Improving PowerEmbed: A Spectral-Inspired Embedding Method for Graph Neural Network

Yuru Song

Abstract

This course project is based on [2]. The author proposed a new embedding method for graph nodes, named as PowerEmbed. Combining message-passing neural network and intuition from spectral clustering method, PowerEmbed aims to utilize the global structure of the graph to better represent node embeddings. The main novelty of the work is to combine power iteration method with node embedding. Power iteration method is used to calculate the eigenvectors of the graph operator without explicitly performing eigen decomposition. PowerEmbed is reported to perform competitively on real-world datasets. In this project, I will evaluate the capability of the power iteration method to calculate eigenvectors in other graphs not shown in the paper. I will show that it fails to capture the top-k eigenvectors in real-world datasets (WebKB graphs) as promised in the paper. I will explore an alternative method that directly uses eigenvectors of graph operators as additional node embeddings. Finally, I will perform failure analysis on the WebKB datasets for node classification and show the statistics of falsely classified nodes. Codes are based on original repository[1].

1 Paper Summary

1.1 Background introduction

Graph-structured data is ubiquitous in real-world. For example, social networks that represents social interactions and protein folding problem that involves predicting protein structures. To process graph data, graph neural networks have been developed and been proven powerful. Graph representation learning is one of the important aspect of graph neural networks. Specifically, this paper considers embedding method for nodes in a graph. Given graph G with n nodes, adjacency matrix $A \in \mathbb{R}^{n\times n}$, node features $X \in \mathbb{R}^{n\times p}$, the problem is to find a function f that embeds the graph by mapping (A, X) to $h \in \mathbb{R}^{n\times k}$. Each node is embedded as the vector $h_i \in \mathbb{R}^k$. The goal of the embeddings are: 1. similar nodes in G should have similar embeddings; 2. the quality of the embedding should be good enough for downstream tasks, such as node classification or link prediction.

Notation used in the paper is as follows. Given graph G with n nodes, adjacency matrix $A \in \mathbb{R}^{n\times n}$, and node labels $Y \in \mathbb{R}^n$. Denote degree matrix D and identity matrix $I \in \mathbb{R}^{n\times n}$. The graph Laplacian is defined as $L = (D + I)^{-1/2}(A + I)(D + I)^{-1/2}$. The random walk graph Laplacian is defined as $L_{rw} = (D + I)^{-1}(A + I)$. Denote a matrix U normalized by its columns as $\frac{U}{|U[:,k]|}$.

1.2 Message-passing neural network

The graph network structure in this paper belongs to the overarching category of message-passing neural network (MPNN) [1]. MPNN is a multi-layer graph neural network that iteratively updates node features. For a L-layer MPNN, the first layer is set to be $h^{(0)} = X$. At each iteration l, the

[1] https://github.com/nhuang37/spectral-inspired-gnn

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

embedding of node i is set as $h_i^l = \phi\Big(h_i^{l-1}, \sum_{j\in\mathcal{N}(i)} \psi(h_i^{l-1}, h_j^{l-1})\Big)$. In this way, the message from neighbouring nodes $\mathcal{N}(i)$ can be passed to node i, using functoin ψ. And node i can update itself using function ϕ. For graph convolutional network (GCN), the message passing function ϕ and update function ψ are set such that $h^l = \sigma(L_{sym}h^{l-1}W^{l-1})$. For simple graph convolution (SGC), the nonlinearity σ becomes linear. Thus for L-th layer, $h^L = L_{sym}^L XW^{L-1}\dots W^0$.

Over-smoothing [3] describes a phenomenon that for deep GNN, its performance does not improve as the number of layers increases. Over-squashing [4] is another issue for GNN, consisting in the distortion of messages being propagated from distant nodes. The theoretical understanding of these two known issues are still under investigation. Despite mentioning the two issues, the paper does not directly address them in their solution. However, the author could argue that they only use very small number of layers in the MPNN, thus the over-smoothing and over-squashing problems may exist.

1.3 PowerEmbed

This algorithm aims to express top-k eigenvectors of a graph operator, e.g. adjacency matrix, random walk Laplacian or symmetric graph Laplacian. Denote a graph operator $S \in \mathbf{R}^{n\times n}$ and node features $X \in \mathbf{R}^{n\times k}$. Algorithmic procedure is as follows:

Initialize a list $P = [X]$ and a matrix $U(t) = X$.
With $t = 0$ to $L-1$, update $U(t+1)$:
1. calculating message passing step with $U(t+1) = SU(t)$;
2. normalizing the message with $U(t+1) = U(t+1)[U(t+1)^T U(t+1)]^{-1}$, and appending $U(t+1)$ to list P.

