Degree-based stratification of nodes in Graph Neural Networks

Ameen Ali   AMEENALI@MAIL.TAU.AC.IL
Lior Wolf   WOLF@MAIL.TAU.AC.IL
School of Computer Science, Tel Aviv, Israel

Hakan Cevikalp   HAKAN.CEVIKALP@GMAIL.COM
MLCV Laboratory, Eskisehir Osmangazi University, Eskisehir, Turkey

Abstract

Despite much research, Graph Neural Networks (GNNs) still do not display the favorable scaling properties of other deep neural networks such as Convolutional Neural Networks and Transformers. Previous work has identified issues such as oversmoothing of the latent representation and have suggested solutions such as skip connections and sophisticated normalization schemes. Here, we propose a different approach that is based on a stratification of the graph nodes. We provide motivation that the nodes in a graph can be stratified into those with a low degree and those with a high degree and that the two groups are likely to behave differently. Based on this motivation, we modify the Graph Neural Network (GNN) architecture so that the weight matrices are learned, separately, for the nodes in each group. This simple-to-implement modification seems to improve performance across datasets and GNN methods. To verify that this increase in performance is not only due to the added capacity, we also perform the same modification for random splits of the nodes, which does not lead to any improvement.

Keywords: graph neural networks; node degree; message passing.

1. Introduction

Graph Neural Networks (GNNs) provide an effective way to learn over graphs, which is extremely applicable in domains as diverse as hardware design, economy, medicine, recommendation systems, and physics (Zhou et al., 2020). However, unlike other neural models, this success does not increase with depth. In fact, in most cases, shallow GNNs outperform deeper ones, cf. (Guo et al., 2023).

To overcome the paradoxical behavior of depth, some contributions try to overcome oversmoothing, the phenomenon in which node embeddings become too similar with depth (Wu et al., 2023; Rusch et al., 2023). One approach to this is to incorporate skip connections (Xu et al., 2021, 2020). Such connections are used, for example, in Graph Attention Networks (GATs) (Veličković et al., 2017) and Graph Convolutional Networks with attention mechanisms (GCANs) (Guo et al., 2019). Another approach is to apply specific forms for normalization, e.g., PairNorm (Zhao and Akoglu, 2020), in which pairwise node embeddings in every layer are prevented from becoming too similar, and the recent ContraNorm (Guo

et al., 2023), which achieves the same goal by applying a shattering operator to the latent representation.

In this work, we study a different way to add capacity to GNNs. Instead of trying to add depth, we stratify the nodes by their graph degrees and learn a separate weight matrix for each group. Specifically, we split the nodes of the graph into low- and high-degree nodes and learn two sets of weights (one weight matrix per layer, per group).

Recall that a node's degree is the number of nodes that are connected to it and that this is entirely dependent on the input adjacency matrix. We motivate our method by presenting evidence that the low-degree nodes are harder to classify and that they tend to lie on the decision boundaries. We argue that high-degree nodes, which tend to lie in uniform areas, are less affected by the representation collapse caused by oversmoothing. On the other hand, low-degree nodes require more careful consideration, which is obtained by assigning a specified weight matrix to handle these nodes.

By assigning one weight matrix to low-degree nodes, and another to high-degree nodes, we double the capacity of the GNN. This, as we demonstrate, is helpful across datasets and for multiple GNN architectures. However, as our ablation shows, adding this capacity by randomly splitting the nodes into two groups is detrimental to the overall accuracy, and performs worse than the original GNN. Our work, therefore, demonstrates the need to apply GNNs message passing in a way that takes into account the degree of each node.

2. Related work

Graph Neural Networks (GNNs) have recently attracted increasing interest as a powerful framework for learning graph-structured data. Inspired by the success of Convolutional Neural Networks (CNNs), GNNs aim to extend the idea of convolutions to handle non-grid data (graphs) by leveraging the graph structure. GNNs have been shown to learn expressive node/graph representations (Kipf and Welling, 2016; Hamilton et al., 2017; Xu et al., 2018) and have achieved state-of-the-art results over various graph tasks (Veličković et al., 2017; Bojchevski and Günnemann, 2017; Chen et al., 2020; Gasteiger et al., 2019).

