From Cluster Assumption to Graph Convolution: Graph-based Semi-Supervised Learning Revisited

Zheng Wang, Hongming Ding, Li Pan, Jianhua Li, Zhiguo Gong, Philip S. Yu

Shanghai Jiao Tong University

Project page: https://github.com/zhengwang100/ogc_ggcm
Friendly tips: codes in PyG, DGL and rLLM are also provided in the above page.

Outline

- **Background**
- Understanding GCNs
- Our methods
- Experiments
- Conclusion

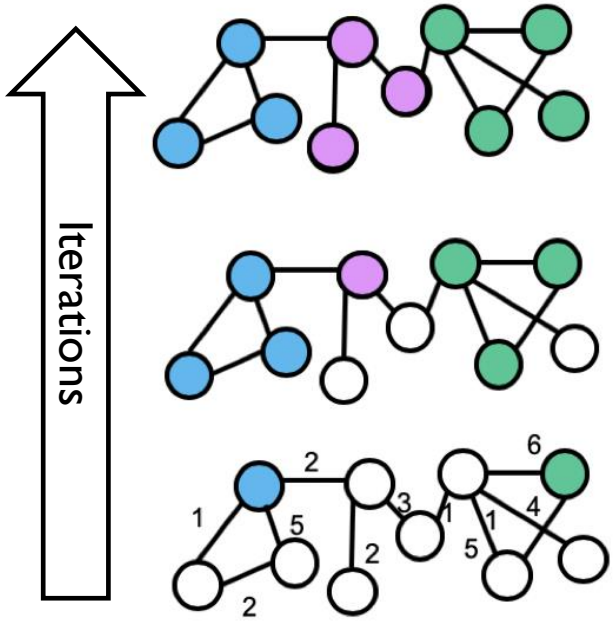
Background: Graph-based Semi-Supervised Learning (GSSL)

- GSSL focus on graph learning, especially graph structure guidance, including
  - Traditional Shallow methods, like LP [1] and Consistency [2]. (mainly from 1990s to 2010s)
  - Deep (neural network) methods, like GCN [3] and GAT [4] (mainly from 2016 to now)

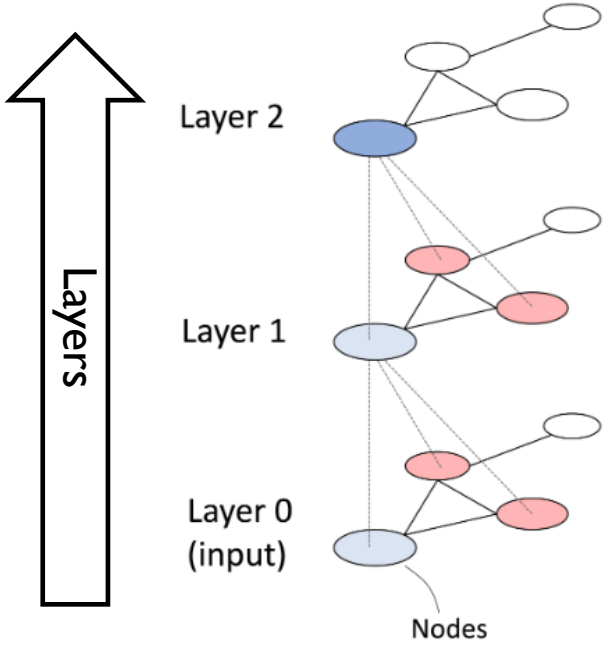
[1] Xiaojin Zhu, Zoubin Ghahramani, and John Lafferty. Semi-supervised learning using gaussian fields and harmonic functions.
[2] Zhou, Dengyong, et al. "Learning with local and global consistency." Advances in neural information processing systems 16 (2003).
[3] Kipf, Thomas N., and Max Welling. "Semi-supervised classification with graph convolutional networks." arXiv preprint arXiv:1609.02907 (2016).
[4] Veličković, Petar, et al. "Graph attention networks." arXiv preprint arXiv:1710.10903 (2017).