This algorithm proves to return the top-k eigenvectors (up to an orthogonal transformation) of a symmetric matrix S upon convergence, with the assumptions that the input features X has full column rank and the S has an eigen-gap. This method embeds the original graph to a list of features, which can extracts local features from first few iterations and global information from the list of features. The authors claim that PowerEmbed can avoid over-squashing and alleviate over-smoothing.

1.4 Network structure and benchmark datasets

This paper considers graph node classification. The embedded node features are further processed in an Inception Network. This structure use separate multi-layer perceptron (MLP) to transform features from separate layers. The transformed features from all MLPs are concatenated, which are used by a classifier for final classification.

The first testing dataset is a synthetic graph, constructed from a 2-block stochastic block model (SBM). The nodes either belongs to cluster 0 or cluster 1. Nodes from the same cluster have features drawn from the same m-dimensional multivariate Gaussian distribution. On this dataset, the authors first show that PowerEmbed converges faster on dense graph than sparse graph. Then the algorithm is compared with global spectral methods and graph convolutional networks (GCN). On sparse graphs, PowerEmbed outperforms spectral embedding methods. On dense graphs, PowerEmbed has similar performance with some spectral methods.

Then the algorithm is tested on 10 real-world graph datasets, ranging from webpage networks, citation networks to co-purchase networks. In all heterophily graph datasets, PowerEmbed performs the best. In homophilous graph datasets, PowerEmbed performs best only in 1 out of 5 datasets.

In summary, this paper proposes a method to combine both local and global information from the graph. It uses a layer-wise normalization method to express top-k eigenvectors of a graph. Although this algorithm can achieve improved performance on heterophily datasets, its performance on homophilous datasets is not as good as other existing methods.

2 Criticism of the paper

Overall, this paper can be better written. The paper tends to use decorative math to justify their engineer intuition. For example, when describing over-smoothing and over-squashign, the authors

use 'Definition 3.4' and 'Definition 3.5'. However, neither of the two definitions are later mentioned in their paper. The authors also wrote in mathematical terms how PowerEmbed is proved to converge to top-k eigen-subspace. Yet, they did not apply their theoretical to real-world datasets. For example, the authors should also show convergence speed of power iteration on real-world graphs.

The experimental simulations can be more rigorous in this paper. For example, the author only showed the convergence of power iteration with a synthetic SBM and only used one size of the graph ($n = 500$). However, as I will show in the report, power iteration on smaller size of the SBM graph converges much slower. On node classification task of real world datasets, I find it difficult to reproduce the original results. For example, in Cornell datasets, the authors claim PowerEmbed can achieve accuracy as high as $78.30 \pm 1.58\%$. However, my reproduced results could only achieve less than 73%.

3 Project Report

3.1 Re-evaluating power iteration in synthetic and real-world dataset

In the paper, authors first show power iteration can converge to top-k eigenvectors on stochastic block model (SBM). The SBM has two blocks (2B-SBM). Nodes within the same block have probability of p to make connections. Nodes from different blocks have probability of q to make connections. If p and q are relatively close to 1 (e.g. $1/2$), then the graph is dense. Otherwise, the graph is sparse. Nodes within the same block have the same label and similar node features. If $p > q$, the 2B-SBM is a homophilous graph. Otherwise it's a heterophilous graph.

First, I varied the number of nodes in the graph to smaller numbers, shown in Fig. 1. For $(n = 300, p = 1/2, p/3)$, the power iteration takes about 15 iterations to converge to first eigenvector of graph Laplacian. However, for a smaller network $(n = 100, p = 1/2, p/3)$, it takes about 30 iterations to converge. For sparse network $(p = 1/10, q = 1/15)$ and $(p = 1/20, q = 1/30)$, it takes more than 50 iterations to converge to the first eigenvector.

The authors did not provide convergence speed of power iteration for other graphs. Here, I evaluate the convergence speed on WebKB datasets, which are datasets containing WWW-pages collected from computer science departments of various universities in January 1997 [5]. Each node is one webpage. Edges are hyperlinks between webpages. There are three graphs in the WebKB datasets. The graphs are sparse and heterphilous. The homophily score measure the tendency of node connection within the same label versus across different labels. The score is between 0 and 1, with lower score indicating a more heterophilous graph. Cornell dataset has 183 nodes, 280 edges and homophily score as 0.3. Texas dataset has 183 nodes, 295 edges and homophily score as 0.11.