Weight sharing is a fundamental concept that is employed in various neural network architectures, including CNNs (LeCun et al., 1998) and GNNs (Hamilton et al., 2017). In these models, weight sharing refers to the practice of using the same set of weights across different regions or nodes to capture shared patterns and achieve parameter efficiency. By sharing weights, the model can generalize across regions or nodes, effectively capturing common features and relationships. In addition to weight sharing, recent research has explored other techniques to enhance the expressive power of GNNs. Attention mechanisms have gained significant interest in the field of graph representation learning. Graph Attention Networks (GAT) (Veličković et al., 2017) proposed using attention mechanisms for neighbors aggregation, allowing nodes to attend to their neighbors with varying weights. While weight sharing and attention mechanisms have contributed to the success of GCNs, there is ongoing research on further enhancing their capabilities. (Zhang et al., 2022) proposed Graph-adaptive Rectified Linear Unit (GReLU), a topology-aware parametric activation function that incorporates neighborhood information in a novel and efficient way, improving GNN performance across various tasks. (Wang et al., 2022) introduces a novel propagation mechanism in Graph Convolutional Networks (GCNs) that adaptively adjusts the propa-

gation and aggregation process based on the homophily or heterophily between node pairs. (Kim and Oh, 2022) presents an improved model for noisy graphs, utilizing two attention forms compatible with a self-supervised task to predict edges and encode relationships, resulting in more expressive attention in distinguishing mislinked neighbors. In (Chen et al., 2023) the authors argue that the traditional perspective on inter-class edges and existing metrics cannot fully explain the performance of Graph Neural Networks (GNNs) on heterophilic graphs, and propose a new metric based on von Neumann entropy to re-examine the heterophily problem. They also introduce a Conv-Agnostic GNN framework (CAGNNs) that enhances GNN performance on heterophilic datasets by learning the neighbor effect for each node, achieving significant improvements in performance compared to GIN, GAT, and GCN models on various benchmark datasets. (Wu et al., 2019) addresses the limitations of existing graph neural networks by proposing DEMO-Net, a degree-specific graph neural network that incorporates seed-oriented, degree-aware, and order-free graph convolution properties, along with multi-task learning methods and a novel graph-level pooling scheme, achieving superior performance on node and graph classification benchmarks. Inspired by the Transformer architecture, Graph Transformers (Yun et al., 2019; Hu et al., 2020; Ying et al., 2021; Dwivedi and Bresson, 2020; Dwivedi et al., 2021) incorporate a self-attention mechanism with multiple heads to assign varying weights to neighbor nodes, enabling effective information aggregation and representation learning.

3. Motivation

We build our proposed method on the cluster assumption of (Chapelle and Zien, 2005) which states that the decision boundaries should not cross high-density regions and should lie in low-density regions. To this end, we split the node degrees into two separate sets according to high- and low-degree nodes, and create two graphs, by using each. We treat these two graphs as if they were two different graphs and obtain weight matrices for both, which, as we demonstrate, improves classification accuracy.

The data samples corresponding to the nodes with high degrees lie in high-density cluster regions surrounded by the neighboring data samples from the same class categories. On the other hand, the samples from the nodes with lower degrees typically lie in the critical regions where the classes overlap with each other, i.e., inter-class decision boundaries[1]. These nodes are critical for successful classification and the propagation of information among them must be slower so that the nodes from different class categories do not collapse to similar points and hence can be separated from each other. To verify our claims, we classified the nodes with lower and higher degrees separately, using the different GNN baseline methods in Table 1. As can be seen from the table, it is evident that the classification accuracies achieved by the baseline models exhibit a notable disparity between nodes with lower degrees and those with higher degrees across all three datasets. Specifically, the accuracies attained for nodes with lower degrees are considerably inferior in comparison to the accuracies observed for nodes with higher degrees.