An Interesting Problem

- Why do deep GCNs encounter the over-smoothing problem while traditional shallow GSSL methods do not, despite both progressing through the graph in a similar iterative manner?



Label Propagation



Graph Convolutional Network (GCN)

Picture from https://mbernste.github.io/posts/gcn/

Outline

- Background
- **Understanding GCNs**
- Our methods
- Experiments
- Conclusion

Traditional GSSL Methods

- The key is the cluster assumption (i.e., linked nodes tend to have similar class labels), by jointly considering a supervised classification loss (Qsup) and an unsupervised Laplacian smoothing loss (Qsmo):

$$\min_{f,g} \quad \mathcal{Q} = \mathcal{Q}_{sup} + \mathcal{Q}_{smo}$$
$$= \sum_{v_i \in \mathcal{V}_L} \ell\left(g(f(X_i)), Y_i\right) + \mathrm{tr}(f(X)^\top L f(X)),$$

GCN Type Methods

- GCN performs two basic operations on each node.

1. Graph convolution: aggregating information for each node from its neighbors:

$$\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(k)}$$

2. Feature mapping: feeding the above convolution results to a fully-connected NN layer:

$$H^{(k+1)} = \sigma(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(k)}\Theta^{(k)})$$

SGC Analysis (I)

Without loss of generality, we still consider a self-loop graph $\tilde{A}$ with its degree matrix $\tilde{D}$, and use $\tilde{L} = \tilde{D} - \tilde{A}$ to denote the Laplacian matrix. Let $U = f(X)$ denote the learned node embedding matrix, i.e., U is a matrix of node embedding vectors.[4] In addition, we further symmetrically normalize $\tilde{L}$ as $\tilde{D}^{-\frac{1}{2}}\tilde{L}\tilde{D}^{-\frac{1}{2}}$, and initialize U with the node attribute matrix X at the beginning, i.e., $U^{(0)} = X$. As such, the objective function in Eq. 1 becomes:

$$\min_{U,g,U^{(0)}=X} \bar{\mathcal{Q}} = \bar{\mathcal{Q}}_{sup} + \bar{\mathcal{Q}}_{smo} = \sum_{v_i \in \mathcal{V}_L} \ell(g(U_i), Y_i) + \mathrm{tr}(U^\top \tilde{D}^{-\frac{1}{2}}\tilde{L}\tilde{D}^{-\frac{1}{2}}U), \tag{3}$$

where $\bar{\mathcal{Q}}_{sup}$ is the supervised classification loss, and $\bar{\mathcal{Q}}_{smo}$ is the unsupervised smoothing loss.

SGC Analysis (II)

With the standard gradient descent, we can reduce the smoothness loss in SGC as:

$$U^{(k+1)} = U^{(k)} - \eta_{smo} \frac{\partial \bar{\mathcal{Q}}_{smo}}{\partial U^{(k)}}$$

Let $\beta = 2\eta_{smo}$ and ensure $\beta \in (0, 1)$.

$$U^{(k+1)} = [\beta \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} + (1 - \beta) I_n] U^{(k)}$$

Lazy Graph Convolution (LGC)

LGC acts as a bridge between graph convolution and gradient descent:

If we set beta=1, we get the standard graph convolution:

$$(\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}) U^{(k)}$$

If we set beta=1/2, we get graph convolution with a residual connection:

$$[\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} + I_n] / 2$$

By adjusting beta, LGC can use a smaller learning rate to slow down the convergence/over-smoothing.

GCN Analysis

The analysis of GCN is analogous to SGC. Actually, the analysis of a single-layer GCN is the same as that of SGC. Here, we consider the k-th layer of a multi-layer GCN.

