Measuring and Relieving the Over-smoothing Problem for Graph Neural Networks from the Topological View

Deli Chen,[1] **Yankai Lin,**[2] **Wei Li,**[1] **Peng Li,**[2] **Jie Zhou,**[2] **Xu Sun**[1]

[1]MOE Key Lab of Computational Linguistics, School of EECS, Peking University
[2]Pattern Recognition Center, WeChat AI, Tencent Inc., China
{chendeli,liweitj47,xusun}@pku.edu.cn, {yankailin,patrickpli,withtomzhou}@tencent.com,

Abstract

Graph Neural Networks (GNNs) have achieved promising performance on a wide range of graph-based tasks. Despite their success, one severe limitation of GNNs is the over-smoothing issue (indistinguishable representations of nodes in different classes). In this work, we present a systematic and quantitative study on the over-smoothing issue of GNNs. First, we introduce two quantitative metrics, MAD and MADGap, to measure the smoothness and over-smoothness of the graph nodes representations, respectively. Then, we verify that smoothing is the nature of GNNs and the critical factor leading to over-smoothness is the low information-to-noise ratio of the message received by the nodes, which is partially determined by the graph topology. Finally, we propose two methods to alleviate the over-smoothing issue from the topological view: (1) MADReg which adds a MADGap-based regularizer to the training objective; (2) AdaEdge which optimizes the graph topology based on the model predictions. Extensive experiments on 7 widely-used graph datasets with 10 typical GNN models show that the two proposed methods are effective for relieving the over-smoothing issue, thus improving the performance of various GNN models.

Introduction

[1]Graph Neural Networks form an effective framework for learning graph representation, which have proven powerful in various graph-based tasks (Yang, Cohen, and Salakhutdinov 2016; Veličković et al. 2018; Zitnik and Leskovec 2017). Despite their success in graph modeling, over-smoothing is a common issue faced by GNNs (Li, Han, and Wu 2018; Zhou et al. 2018), which means that the representations of the graph nodes of different classes would become indistinguishable when stacking multiple layers, which seriously hurts the model performance (e.g., classification accuracy). However, there is limited study on explaining why and how over-smoothing happens. In this work, we conduct a systematic and quantitative study of the over-smoothing issue of GNNs on 7 widely-used graph datasets with 10 typical GNN models, aiming to reveal what is the crucial factor

---

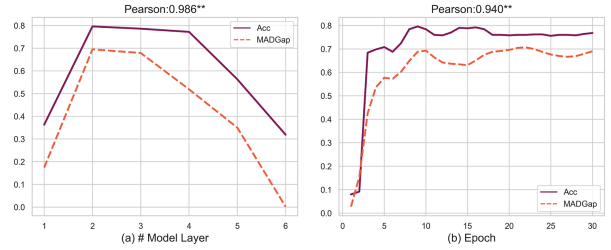[1]Accepted by AAAI 2020. This complete version contains the appendix.

Figure 1: The prediction accuracy (Acc) and MADGap of GCNs (Kipf and Welling 2017) on the *CORA* dataset. We can observe a significantly high correlation between the accuracy and MADGap in two different situations: (a) different models: results of GCNs with different number of layers; (b) different training periods: results after each epoch in the 2-layer GCN. The Pearson correlation coefficient is shown in the title and ** means statistically significant with $p < 0.01$.

bringing in the over-smoothing problem of GNNs and find out a reasonable direction to alleviate it.

We first propose a quantitative metric Mean Average Distance (**MAD**), which calculates the mean average distance among node representations in the graph to measure the smoothness of the graph (**smoothness** means similarity of graph nodes representation in this paper). We observe that the MAD values of various GNNs become smaller as the number of GNN layers increases, which supports the argument that smoothing is the essential nature of GNNs. Hence, the node interaction through the GNN message propagation would make their representations closer, and the whole graph representation would inevitably become smoothing when stacking multiple layers.