The procedure is as follows. First, I sample random vectors U with size $n \times k$. Here, n is the number of graph nodes and k is set as 5. Then, I perform PowerEmbed of U and graph adjacency matrix A for different number of iterative steps in power iteration, to get the approximated eigenvectors $\tilde{U}$. Lastly, I evaluate angles between the approximated eigenvectors and the ground truth eigenvectors. The simulations are repeated for 10 times with different initializations. The results are shown in Fig. 2.

Overall, power iteration method take more than 25 iterations to converge. However, in the experimental simulations for node classification task, the paper uses only up to 10 iterations. Thus, PowerEmbed in their simulations may not really utilize the eigen-subspace of graph Laplacian for node classifications in these three datasets.

3.2 Failure analysis of node classification in WebKB datasets

Considering the lower accuracy of node classification on heterophilous graphs, I visualize the statistics of nodes in the WebKB datasets. First, I visualize the label frequency of each graph. Node labels are webpage categories including student, project, course, staff and faculty. In the left column of Fig. 3, we can observe there are imbalance in the graph dataset. For example, in Cornell dataset, there are more than 40% data points belong to staff and less than 10% datapoints belong to project. We can also observe that those misclassified datapoints tend to be labels with less data points, such as project and course in the Cornell dataset. One solution to this could be using weighted loss function during