In graph convolutional networks, high-degree nodes have a larger number of connections or neighbors, therefore information spreads more quickly and widely through them. On the

---

1. Most of the node embeddings with lower node degrees lie closer to decision boundaries and the remaining ones lie in isolated regions far from both the decision boundaries and high-density cluster regions.

other hand, low-degree nodes have fewer connections and may not be directly connected to as many nodes as high-degree nodes. Therefore, information originating from low-degree nodes may take longer to propagate through the network, since it needs to traverse a longer path through intermediate nodes to reach other parts of the graph. This is also reflected in the spectrum plots of the adjacency matrices obtained from the graphs using lower and higher node degrees given in Fig. 1. As indicated in (Yang et al., 2022), if eigenvalues have similar magnitudes, the resulting graph nodes are more robust to over-smoothing, whereas the node embeddings become similar very rapidly for the graphs with an adjacency matrix containing dissimilar eigenvalues where the majority of the eigenvalues are insignificant (nearly zero). As a result, when we use the full graph structure without splitting the node degrees, the node embeddings coming from the opposing classes near the decision boundaries become rapidly similar. Therefore, the decision boundaries merge and the node embeddings become nonseparable. To avoid this, we split the nodes into lower- and higher-degree nodes, treat them as two different graphs and learn two weight matrices, for each. This way, information will propagate more slowly among nodes with lower degrees, since their eigenvalue spectrum is more uniform. As a result, the model will be more robust to over-smoothing.

4. Method

In this section, we describe the proposed method of using different weight matrices for different node degree groups in a Graph Neural Network. The approach aims to enhance the learning capability of the GNN by considering the varying degrees. By assigning separate weight matrices to different node degrees, the model can capture degree-specific patterns, which improve its expressivity.

4.1. Preliminaries

We begin by providing a brief overview of the general GNN architecture that serves as the foundation for our proposed approach. A GNN operates on a graph represented by an adjacency matrix $A \in \mathbb{R}^{n \times n}$ where n is the total number of nodes in the given graph, and a node feature matrix $X \in \mathbb{R}^{n \times d}$ where d is the feature dimensionality of each node vector. At each layer, i, the GNN updates the node representations based on their neighborhood information and propagates the information through the graph. The specific update rules vary depending on GNN architecture. The updating rule for a GCN model (Kipf and Welling, 2016) can be formulated as follows:

Let $H^i \in \mathbb{R}^{n \times F(i)}$ denote the hidden representations of the different nodes of a certain graph at the i-th layer of the GCN, where $F(i)$ denotes the hidden dimension at layer i, for $i = 0$ we have $F(i) = F(0) = d$, the propagation rule of the GCN can be defined as follows:

$$H^{i+1} = f(A, H^i, W^i) = \sigma(D^{-\frac{1}{2}} \cdot A \cdot D^{-\frac{1}{2}} \cdot H^i \cdot W^i), \qquad (1)$$