Furthermore, we argue that one key factor leading to the over-smoothing issue is the over-mixing of information and noise. The interaction message from other nodes may be either helpful information or harmful noise. For example, in the node classification task, intra-class interaction can bring useful information, while inter-class interaction may lead to indistinguishable representations across classes. To measure the quality of the received message by the nodes, we define the **information-to-noise ratio** as the proportion of

intra-class node pairs in all node pairs that have interactions through GNN model. Based on our hypothesis, we extend MAD to MADGap to measure the over-smoothness of graph (**over-smoothness** means similarity of representations among different classes' nodes in this paper). We notice that two nodes with close topological distance (can reach with a few hops) are more likely to belong to the same class, and vice versa. Therefore, we differentiate the role between remote and neighboring nodes and calculate the gap of MAD values (**MADGap**) between remote and neighboring nodes to estimate the over-smoothness of graph representation. Experimental results prove that MADGap does have a significantly high correlation with the model performance in general situations, and an example is shown in Figure 1. Further experiments show that both the model performance and the MADGap value rise as the information-to-noise ratio increases, which verifies our assumption that the information-to-noise ratio affects the smoothness of graph representation to a great extent.

After more in-depth analysis, we propose that low information-to-noise ratio is caused by the discrepancy between the graph topology and the objective of the downstream task. In the node classification task, if there are too many inter-class edges, the nodes will receive too much message from nodes of other classes after several propagation steps, which would result in over-smoothing. To prove our assumption, we optimize the graph topology by removing inter-class edges and adding intra-class edges based on the gold labels, which proves very effective in relieving over-smoothing and improving model performance. Hence, the graph topology has a great influence on the smoothness of graph representation and model performance. That is to say, there is a deviation from the natural graph to the downstream task. However, in the previous graph-related studies (Veličković et al. 2018; Li et al. 2016; Bianchi et al. 2019), researchers mainly focus on designing novel GNN architectures but pay less attention to improve the established graph topology.

Based on our observations, we propose two methods to relieve the over-smoothing issue from the topological view: (a) **MADReg:** we add a MADGap-based regularizer to the training objective to directly increase received information and reduce noise; (b) Adaptive Edge Optimization (**AdaEdge**[2]): we iteratively train GNN models and conduct edge remove/add operations based on the prediction to adjust the graph adaptively for the learning target. Experimental results show that our two proposed methods can significantly relieve the over-smoothing issue and improve model performance in general cases, which further verifies our conclusions and provides a compelling perspective towards better GNNs performance.

The contributions of this work are threefold:

- We conduct a systematic and quantitative study of the over-smoothing issue on a wide range of graph datasets and models. We propose and verify that a key factor be-

---

[2]The algorithm name is changed from the previous AdaGraph to AdaEdge since the conflicting using of AdaGraph with other work.

Model	Propagate
GCN (Kipf and Welling 2017)	Convolution
ChebGCN (Defferrard, Bresson, and Vandergheynst 2016)	Convolution
HyperGraph (Bai, Zhang, and Torr 2019)	Convolution &Attention
FeaSt (Verma, Boyer, and Verbeek 2018)	Convolution
GraphSAGE (Hamilton, Ying, and Leskovec 2017)	Convolution
GAT (Veličković et al. 2018)	Attention
ARMA (Bianchi et al. 2019)	Convolution
GraphSAGE (Hamilton, Ying, and Leskovec 2017)	Convolution
HighOrder (Morris et al. 2019)	Attention
GGNN (Li et al. 2016)	Gated

Table 1: Introduction of baseline GNN models. The information propagation method is also displayed.

hind the over-smoothing issue is the information-to-noise ratio which is influenced by the graph topology.

- We design two quantitative metrics: MAD for smoothness and MADGap for over-smoothness of graph representation. Statistical analysis shows that MADGap has a significantly high correlation with model performance.

- We propose two methods: MADReg and AdaEdge to relieve the over-smoothing issue of GNNs. Experimental results show that our proposed methods can significantly reduce over-smoothness and improve the performance of multiple GNNs on various datasets.