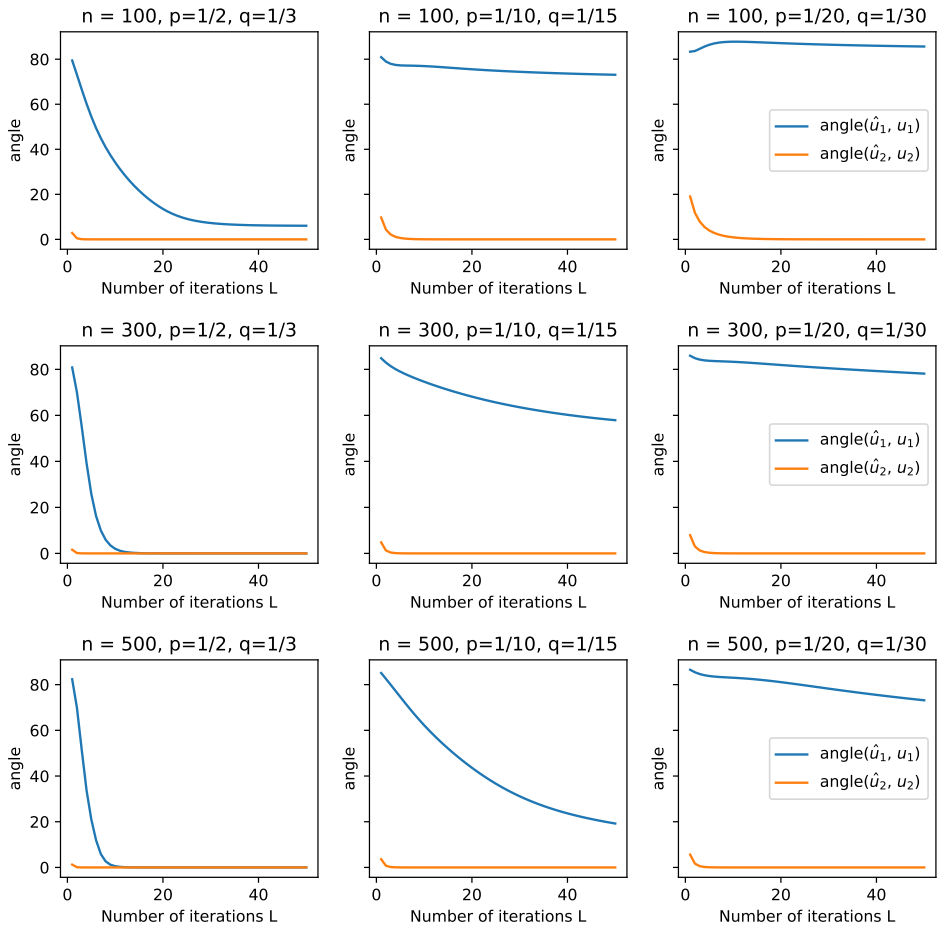


Figure 1: **Convergence to principal subspace on synthetic stochastic block model.** In the original results on 2B-SBM, the author use paramter settings as $(n = 500, p = 1/2, q = 1/3)$, $(n = 500, p = 1/10, q = 1/15)$, and $(n = 500, p = 1/20, q = 1/30)$. I reproduced this result, as shown in the third row. I also changed the size of the synthetic graph by using $(n = 100)$ in the first row and $(n = 300)$ in the second row. Blue curves are the angle between iterative representation and the first eigenvector of graph Laplacian. Orange curves are the angle between iterative representation and the second eigenvector. X-axis is the number of iteration in power iteration method. Y-axis the the angle between vectors, ranging from 0 to 90 degrees. The results are averaged from 10 trials with different random seeds.

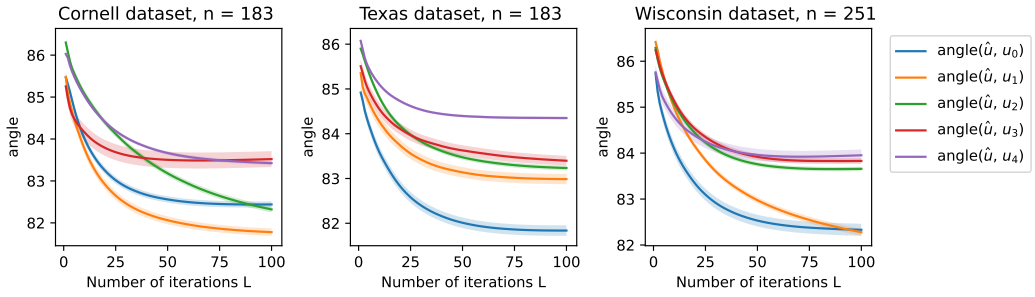
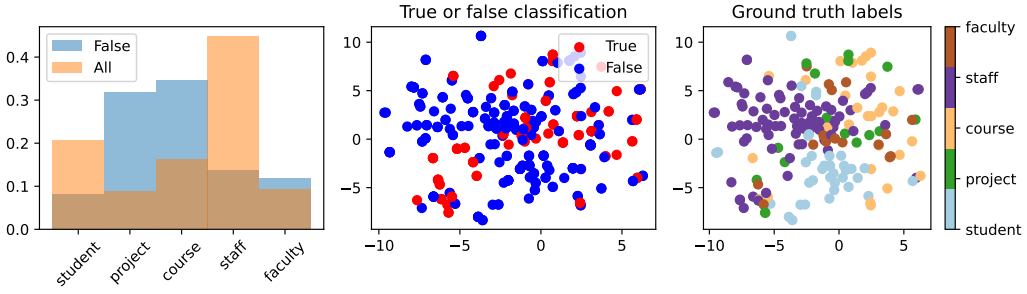


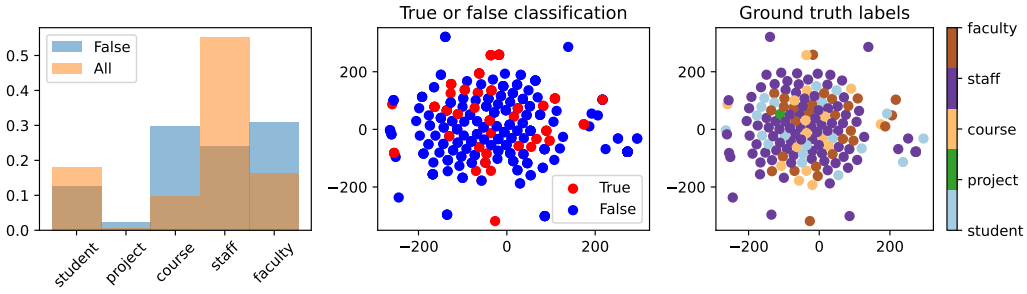
Figure 2: **Convergence to principal subspace on WebKB graphs.** u_0, u_1 to u_4 are the top-5 eigenvectors of the graph Laplacian. X-axis is the number of iteration in power iteration method. Y-axis the the angle between vectors, ranging from 0 to 90 degrees. The results are averaged from 10 trials with different random seeds.

training, such that labels with less data points have higher weights in the loss function. I will show the result of this implementation in the next section.