Like the analysis in Section 3.1, we continue to consider a graph with self-loops, and we still adopt the symmetric normalized Laplacian matrix. Let $H^{(k)} = f(X)$ denote the learned node embedding matrix at the k-th layer. Then, at this layer, the objective function in Eq. 1 becomes:

$$\min_{\substack{H^{(k)},g^{(k)} \\ H^{(0)}=X}} \hat{\mathcal{Q}}^{(k)} = \hat{\mathcal{Q}}^{(k)}_{sup} + \hat{\mathcal{Q}}^{(k)}_{smo} = \sum_{v_i \in \mathcal{V}_L} \hat{\mathcal{Q}}^{(k)}_{sup} + \mathrm{tr}(H^{(k)\top} \tilde{D}^{-\frac{1}{2}} \tilde{L} \tilde{D}^{-\frac{1}{2}} H^{(k)}), \tag{4}$$

where $g^{(k)}$ stands for the mapping function at the k-th layer. $\hat{\mathcal{Q}}^{(k)}_{sup}$ and $\hat{\mathcal{Q}}^{(k)}_{smo}$ stand for the supervised classification loss and unsupervised Laplacian smoothing loss at the k-th layer, respectively.

Summary I: Traditional GSSL vs GCNs

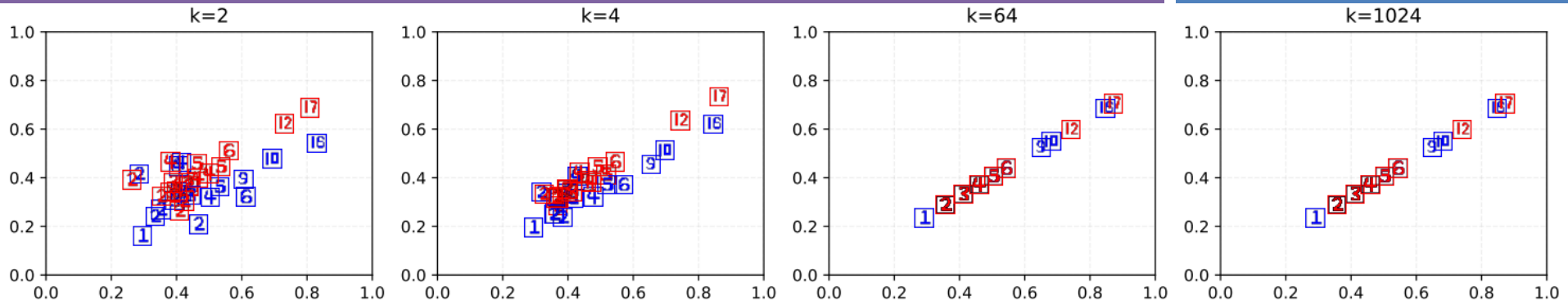
In addition, the main difference of the typical GCNs compared to the classical shallow GSSL methods are summarized as follows. First, typical GCNs further consider a self-loop graph (i.e., $\tilde{A}$ in the graph convolution process defined in Eq. 2). Second, typical GCNs may involve some non-linear operations (i.e., $\sigma(\cdot)$ in the convolution process), and have a larger model capacity (if consisting of multiple trainable layers). Last, and most notably, typical GCNs may not guarantee to jointly reduce the supervised classification loss and the unsupervised Laplacian smoothing loss at each layer. In other words, typical GCNs may fail to jointly consider the graph structure and label information at each layer. This may bring new insights for designing novel GCN-type methods.