Datasets and Models

node classification task, one of the most basic graph-based tasks, is usually conducted to verify the effectiveness of GNN architectures (Veličković et al. 2018; Hamilton, Ying, and Leskovec 2017) or analyze the characteristics of GNNs (Li, Han, and Wu 2018; Maehara 2019). Therefore, we select the node classification task for our experiments. We conduct experiments on 7 public datasets in three types, namely, (1) citation network: *CORA*, *CiteSeer*, *PubMed* (Sen et al. 2008); (2) coauthor network: CS, Physics;[3] (3) Amazon product network: Computers, Photo (McAuley et al. 2015). We conduct our detailed analysis on the three citation networks, which are usually taken as the benchmarks for graph-related studies (Li, Han, and Wu 2018; Maehara 2019) and verify the effectiveness of the proposed method on all these datasets.

To guarantee the generalizability of our conclusion, we conduct experiments with 10 typical GNN models in this work. The GNN models and their propagation methods are listed in Table 1, in which the propagation taxonomy follows Zhou et al. (2018). The implementation of the baselines is partly based on Fey and Lenssen (2019) and Maehara (2019). More details about the datasets and experimental settings are given in Appendix A.

Measuring Over-smoothing Problem from the Topological View

In this section, we aim to investigate what is the key factor leading to the over-smoothing problem. To this end, we propose two quantitative metrics MAD and MADGap to mea-

---

[3]https://kddcup2016.azurewebsites.net

sure the smoothness and over-smoothness of graph representation, which are further used to analyze why and how the over-smoothing issue happens.

MAD: Metric for Smoothness

To measure the smoothness of the graph representation, we first propose a quantitative metric: Mean Average Distance (MAD). MAD reflects the smoothness of graph representation by calculating the mean of the average distance from nodes to other nodes. Formally, given the graph representation matrix $H \in \mathbb{R}^{n \times h}$ (we use the hidden representation of the final layer of GNN. Term h is the hidden size), we first obtain the distance matrix $D \in \mathbb{R}^{n \times n}$ for H by computing the cosine distance between each node pair:

$$D_{ij} = 1 - \frac{H_{i,:} \cdot H_{j,:}}{|H_{i,:}| \cdot |H_{j,:}|} \qquad i,j \in [1,2,\cdots,n], \quad (1)$$

where $H_{k,:}$ is the k-th row of H. The reason to use cosine distance is that cosine distance is not affected by the absolute value of the node vector, thus better reflecting the smoothness of graph representation. Then we filter the target node pairs by element-wise multiplication D with a mask matrix M^{tgt}

$$D^{tgt} = D \circ M^{tgt}, \qquad (2)$$

where $\circ$ denotes element-wise multiplication; $M^{tgt} \in \{0,1\}^{n \times n}$; $M_{ij}^{tgt} = 1$ only if node pair (i,j) is the target one. Next we access the average distance $\bar{D}^{tgt}$ for non-zero values along each row in D^{tgt}:

$$\bar{D}_i^{tgt} = \frac{\sum_{j=0}^{n} D_{ij}^{tgt}}{\sum_{j=0}^{n} \mathbb{1}\left(D_{ij}^{tgt}\right)}, \qquad (3)$$

where $\mathbb{1}(x) = 1$ if $x > 0$ otherwise 0. Finally, the MAD value given the target node pairs is calculated by averaging the non-zero values in $\bar{D}^{tgt}$:

$$\text{MAD}^{tgt} = \frac{\sum_{i=0}^{n} \bar{D}_i^{tgt}}{\sum_{i=0}^{n} \mathbb{1}\left(\bar{D}_i^{tgt}\right)}. \qquad (4)$$

Li, Han, and Wu (2018) perform a theoretical analysis on the graph convolution network (GCN), and conclude that performing smoothing operation on node representations is the key mechanism why GCN works. We extend the conclusion empirically to the 10 typical GNNs listed in Table 1 with the help of the proposed MAD metric. To this end, for each GNN model with different number of layers, we compute the MAD value $\text{MAD}^{\text{global}}$ by taking all node pairs into account, i.e., all values in M^{tgt} are 1, to measure the global smoothness of the learned graph representation.