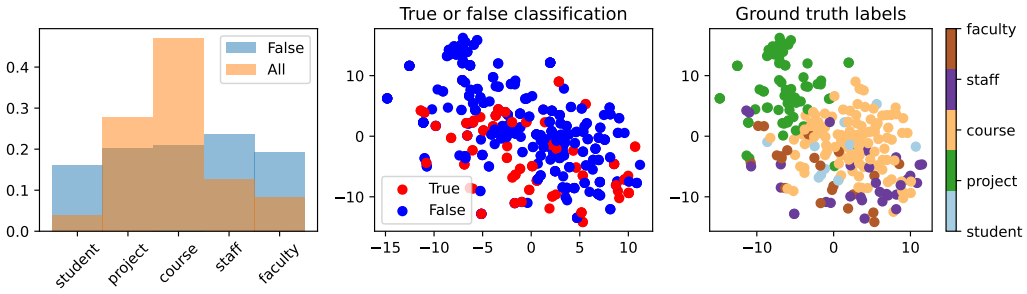
Then, I visualize the tSNE embeddings of node features. Nodes features of these graphs are bag-of-words feature vectors. There are 1700 unique words in all webpages. I first normalize the node features, then use principal component analysis to reduce dimensionality to 50, then use two component tSNE to visualize the features. In middle column of Fig. 3, nodes are colored by the classification results. There are no distinct distribution of misclassified nodes. In right column of Fig. 3, nodes are colored by their labels. There are some clustering of nodes with different labels. For example, in the Cornell dataset, the staff and student nodes seem to have clusters, yet the project and faculty nodes are scattered.



(a) Cornell dataset



(b) Texas dataset



(c) Wisconsin dataset

Figure 3: **Node statistics in WebKB datasets.** Left column: Node label distributions in the graphs. Orange bars represent distribution of node labels for all data points. Blue bars represent distribution of node labels for misclassified data points. Middle column: tSNE embedding of node features colored by classification results. Right column: tSNE embedding of node features colored by node labels. The misclassified data points are aggregated from 10-fold cross validations of all points.

3.3 Weighted cross-entropy loss

As shown in the previous section, the node classifications are less accurate on labels with less data points. To mitigate this, I explored weighting the loss function to balance the labels. More specifically, for each node labels, I first count the number of nodes in the training set, denoted as $n_i, i = 1, 2, ..., 5$. During training, the loss value for each class is weighted using $1/n_i$. The results are shown in Fig. 4. Panels in the left column are PowerEmbed using graph Laplacian. Middle column uses graph adjacency matrix and the right column uses random walk Laplacian. Blue dots are reproduced results from the paper and orange dots are results with weighted loss function. Here, different number of power iterations are used, as indicated on the x-axis. As can be seen, weighted loss function seems to only improve in the Cornell datasets, but not in the Texas or Wisconsin datasets.