Summary II: Over-smoothing Results

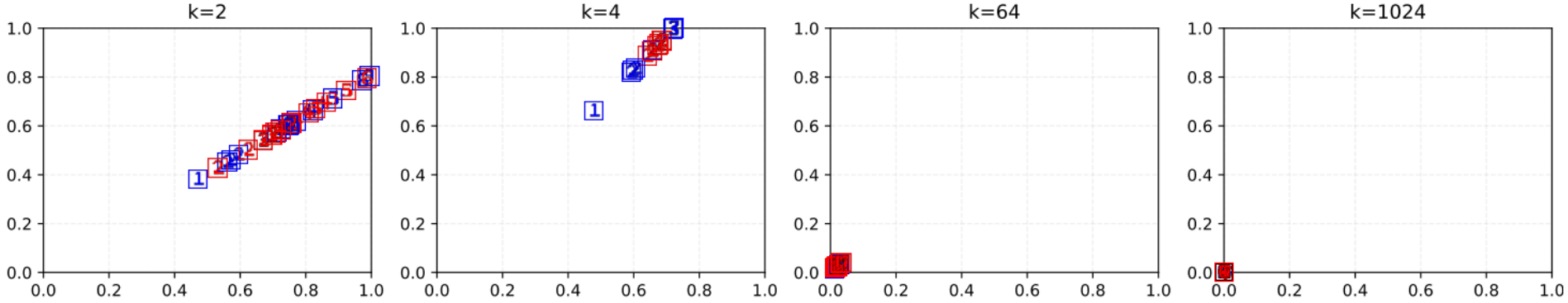
Theorem 1. *In SGC and GCN, if the node embedding results converge at the k-th layer, for each node pair* $< v_i, v_j >$ *in a connected component, we can obtain that:* $U_i^{(k)} = \frac{\sqrt{D_{ii}+1}}{\sqrt{D_{jj}+1}} U_j^{(k)}$ *and* $H_i^{(k)} = \frac{\sqrt{D_{ii}+1}}{\sqrt{D_{jj}+1}} H_j^{(k)}$.

The precise node embedding distribution in typical GCNs is theoretically proved!

Numerical Verification of Theorem 1



(a) SGC



(b) GCN

Figure 3: Embedding visualization on Zachary's karate club network. Colors denote class labels, and numbers (in squares) denote node degrees.

Numerical Verification of Theorem 1 (Continued)

Table 5: Parts of the detailed node embedding results in Zachary's karate club network.

No. (Deg.)	SGC (k=64/1024)	GCN (k=64)	GCN (k=1024)
11 (1)	[.2904, .2344]	[.0111, .0137]	$[1.09e-20, 4.39e-21]$
...	...	...	...
4 (3)	[.4107, .3314]	[.0157, .0193]	$[1.53e-20, 6.21e-21]$
10 (3)	[.4107, .3314]	[.0157, .0193]	$[1.53e-20, 6.21e-21]$
...	...	...	...
5 (4)	[.4591, .3705]	[.0175, .0216]	$[1.72e-20, 6.94e-21]$
6 (4)	[.4591, .3705]	[.0175, .0216]	$[1.72e-20, 6.94e-21]$
7 (4)	[.4591, .3705]	[.0175, .0216]	$[1.72e-20, 6.94e-21]$
...	...	...	...
0 (16)	[.8466, .6832]	[.0323, .0399]	$[3.16e-20, 1.28e-20]$
33 (17)	[.8712, .7031]	[.0333, .0410]	$[3.26e-20, 1.32e-20]$

1. Nodes (in the same connected component) with the same degree tend to converge to the same results.
2. Ratio of the converged embeddings of nodes vi and vj is always sqrt(Dii+1)/sqrt(Djj+1), where D is the degree matrix.

Outline

- Background
- Understanding GCNs
- **Our methods**
- Experiments
- Conclusion

Proposed Methods by Fixing Over-smoothing

- Supervised Method: OGC
  - OGC jointly considers both the smooth loss and supervised loss in the convolution process.

- Unsupervised Methods: GGC and GGCM
  - Their aims are both to preserve the graph structure during the feature aggregation procedure.

A Supervised Method: OGC

At the each (i.e., k-th) iteration, OGC can be formulated in one line:

$$U^{(k+1)} = \underbrace{[\beta \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} + (1-\beta) I_n] U^{(k)}}_{\text{Lazy Graph Convolution}} - \underbrace{\eta_{sup} S(-Y + Z^{(k)}) W^{\top}}_{\textit{Supervised EmBedding (SEB)}}, \quad (6)$$

S is a diagonal matrix with $S_{ii} = 1$ if node v_i is labeled, and $S_{ii} = 0$ otherwise.