The results on the *CORA* dataset are shown in Figure 2. We can observe that as the number of GNN layers increases, the MAD values become smaller. Apart from this, the MAD value of high-layer GNNs gets close to 0, which means that all the node representations become indistinguishable. GNN models update the node representation based on the features from neighboring nodes. We observe that the interaction between nodes makes their representations similar to each other. Similar phenomenons that the smoothness rises as the layer increases are also observed in other datasets as presented in Appendix B. Therefore, we conclude that smoothing is an essential nature for GNNs.
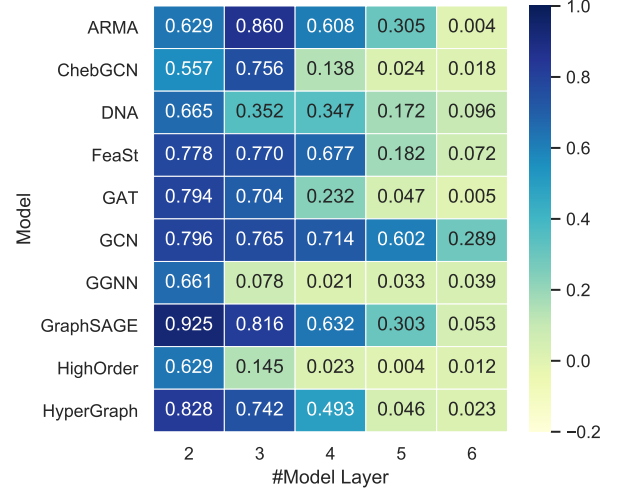

Figure 2: The MAD values of various GNNs with different layers on the *CORA* dataset. Darker color means larger MAD value. We can find that the smoothness of graph representation rises as the model layer increases.

Information-to-noise Ratio Largely Affects Over-smoothness

With the help of MAD, we can quantitatively measure the smoothness of graph representation. Here come two new questions: since smoothing is the nature of GNNs, what is over-smoothing, and what results in over-smoothing?

We assume that the over-smoothing problem is caused by the over-mixing of information and noise, which is influenced by the quality of the nodes received message. The interaction message from other nodes by GNN operation may be either helpful information or interference noise. For example, in the node classification task, interaction between nodes of the same class brings useful information, which makes their representations more similar to each other and the probability of being classified into the same class is increased. On the contrary, the contact of nodes from other classes brings the noise. Hence, the reason why GNNs work is that the received useful information is more than noise. On the other hand, when the noise is more than the information, the learned graph representation will become over-smoothing.

To quantitatively measure the quality of the received message of the nodes, we define the information-to-noise ratio as the proportion of intra-class node pairs in all contactable node pairs that have interactions through the GNN model. For example, at the second-order, the information-to-noise ratio for each node is the proportion of nodes of the same class in all the first-order and second-order neighbors; the information-to-noise ratio for the whole graph is the proportion of the intra-class pairs in all the node pairs that can be contacted in 2 steps. In Figure 3, we display the information-to-noise ratio of the whole graph for the *CORA, CiteSeer* and *Pubmed* datasets. We can find that there are more intra-class node pairs at low order and vice versa. When the model layer number gets large where the information-to-noise ra-
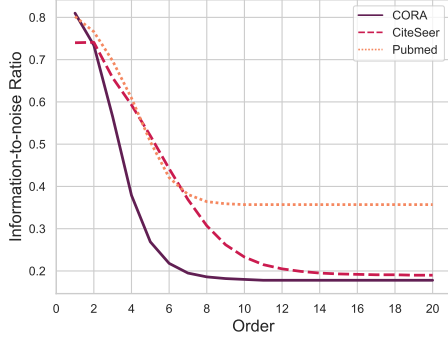
Figure 3: The information-to-noise ratio at different neighbor orders (accumulated) for the *CORA/CiteSeer/PubMed* datasets. We can find that the information-to-noise ratio declines as the orders increases in all these three datasets.

tio is small, the interaction between high-order neighbors brings too much noise and dilutes the useful information, which is the reason for the over-smoothing issue. Based on this observation, we extend MAD to MADGap to measure the over-smoothness in the graph representation. From Figure 3 we notice that two nodes with small topological distance (low-order neighbours) are more likely to belong to the same category. Hence, we propose to utilize the graph topology to approximate the node category, and calculate the gap of MAD values differentiating remote and neighbour nodes to estimate the over-smoothness of the graph representation,

$$\mathrm{MADGap} = \mathrm{MAD}^{\mathrm{rmt}} - \mathrm{MAD}^{\mathrm{neb}}, \qquad (5)$$

where $\mathrm{MAD}^{\mathrm{rmt}}$ is the MAD value of the remote nodes in the graph topology and $\mathrm{MAD}^{\mathrm{neb}}$ is the MAD value of the neighbouring nodes.