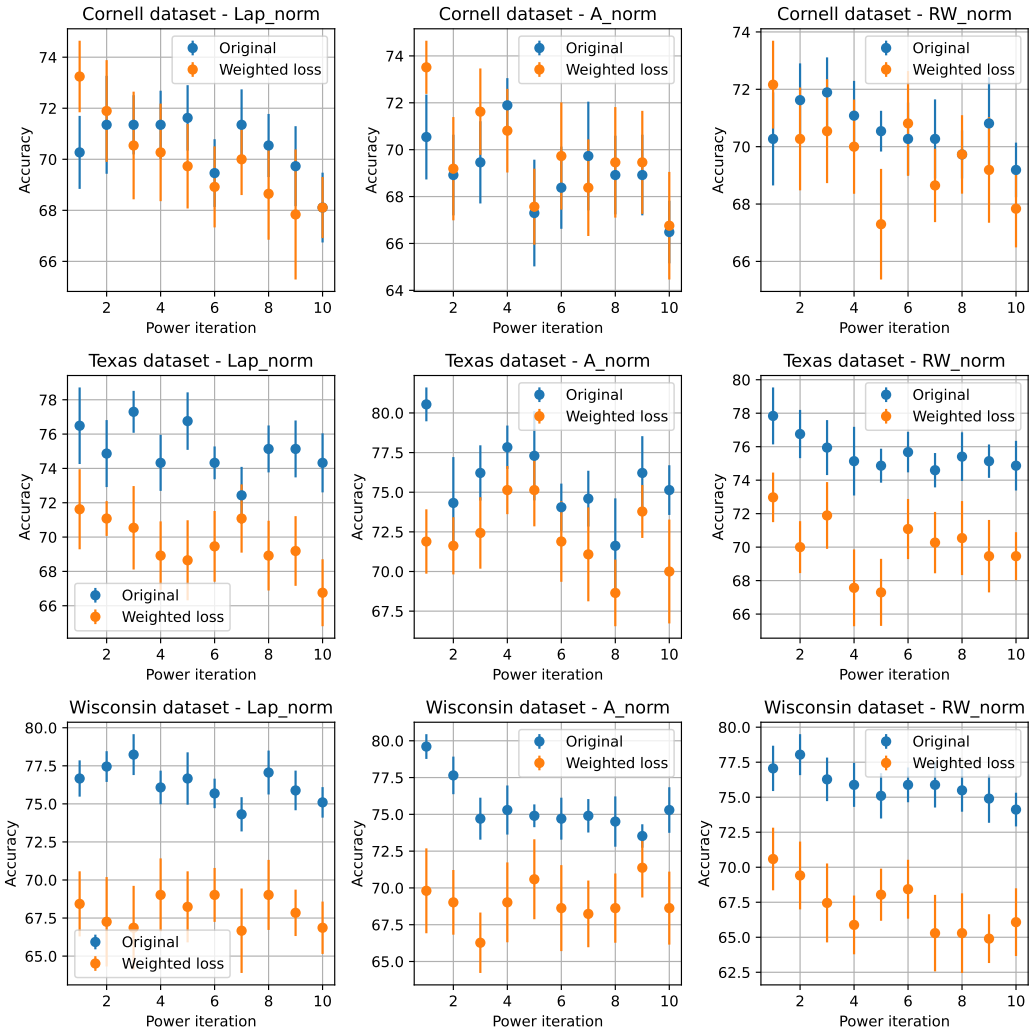


Figure 4: **PowerEmbed using weighted loss.** X-axis is the number of iterations used in PowerEmbed. Y-axis is the accuracy of classification.

3.4 Adding eigenvectors as features in Inception network

In the last section, I show the results of node classification by directly combining graph eigenvectors with embedded node features. More specifically, I consider top-10 eigenvectors of the graph operator as another 'embedded' node features and pass them into the Inception newtork architecture. These eigenvectors will be passed to a multi-layer perceptron for further processing, and later concatenated with other processed embeddings for final classification. Again, the blue dots are original results

and the orange dots are adding top-10 eigenvectors. It seems that in some cases, e.g. Cornell dataset using adjacency matrix and 3 iterations (Fig. 5), adding eigenvectors are helpful. But there is no clear distinction between using or not using top-k eigenvectors. Another drawback of this improvement is that computing eigenvectors directly could be computationally expensive for large networks.