$Z^{(k)}$ is the predicted soft labels at the k-th iteration

Pseudo-code of OGC

Algorithm 1 OGC

Input: Graph information (A and X), the max iteration number K, and the label information of a small node set $\mathcal{V}_L$

Output: The label predictions

1: Initialize $U^{(0)} = X$ and $k = 0$
2: **repeat**
3: $k = k + 1$
4: Update W manually or by some automatic-differentiation toolboxes (Sect. 4.1)
5: Update $U^{(k)}$ via (lazy) supervised graph convolution (Eq. 6)
6: Get the label predictions $\hat{Y}^{(k)}$ from: $Z^{(k)} = U^{(k)}W$
7: **until** $\hat{Y}^{(k)}$ converges or $k \geq K$
8: **return** $\hat{Y}^{(k)}$

OGC is very simple with only three steps (at each iteration):

1. Update the label prediction function W
2. Update the node embedding results U
3. Get the label prediction results

Less Is More (LIM) Trick

- OGC is a shallow method, which means it does not need huge validation set. In the learning process, we use both train and validation sets as the supervised knowledge.

To further mitigate the risk of overfitting, we introduce a novel trick: **Less Is more (LIM) trick**. Specifically, when updating U in Eq. 6, we only use the train set to build the label indicator diagonal matrix S.

Unsupervised Methods: GGC and GGCM

- Their aims are both to preserve the graph structure during the feature aggregation procedure, which includes to ensure two parts:

1. Similarity between linked nodes via the classical graph convolution

2. Dissimilarity between unlinked nodes via the proposed Inverse Graph Convolution (IGC)

$$U^{(k+1)} = (\neg\tilde{D}^{-\frac{1}{2}}\neg\tilde{A}\neg\tilde{D}^{-\frac{1}{2}})U^{(k)}.$$

$\neg A \in \mathbb{R}^{n\times n}$ denote a randomly generated sparse "negative" adjacency matrix

$\neg D$ is the degree matrix of $\neg A$

$\neg\tilde{A} = 2I_n - \neg A$ and $\neg\tilde{D} = 2I_n + \neg D$.