where $W^i \in \mathbb{R}^{F(i) \times F(i+1)}$ is the weight matrix at the i-th layer, $\sigma(\cdot)$ is the activation function, like ReLU/ELU and $D \in \mathbb{R}^{n \times n}$ is the degree matrix obtained by $D_{v,u} = \sum_u A_{v,u}$.
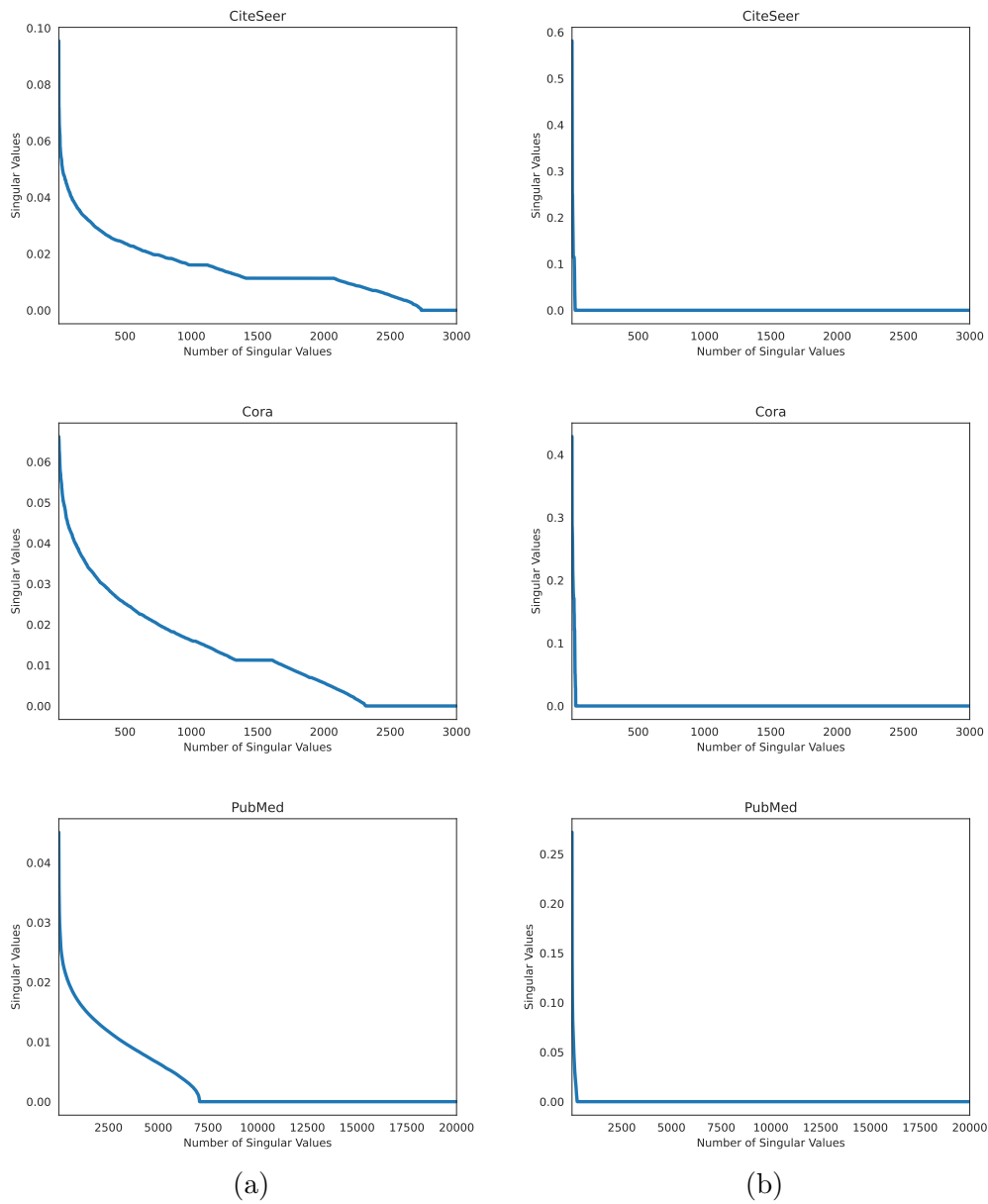
Figure 1: Spectrum of the renormalized adjacency over the low-degree nodes graph (a) and high-degree nodes graph (b).

Similarly, the propagation rule of Graph Attention Networks (GATs) (Veličković et al., 2017) takes a similar form, except that it involves first computing attention coefficients between each node and its neighbors. For node k and its neighbor j, the attention coefficient e_{kj}^i at layer i is computed as follows:

$$e_{kj}^i = LeakyReLU(a^{i\top}[W^i h_k^i || W^i h_j^i]),\qquad(2)$$

where $h_k^i, h_j^i \in \mathbb{R}^{F(i),}$ are the hidden representations of nodes k and j at layer i. $W^i \in \mathbb{R}^{m \times F(i)}$ is the weight matrix at layer i which is used to transform the node features, $a^i \in \mathbb{R}^{2 \times m,}$ is the attention parameter vector used in the shared attention mechanism at layer i and $||$ denotes the concatenation operator. $e_{kj}^i \in \mathbb{R}$ representing the attention coefficient; the attention scores are subsequently obtained by :

$$\alpha_{kj}^i = \frac{exp(e_{kj}^i)}{\sum_{m \in \mathcal{N}(k)} exp(e_{km}^i)}\qquad(3)$$

The aggregating step is based on these scores rather than uniform aggregating as in GCNs, which is formulated in the following equation :

$$h_k^i = \sigma\Big(\sum_{j \in \mathcal{N}(k)} \alpha_{kj}^i \cdot h_j^{i-1}\Big)\qquad(4)$$

Lastly, a Graph-SAGE model operates on a graph by aggregating and updating information from up to k-hop neighborhoods for each node. The model follows a cyclic process for each node in the graph, where it gathers information from the node's k-hop neighborhood, aggregates that information and then updates the node's representation. This process is repeated iteratively to refine the node representations. The aggregation & updating cycle is formulated as follows: for each node v, aggregate the representations from its k-hop neighborhood $\mathcal{N}_k(v)$:

$$h_{\mathcal{N}(v)}^k = AGGREGATE(h_u^{k-1} | d(u,v) = k)\qquad(5)$$

$$h_v^k = \sigma(W^k \cdot CONCAT(h_v^{k-1}, h_{\mathcal{N}(v)}^k))\qquad(6)$$

Where h_v^0 is the initial node feature representation, K is the depth hyperparameter, $k \leq K$ is the iteration step, W^k is a weight matrix for iteration k, $d(u,v)$ represents the shortest path distance between nodes u and v and $AGGREGATE$ is a placeholder for any differentiable function.