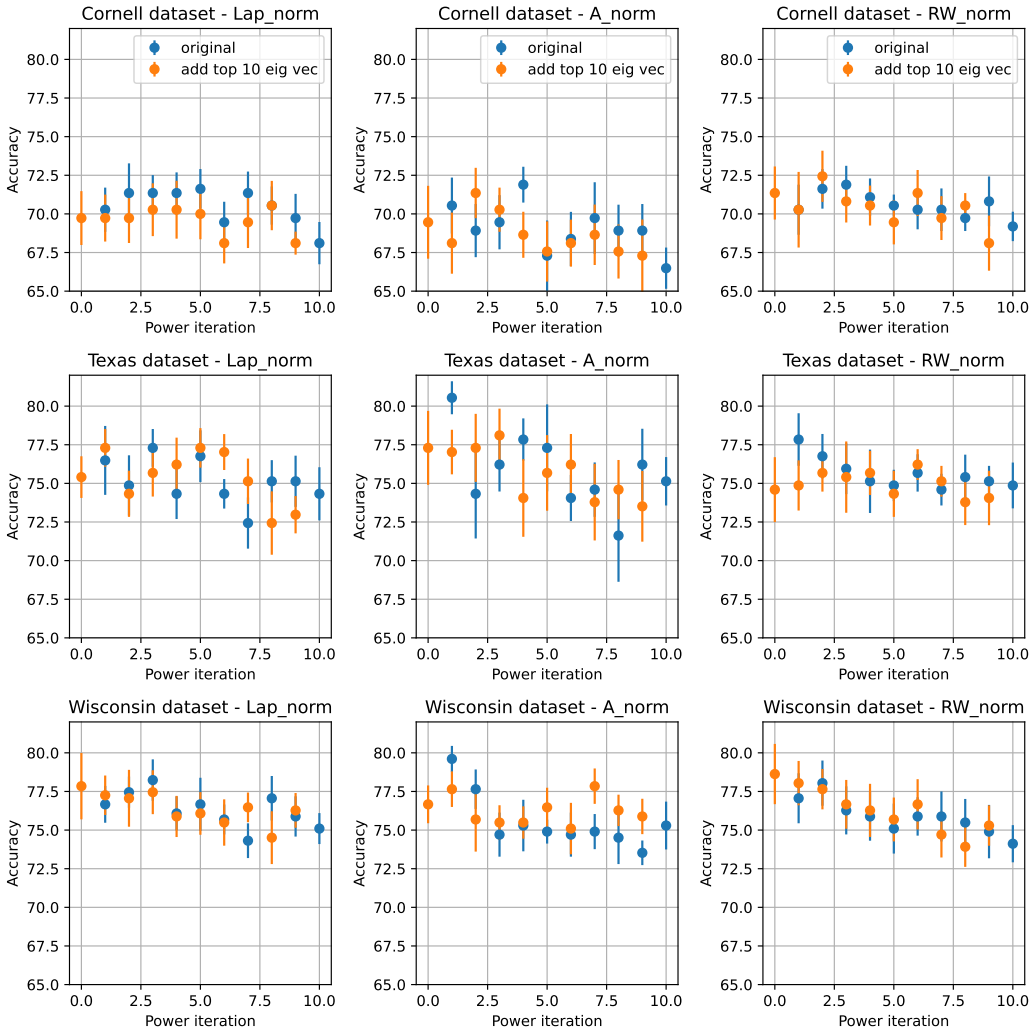


Figure 5: **Using eigenvectors in Inception network**. X-axis is the number of iterations used in PowerEmbed. Y-axis is the accuracy of classification.

3.5 Summary

This course project explored the unsupervised node embedding method with inspiration from spectral clustering. The paper [2] proposed to combine power iteration to implicitly compute graph eigen-subspace. However, in this project, I showed that this method may not converge to top eigen-subspace using small number of layers in the graph neural network. I also explored a couple of ways to improve the node classification results in the paper.

References

[1] Justin Gilmer et al. "Neural message passing for quantum chemistry". In: *International conference on machine learning*. PMLR. 2017, pp. 1263–1272.

[2] Ningyuan Huang et al. *From Local to Global: Spectral-Inspired Graph Neural Networks*. 2022. arXiv: 2209.12054 [stat.ML].

[3] Qimai Li, Zhichao Han, and Xiao-Ming Wu. "Deeper insights into graph convolutional networks for semi-supervised learning". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 32. 1. 2018.

[4] Jake Topping et al. "Understanding over-squashing and bottlenecks on graphs via curvature". In: *arXiv preprint arXiv:2111.14522* (2021).

[5] *WebKB dataset*. 1998. URL: https://www.cs.cmu.edu/afs/cs.cmu.edu/project/theo-20/www/data/.

4 Supplementary

4.1 Eigen-decomposition of graph operators in WebKB datasets

Below shows the eigen-decomposition of graph operators in WebKB datasets (Fig. 6, Fig. 7 and Fig. 8). The graph operators considered are adjacency matrix A, unnormalized graph Laplacian $D - A$, symmetric graph Laplacian $D^{-1/2}(D - A)D^{-1/2}$, random walk Laplacian $D^{-1}(D - A)$ and the last two operators with self-loop versions. Eigenvalues are ordered in the decreasing order of their magnitudes. The first 20 eigenvectors and the last 20 eigenvectors are visualized.