The Expressive Power of IGC

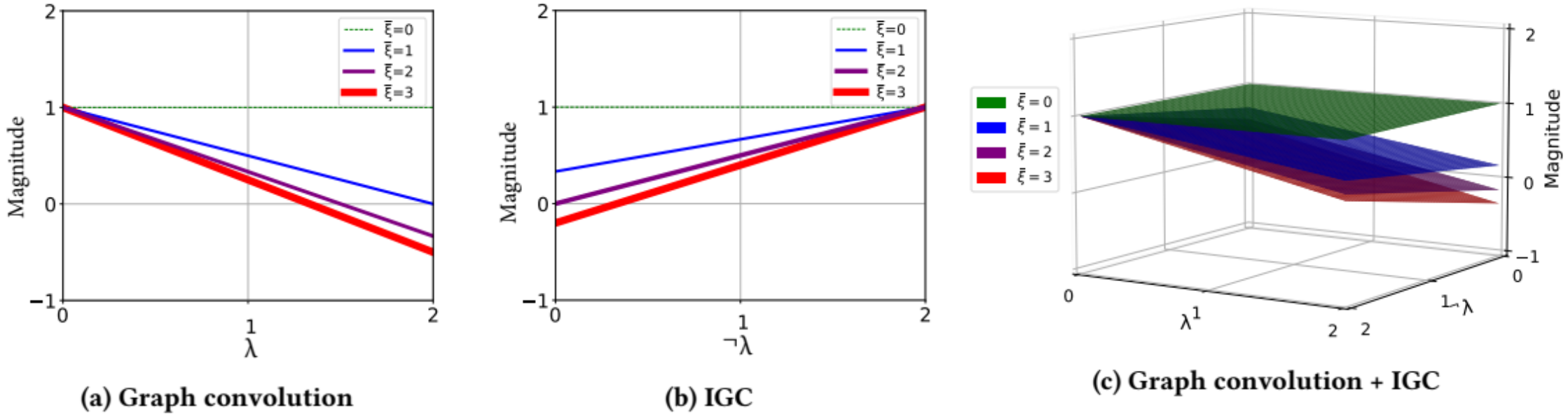


Figure 1: Filter functions. λ and $\neg\lambda$ denote the eigenvalues of symmetric normalized Laplacian matrix of A and $\neg A$, respectively.

Three key properties:

1. IGC is a high-pass filter
2. IGC is numerically stable.
3. Combining IGC and GC will preserve the graph strcture.

Pseudo-code of GGC

Algorithm 2 GGC

Input: Graph information (A and X), the max iteration number K, and moving out probability β

Output: The learned node embedding result set $\{U^{(k)}|k = 0 : K\}$

Initialize $U^{(0)} = X$
for $k = 1$ to K **do**
    Get $U^{(k)}_{smo}$ via (lazy) graph convolution
    Get $U^{(k)}_{sharp}$ via (lazy) IGC (Eq. 10)
    Get $U^{(k)} = (U^{(k)}_{smo} + U^{(k)}_{sharp})/2$
    Decline β with a decay factor
end for
return $\{U^{(0)}, U^{(1)}, ..., U^{(K)}\}$

GGC is very simple with only three steps (at each iteration):

1. Conduct lazy graph convolution to get the first embedding part
2. Conduct lazy inverse graph convolution to get the other embedding part
3. Sum the above obtained two embedding parts

Pseudo-code of GGCM

Algorithm 3 GGCM

Input: Graph information (A and X), the max iteration number K, and moving out probability β

Output: The learned multi-scale node embedding result set $\{{U_M}^{(k)}|k = 0 : K\}$

Initialize $U^{(0)} = X$
for $k = 1$ to K **do**
    Get $U_{smo}^{(k)}$ via (lazy) graph convolution
    Get $U_{sharp}^{(k)}$ via (lazy) IGC (Eq. 10)
    Set $U^{(k)} = U_{smo}^{(k)}$
    ${U_M}^{(k)} = \alpha X + (1-\alpha)\frac{1}{k}\sum_{t=1}^{k}[(U_{smo}^{(t)} + U_{sharp}^{(t)})/2]$
    Decline β with a decay factor
end for
return $\{{U_M}^{(0)}, {U_M}^{(1)}, ..., {U_M}^{(K)}\}$

At each iteration, GGCM just separately sums two kinds of embedding results in GGC.