The above formulations use a single-weight matrix W^i for each layer i to capture the interactions between the different nodes. In our proposed approach, we aim to extend the standard formulation by incorporating different weight matrices for each group of node degrees.

4.2. Degree-Specific Weight Matrices

In our approach, we propose extending the standard GNNs with the use of different weight matrices for each group of node degrees. The motivation behind this is to account for the variation of the properties between the groups, since nodes with different degrees may contribute differently to the overall task at hand, see Sec. 3.

For each layer in the GNN, we introduce a set of weight matrices, each corresponding to a specific group of node degrees. For maximal simplicity, we set the number of weight matrices to two, accounting for nodes with high and low degrees. In preliminary experiments, even setting one weight matrix for each degree that is present in the graph works well. However, this results in a much larger model that makes an apple-to-apple comparison hard. Yet, only two degree-specific weight matrices can already capture the degree-specialized relationships and enable the model to learn different patterns associated with nodes of different degrees.

In order to incorporate the degree-specific weight matrices, we modify the updating rule of the GNN to consider node degrees during the propagation step. The specific formulation depends on the GNN architecture being used. For instance, for a GCN model, the update rule at the i-th layer can be modified as follows :

$$H_{low}^{i+1} = f(A, H^i, W_{low}^i) = \sigma(D^{-\frac{1}{2}} \cdot A \cdot D^{-\frac{1}{2}} \cdot H^i \cdot W_{low}^i), \tag{7}$$

$$H_{high}^{i+1} = f(A, H^i, W_{high}^i) = \sigma(D^{-\frac{1}{2}} \cdot A \cdot D^{-\frac{1}{2}} \cdot H^i \cdot W_{high}^i), \tag{8}$$

$$H^{i+1}[k,:] = \begin{cases} H_{low}^{i+1}[k,:], & \text{if } Deg[k] \leq \theta \\ H_{high}^{i+1}[k,:], & \text{if } Deg[k] > \theta, \end{cases} \tag{9}$$

where $H_{low}^{i+1} \in \mathbb{R}^{n \times F(i+1)}, H_{high}^{i+1} \in \mathbb{R}^{n \times F(i+1)}$ represents the hidden representation at layer i as computed for low and high degrees nodes, respectively, (it is written this way to simplify notation; in practice, we discard half of these computations) $Deg \in \mathbb{R}^{n \times 1}$ is a vector that represents the degree of the corresponding node, square brackets are used to indicate a selection of the column or element that corresponds to a specific node, and θ is the threshold used to split nodes into low/high degree sets.

Similarly, in a GAT, we modify the attention coefficient calculation in the following way

$$e_{kj}^i = \begin{cases} LeakyReLU(a_{low}^{i\top}[W_{low}^i h_k^i || W_{low}^i h_j^i]), & \text{if } Deg[k] \leq \theta \\ LeakyReLU(a_{high}^{i\top}[W_{high}^i h_k^i || W_{high}^i h_j^i]), & \text{if } Deg[k] > \theta \end{cases} \tag{10}$$

Where $W_{low}^i, W_{high}^i \in \mathbb{R}^{m \times F(i)}$, $a_{low}^{i\top}, a_{high}^{i\top} \in \mathbb{R}^{2 \times m}$ are the weight matrices and attention parameters vectors at layer i, respectively, for both low and high degrees.

