparts (SIGN denoted as "RW", SGC(Incep) denoted as "Lap"); spectral methods, and semi-supervised MPNNs.

Numerical Experiments - Real-world Graphs - Homophily

Table 2: `PowerEmbed` achieves competitive performance as other MPNN baselines.

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 ± 0.22	93.84 ± 0.17	93.93 ± 0.11	81.69 ± 0.50	69.67 ± 0.63
Power(RW)-10	91.14 ± 0.19	94.16 ± 0.20	93.88 ± 0.13	85.03 ± 0.44	73.15 ± 0.54
Power(Lap)-10	91.20 ± 0.14	93.97 ± 0.19	94.26 ± 0.09	84.95 ± 0.40	72.61 ± 0.51
Power-2	90.85 ± 0.15	94.04 ± 0.21	94.32 ± 0.11	81.23 ± 0.52	72.03 ± 0.41
Power(RW)-2	$\mathbf{91.43 \pm 0.13}$	94.56 ± 0.19	94.30 ± 0.08	83.56 ± 0.44	72.62 ± 0.48
Power(Lap)-2	91.33 ± 0.15	94.58 ± 0.21	94.75 ± 0.09	83.52 ± 0.27	73.27 ± 0.75
SGC(Incep)-10	90.61 ± 0.16	94.65 ± 0.18	94.44 ± 0.10	84.89 ± 0.71	73.39 ± 0.62
SIGN-10	90.65 ± 0.18	94.63 ± 0.25	94.05 ± 0.13	85.45 ± 0.32	72.78 ± 0.51
SGC(Incep)-2	90.97 ± 0.17	94.47 ± 0.22	94.54 ± 0.08	83.74 ± 0.53	72.47 ± 0.61
SIGN-2	90.89 ± 0.20	$\mathbf{94.59 \pm 0.19}$	94.09 ± 0.12	83.92 ± 0.43	73.27 ± 0.53
Cov(X)	82.42 ± 0.14	89.45 ± 0.26	91.70 ± 0.15	69.24 ± 0.56	66.79 ± 0.62
ASE	77.61 ± 0.20	85.84 ± 0.22	75.24 ± 0.21	72.84 ± 0.48	51.73 ± 1.67
A_X	89.97 ± 0.21	94.22 ± 0.22	93.69 ± 0.13	80.89 ± 0.56	69.84 ± 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 ± 0.16	93.86 ± 0.18	94.32 ± 0.26	84.14 ± 0.47	72.17 ± 0.66
GPR-GNN-10	87.62 ± 0.85	93.52 ± 0.39	$\mathbf{94.81 \pm 0.27}$	85.77 ± 0.67	73.22 ± 0.73

Conclusion

- **Augment the MPNN with a simple normalization step**
  Express the top-k eigenvectors of the graph operator, which is agnostic to the graph topology (homophoily or heterophily).

Conclusion

- **Augment the MPNN with a simple normalization step**
  Express the top-k eigenvectors of the graph operator, which is agnostic to the graph topology (homophoily or heterophily).
- **Couple `PowerEmbed` with an inception network**
  Learn the rich representations that interpolate from *local* message-passing features to *global* spectral information, which provably avoids *over-smoothing* and *over-squashing*.

Conclusion

- **Augment the MPNN with a simple normalization step**
  Express the top-*k* eigenvectors of the graph operator, which is agnostic to the graph topology (homophoily or heterophily).
- **Couple `PowerEmbed` with an inception network**
  Learn the rich representations that interpolate from *local* message-passing features to *global* spectral information, which provably avoids *over-smoothing* and *over-squashing*.
- **Perform comprehensive studies to show `PowerEmbed`'s superiority**
  Demonstrate numerically that our simple techniques achieve competitive performance for node classification in a wide range of simulated and real-world graphs.

Discussion - Future Work

- **Extend `PowerEmbed` to semi-supervised graph learning tasks**
  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.

Discussion - Future Work

- **Extend `PowerEmbed` to semi-supervised graph learning tasks**
  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.
- **Consider more powerful versions of `PowerEmbed` based on higher-order MPNNs instead of local ones**
  It remains open to fully understand the relations between graph spatial information (i.e., symmetries) and graph spectral information (e.g., eigenvalues and eigenvectors).

Thanks!

Q & A