Outline

- Background
- Understanding GCNs
- Our methods
- **Experiments**
- Conclusion

Node Classification Results

	Method	#Layer	Cora	Citeseer	Pubmed
supervised	LP	–	71.5	48.9	65.8
	ManiReg	–	59.5	60.1	70.7
	GCN	2	81.5	70.3	79.0
	APPNP	2	83.3	71.8	80.1
	Eigen-GCN	2	78.9± 0.7	66.5±0.3	78.6±0.1
	GNN-LF/HF	2	84.0±0.2	72.3±0.3	80.5±0.3
	C&S	3	84.6±0.5	75.0±0.3	81.2±0.4
	NDLS	2	84.6±0.5	73.7±0.6	81.4±0.4
	ChebNetII	2	83.7±0.3	72.8±0.2	80.5±0.2
	OAGS	2	83.9±0.5	73.7±0.7	81.9±0.9
	JKNet	{4, 16, 32}	82.7 ± 0.4	73.0 ± 0.5	77.9 ± 0.4
	Incep	{64, 4, 4}	82.8	72.3	79.5
	GCNII	{64, 32, 16}	85.5±0.5	73.4 ± 0.6	80.2 ± 0.4
	GRAND	{8, 2, 5}	85.4±0.4	75.4±0.4	82.7±0.6
	ACMP	{8, 4, 32}	84.9±0.6	75.5 ± 1.0	79.4±0.4
	OGC (ours)	> 2	**86.9 ± 0.0**	**77.5 ± 0.0**	**83.4 ± 0.0**
unsupervised	DeepWalk	–	67.2	43.2	65.3
	node2vec	–	71.5	45.8	71.3
	VERSE	–	72.5 ± 0.3	55.5 ± 0.4	–
	GAE	4	71.5 ± 0.4	65.8 ± 0.4	72.1 ± 0.5
	DGI	2	82.3 ± 0.6	71.8 ± 0.7	76.8 ± 0.6
	DGI Random-Init	2	69.3 ± 1.4	61.9 ± 1.6	69.6 ± 1.9
	BGRL	2	81.1±0.2	71.6±0.4	80.0±0.4
	N2N	2	83.1±0.4	73.1±0.6	80.1±0.7
	SGC	2	81.0 ±0.0	71.9 ±0.1	78.9 ±0.0
	S^2GC	16	83.0 ±0.2	73.6 ±0.1	80.2 ±0.2
	G^2CN	10	82.7	73.8	80.4
	GGC (ours)	> 2	81.8 ± 0.3	73.8 ± 0.2	79.6 ± 0.1
	GGCM (ours)	> 2	**83.6 ± 0.2**	**74.2 ± 0.1**	**80.8 ± 0.1**

State of the Art | Node Classification on CiteSeer with Public Split: fixed 20 nodes per class

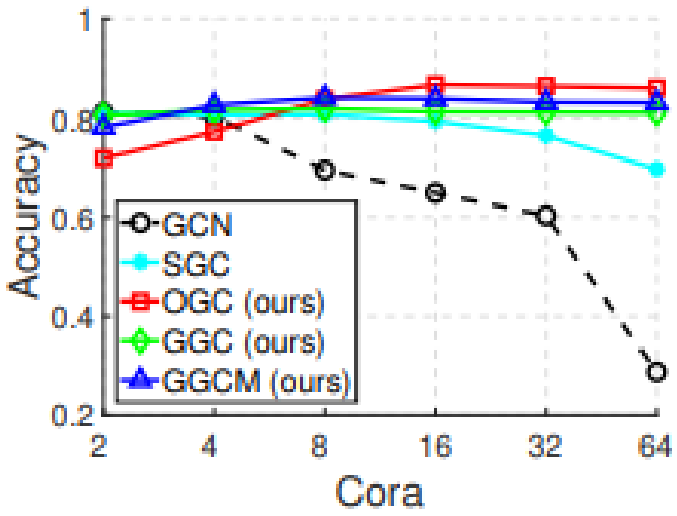
State of the Art | Node Classification on Cora with Public Split: fixed 20 nodes per class

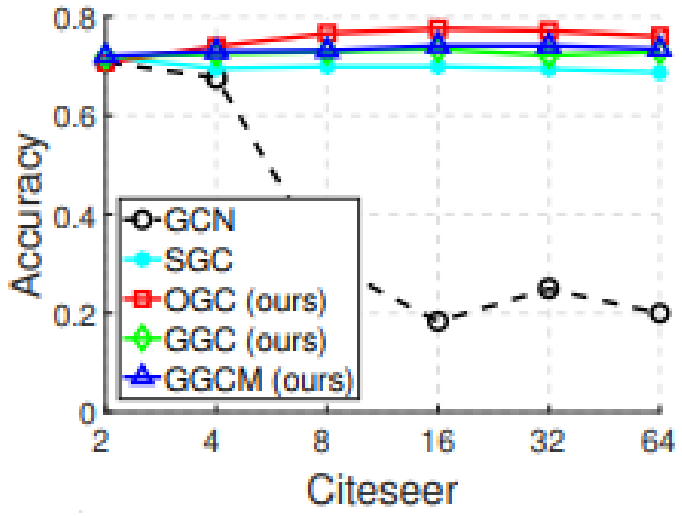
State of the Art | Node Classification on PubMed with Public Split: fixed 20 nodes per class