Lastly, in graphSAGE we modify the update step in each iteration of the cyclic process of aggregation and updating as follows:

$$h_v^k = \begin{cases} \sigma(W_{low}^k \cdot CONCAT(h_v^{k-1}, h_{\mathcal{N}(v)}^k)), & \text{if } Deg[v] \leq \theta \\ \sigma(W_{high}^k \cdot CONCAT(h_v^{k-1}, h_{\mathcal{N}(v)}^k)), & \text{if } Deg[v] > \theta \end{cases} \tag{11}$$

5. Experiments

In this section, we present the experimental setup, datasets, and the results obtained from evaluating the proposed approach of using different weight matrices for each group of node degrees in different types of Graph Neural Networks (GNNs).

5.1. Datasets

We have evaluated our approach over three commonly used node classification datasets: Cora, Citeseer, and PubMed. These datasets represent citation networks where the nodes correspond to documents and edges represent citation relationships. Each node has associated textual features, and the task is to classify the nodes into predefined categories based on their features and connectivity. The Cora and CiteSeer datasets consist of graphs representing academic papers, where nodes represent papers and the edges represent citation relationships between them, the nodes are assigned to a specific set of academic topics, and the task is to classify each node (paper) into different classes based on their topics. The Cora dataset contains 2708 nodes with a total of 5278 edges, where each node is represented by a 1433 dimensional feature vector; the total number of classes is 7. The CiteSeer dataset contains 3327 total nodes and 4552 edges, with each node represented by a 3703 dimensional feature vector, and the total number of classes in this dataset is 6. The PubMed dataset contains a total of 19717 nodes (scientific publications), which are categorized into 3 classes, with a total of 44338 edges (connections between the publications); each node in the graph is represented by a TF/IDF word vector constructed using a vocabulary of 500 unique words.

5.2. Models

In our experiments, we verify the effectiveness of our proposed approach of learning a different weight matrix per each set of node degrees on several graph-learning-based models commonly used in graph analysis. Specifically, we implemented our method on three graph representation learning models: Graph Convolutional Neural Network (GCN), Graph Attention Network (GAT), and GraphSAGE. By including these three baseline methods, we aim to evaluate the performance of our proposed approach comprehensively and assess its effectiveness.

5.3. Implementation Details

We implemented our proposed approach using the PyTorch library (Paszke et al., 2019). The models were trained using an Adam optimizer with a learning rate of 0.001 and a weight decay of 5e-4. In all of the experiments, we used a two-layer architecture, with a hidden dimension of 32 for the first and second layers. The ReLU activation function was applied after each layer, and a softmax function was used for the final classification layer. The parameter θ is a hyperparameter used to split the nodes into two groups, those of low degree and those of high degree. To determine θ, we first calculate the degree of each node in the graph and create a histogram of node degrees. We then automatically identify a threshold that divides the nodes into low-degree and high-degree groups by analyzing the histogram for a significant gap in the degree distribution. This is done using the conventional

Table 1: Classification accuracy of the baseline GCN and our method over high- and low-degree nodes for different datasets.

Dataset	Variant	Low-Degree Nodes	High-Degree Nodes
Cora	Baseline	0.775	0.809
Cora	Ours	**0.798**	**0.831**
CiteSeer	Baseline	0.612	0.904
CiteSeer	Ours	**0.661**	**0.965**
PubMed	Baseline	0.737	0.775
PubMed	Ours	**0.758**	**0.801**

Table 2: Classification accuracy over the test set of different datasets, results are averaged across 10 different runs, we used the default hyperparameters of the (Guo et al., 2023) method.

Dataset	Variant	GCN	GAT	Graph SAGE
Cora	Baseline	0.7909 ± 0.012	0.7885 ± 00072	0.7256 ± 0.006
Cora	Ours	**0.8035 ± 0.003**	**0.8109 ± 0.004**	**0.7463 ± 0.10**
Cora	Ablation (rand division)	0.7749 ± 0.015	0.7710 ± 0.013	0.7067 ± 0.007
CiteSeer	Baseline	0.6308 ± 0.010	0.6578 ± 0.0054	0.6206 ± 0.004
CiteSeer	Ours	**0.6491 ± 0.009**	**0.6787 ± 0.007**	**0.6431 ± 0.010**
CiteSeer	Ablation (rand division)	0.6231 ± 0.012	0.6452 ± 0.017	0.6182 ± 0.009
PubMed	Baseline	0.7633 ± 0.002	0.7626 ± 0.005	0.7402 ± 0.009
PubMed	Ours	**0.7739 ± 0.005**	**0.7724 ± 0.003**	**0.7512 ± 0.012**
PubMed	Ablation (rand division)	0.7523 ± 0.012	0.7611 ± 0.019	0.7388 ± 0.023

Otsu's method Otsu (1979). In our experiments, we use $\theta = 2, 3, 5$ for Cora, CiteSeer, and PubMed datasets respectively.