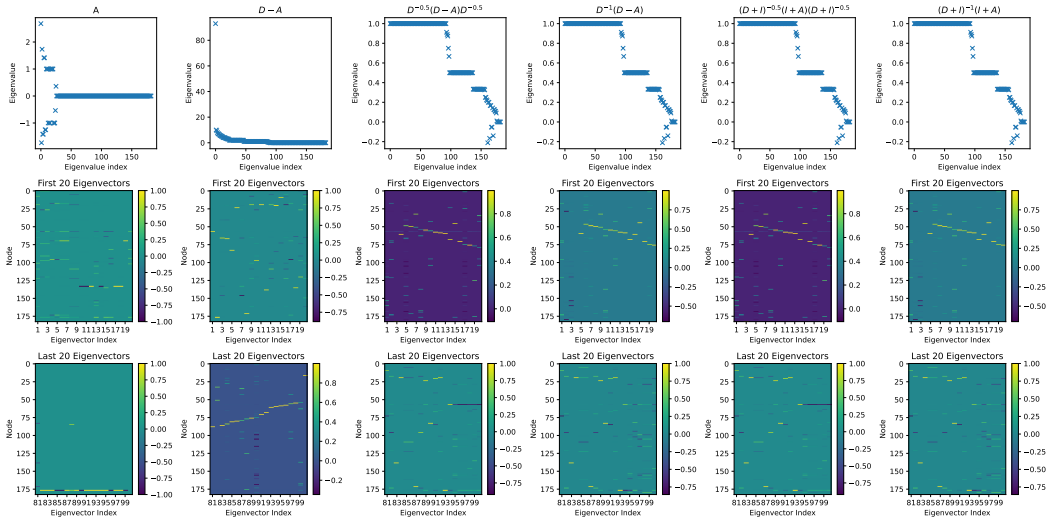


Figure 6: Cornell dataset

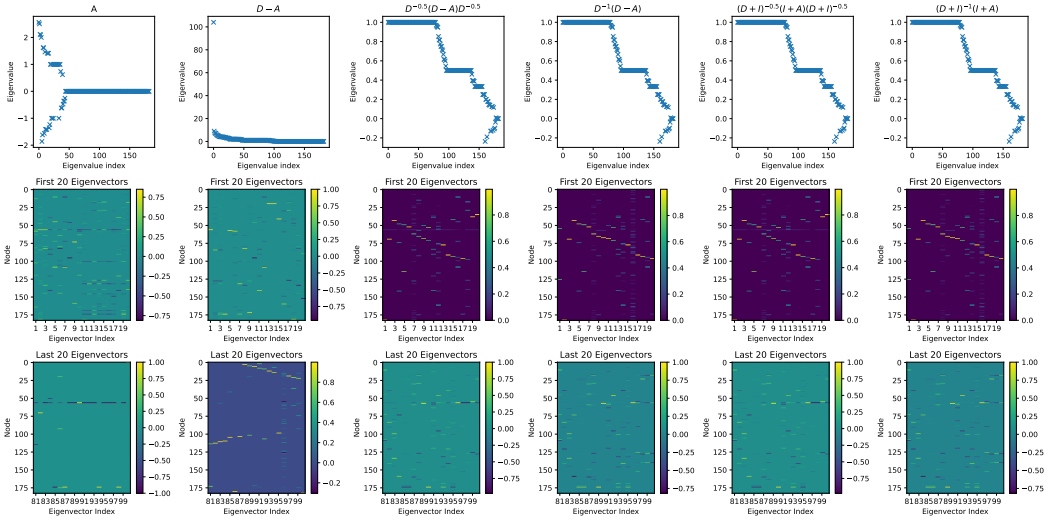


Figure 7: Texas dataset

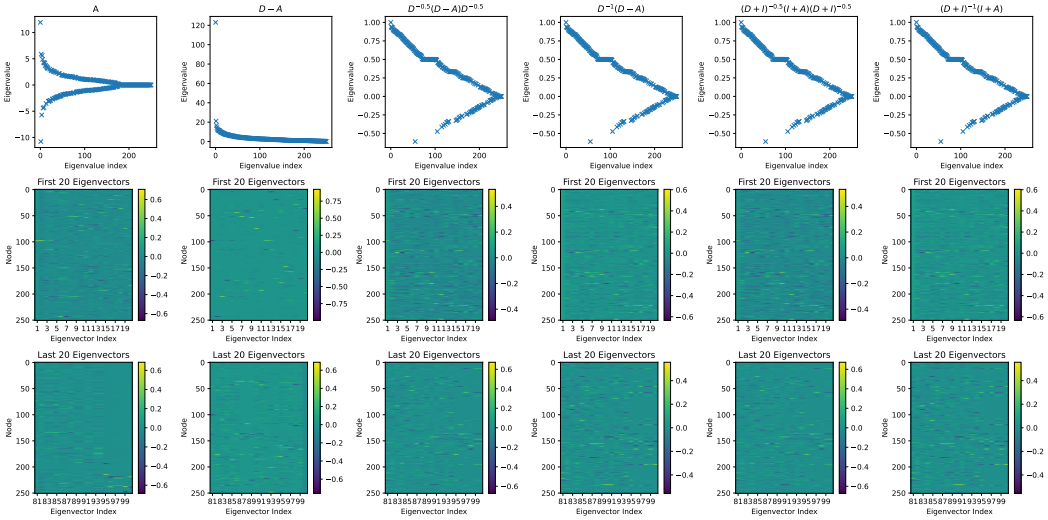


Figure 8: Wisconsin dataset