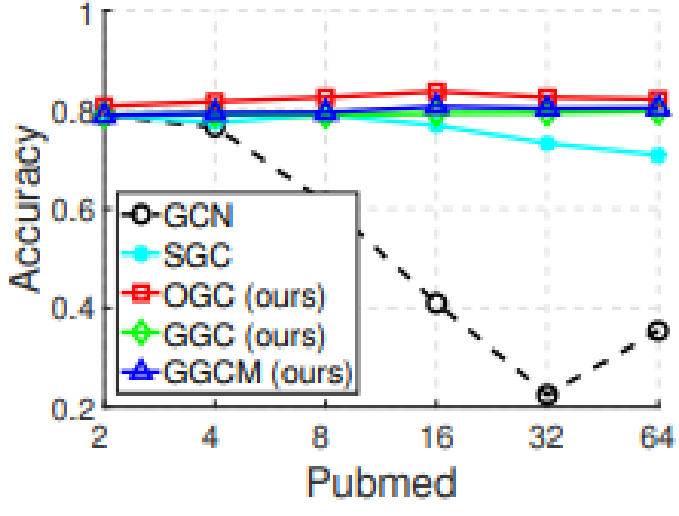
Recorded by Paperswithcode.com

Both OGC and GGCM obtain new SOTA!

Node Classification w.r.t. Layer Number







To fix the over-smoothing problem, OGC introducing supervised knowledge, and GGC/GGCM preserves graph structure in the graph convolution process.

Efficiency Testing

Table 7: Running time (seconds) on Pubmed.

Type	Method	Running on CPU							Running on GPU						
		Layers/Iterations						Total*	Layers/Iterations						Total*
		2	4	8	16	32	64		2	4	8	16	32	64	
supervised	GCNII	98.02	234.60	545.44	923.61	1555.37	4729.32	8086.36	4.60	10.76	29.64	54.08	200.20	780.24	1079.52
	GRAND	857.23	952.29	2809.84	4361.91	6225.39	15832.91	31039.57	35.45	70.99	274.95	2090.65	2961.71	6829.78	12263.54
	OGC	1.88	3.69	7.29	14.63	30.36	61.88	61.88				-			-
unsupervised	SGC	0.22	0.28	0.44	0.77	1.46	2.80	5.78	0.52	0.52	0.55	0.61	0.70	0.82	3.59
	S^2GC	0.25	0.36	0.56	0.94	2.16	3.66	6.88	0.57	0.60	0.62	0.66	0.75	0.92	3.99
	GGC	0.27	0.50	0.85	1.60	3.10	6.02	10.26	0.58	0.59	0.62	0.67	0.80	1.01	4.08
	GGCM	0.37	0.57	1.06	1.93	3.77	7.27	12.45	0.60	0.63	0.64	0.69	0.85	1.09	4.35

OGC is 130-500 times more efficient than those SOTA deep GNNs.

Outline

- Background
- Understanding GCNs
- Our methods
- Experiments
- **Conclusion**

Conclusion

- The key difference between GCN-type and traditional GSSL methods are:
  1. GCNs further consider to add self-loops for a graph.
  2. GCNs may involve some non-linear operations, and possess a larger model capacity.
  3. GCNs cannot guarantee to jointly reduce the supervised classification loss and the unsupervised Laplacian smoothing loss at each layer.

Future Work

- With the introduced three operators (LGC, SEB, and IGC)
  - Fixing the over-smoothing problem for various GCNs
  - Designing very deep GCNs

- With the introduce LIM trick
  - Better use of validation set
  - Training large models like LLMs