5.4. Results

We conducted an initial evaluation to assess the comparative performance of our method against the baseline GCN models for classifying low-degree and high-degree nodes separately. Table 1 presents the results. As can be seen in the results, low-degree nodes are harder to separate, indicating that they lie closer to the decision boundaries. It is also
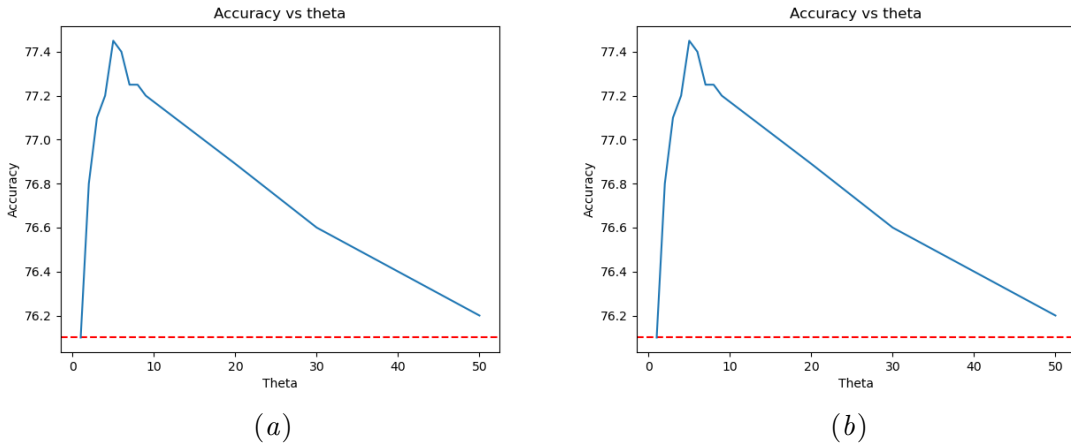
Figure 2: (a) the effect of the parameter θ on the accuracy for our variant of GCN applied to PubMed dataset. The red line is the baseline accuracy and the blue is our result. (b) the distribution of node degrees in the PubMed dataset.

worth mentioning that our method consistently improves all other models across all tested cases.

In Table 2, we present the results of our proposed approach along with the results of the vanilla GNN models from each architecture. From the results, we observe that incorporating different weight matrices based on a node degree threshold leads to improved performance compared to the baseline model (which has one shared weight matrix per layer) across all GNN variants and datasets. This suggests that considering the node degree threshold for weight assignment can capture meaningful variations in the graph structure.

5.5. Ablation study

To show that the added capacity arises from the stratification by node degree and not just from having a second weight matrix per layer, we conduct an ablation study. In this study, we divide the nodes randomly into two groups, whose sizes are identical to what would have been obtained had the division been based on the degree. Furthermore, we conducted an additional random division of nodes into two groups. As can be seen in Table 2, the added capacity, if assigned randomly, (slightly) hurts performance instead of improving it. Additionally, in order to study the effect of different θ values on the performance, we have applied an ablation study over the PubMed dataset, the results can be seen in Fig. 2 .

6. Conclusions

The message-passing mechanism employed by GNNs aggregates the information from multiple neighbors in a way that is agnostic to the local topography of the graph, except for the set of neighbors. In this work, we rely on the simplest node property (the node's degree) and the simplest way to create a topology-dependent treatment of nodes (separation into two groups by thresholding the degree).

With this simple method, we are able to show a very consistent improvement in GNN performance. This is true for both the simplest graph convolutional network and for attention-based variants. Our ablation clearly demonstrates that the improvement in accuracy is not due to the increase in network capacity.

In future work, we would like to rely more heavily on the graph's topology, by encoding the local neighborhood of each node and utilizing this information to condition the message-passing process.

Acknowledgments

References

Aleksandar Bojchevski and Stephan Günnemann. Deep gaussian embedding of graphs: Unsupervised inductive learning via ranking. *ICML*, 2017.

Olivier Chapelle and Alexander Zien. Semi-supervised classification by low density separation. In *Proceedings of the 10th International Workshop on Artifical Intelligence and Statistics*, 2005.

Jie Chen, Shouzhen Chen, Junbin Gao, Zengfeng Huang, Junping Zhang, and Jian Pu. Exploiting neighbor effect: Conv-agnostic gnn framework for graphs with heterophily. *IEEE Transactions on Neural Networks and Learning Systems*, 2023.

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.

Vijay Prakash Dwivedi and Xavier Bresson. A generalization of transformer networks to graphs. *arXiv preprint arXiv:2012.09699*, 2020.

Vijay Prakash Dwivedi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Graph neural networks with learnable structural and positional representations. *arXiv preprint arXiv:2110.07875*, 2021.

Johannes Gasteiger, Stefan Weißenberger, and Stephan Günnemann. Diffusion improves graph learning. *Advances in neural information processing systems*, 32, 2019.

Xiaojun Guo, Yifei Wang, Tianqi Du, and Yisen Wang. Contranorm: A contrastive learning perspective on oversmoothing and beyond. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=SM7XkJouWHm.

Zhijiang Guo, Yan Zhang, and Wei Lu. Attention guided graph convolutional networks for relation extraction. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 241–251, Florence, Italy, July 2019. Association for

Computational Linguistics. doi: 10.18653/v1/P19-1024. URL https://aclanthology.org/P19-1024.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

Ziniu Hu, Yuxiao Dong, Kuansan Wang, and Yizhou Sun. Heterogeneous graph transformer. In *Proceedings of the web conference 2020*, pages 2704–2710, 2020.

Dongkwan Kim and Alice Oh. How to find your friendly neighborhood: Graph attention design with self-supervision. *arXiv preprint arXiv:2204.04879*, 2022.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *ICLR*, 2016.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Nobuyuki Otsu. A threshold selection method from gray-level histograms. *IEEE transactions on systems, man, and cybernetics*, 9(1):62–66, 1979.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

T Konstantin Rusch, Michael M Bronstein, and Siddhartha Mishra. A survey on over-smoothing in graph neural networks. *arXiv preprint arXiv:2303.10993*, 2023.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *ICLR*, 2017.

Tao Wang, Di Jin, Rui Wang, Dongxiao He, and Yuxiao Huang. Powerful graph convolutional networks with adaptive propagation mechanism for homophily and heterophily. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 4210–4218, 2022.

Jun Wu, Jingrui He, and Jiejun Xu. Net: Degree-specific graph neural networks for node and graph classification. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 406–415, 2019.

Xinyi Wu, Zhengdao Chen, William Wei Wang, and Ali Jadbabaie. A non-asymptotic analysis of oversmoothing in graph neural networks. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=CJd-BtnwtXq.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *ICLR*, 2018.

Keyulu Xu, Mozhi Zhang, Jingling Li, Simon S Du, Ken-ichi Kawarabayashi, and Stefanie Jegelka. How neural networks extrapolate: From feedforward to graph neural networks. *arXiv preprint arXiv:2009.11848*, 2020.

Keyulu Xu, Mozhi Zhang, Stefanie Jegelka, and Kenji Kawaguchi. Optimization of graph neural networks: Implicit acceleration by skip connections and more depth. In *International Conference on Machine Learning*, pages 11592–11602. PMLR, 2021.

Mingqi Yang, Yanming Shen, Rui Li, Heng Qi, Qiang Zhang, and Baocai Yin. A new perspective on the effects of spectrum in graph neural networks. In *International Conference on Machine Learning*, 2022.

Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. Do transformers really perform badly for graph representation? *Advances in Neural Information Processing Systems*, 34:28877–28888, 2021.

Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang, and Hyunwoo J Kim. Graph transformer networks. *Advances in neural information processing systems*, 32, 2019.

Yifei Zhang, Hao Zhu, Ziqiao Meng, Piotr Koniusz, and Irwin King. Graph-adaptive rectified linear unit for graph neural networks. In *Proceedings of the ACM Web Conference 2022*, pages 1331–1339, 2022.

Lingxiao Zhao and Leman Akoglu. Pairnorm: Tackling oversmoothing in gnns. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=rkecl1rtwB.

Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI open*, 1:57–81, 2020.