According to our assumption, large MADGap value indicates that the useful information received by the node is more than noise. At this time, GNNs perform reasonable extent of smoothing, and the model would perform well. On the contrary, small or negative MADGap means over-smoothing and inferior performance. To verify the effectiveness of MADGap, we calculate the MADGap value[4] and compute the Pearson coefficient between the MADGap and the prediction accuracy for various GNN models. We report the Pearson coefficient for GNNs with different layers on *CORA*, *CiteSeer* and *PubMed* datasets in Table 2. According to the table, we can find that there exists a significantly high correlation between MADGap and the model performance, which validates that MADGap is a reliable metric to measure graph representation over-smoothness. Besides, MADGap can also be used as an observation indicator to estimate the model performance based on the graph topology without seeing the gold label. It is worth noting that 1-layer GNN usually has small MADGap and prediction accuracy (Figure 1), which is caused by the insufficient information

---

[4] In this work, we calculate $\mathrm{MAD}^{\mathrm{neb}}$ based on nodes with orders ≤ 3 and $\mathrm{MAD}^{\mathrm{rmt}}$ based on nodes with orders ≥ 8.

Model	CORA	CiteSeer	PubMed
GCN	0.986**	0.948**	0.971**
ChebGCN	0.945**	0.984**	0.969**
HyperGraph	0.990**	0.965**	0.932**
FeaSt	0.993**	0.986**	0.906*
GraphSAGE	0.965**	0.995*	0.883
GAT	0.960**	0.998**	0.965**
ARMA	0.909*	0.780	0.787
HighOrder	0.986**	0.800	0.999**
DNA	0.945**	0.884*	0.887*
GGNN	0.940*	0.900*	0.998**

Table 2: The Pearson coefficient between accuracy and MADGap for various models on *CORA/CiteSeer/PubMed* datasets. Pearson coefficient is calculated based on the results of models with different layers (1-6). * means statistically significant with $p < 0.05$ and ** means $p < 0.01$.

transfer, while the over-smoothing issue of high-layer GNN is caused by receiving too much noise.

In Figure 4, we show the MADGap and prediction accuracy for node sets with different information-to-noise ratios in the same model. We can find that even with the same model and propagation step, nodes with higher rate of information-to-noise ratio generally have higher prediction accuracy with smaller over-smoothing degree. We also observe similar phenomena on other datasets, which are shown in Appendix C. This way, we further verify that it is the information-to-noise ratio that affects the graph representation over-smoothness to a great extent, thus influencing the model performance.

Topology Affects the Information-to-noise Ratio

From the previous analysis, we can find that the key factor influencing the smoothness of graph representation is the information-to-noise ratio. Then the following question is: what affects the information-to-noise ratio? We argue that it is the graph topology that affects the information-to-noise ratio. The reason for the node receiving too much noise is related to the discordance between the natural graph and the task objective. Take node classification as an example. If there are too many inter-class edges, the nodes will receive too much noise after multiple steps of message propagation, which results in over-smoothing and bad performance.

The graph topology is constructed based on the natural links. For example, the edges in the citation network represent the citing behavior between papers and edges in the product network represent the products co-purchasing relations. GNN models rely on these natural links to learn node representations. However, natural links between nodes of different classes are harmful to the node classification task. Therefore, we propose to alleviate the over-smoothing issue of GNNs and improve their performance by optimizing the graph topology to match the downstream task.

To verify our assumption, we optimize the graph topology by removing inter-class edges and adding intra-class edges based on the gold labels. The results on the *CORA* dataset are shown in Figure 5. We can find that the MADGap value rises consistently as more inter-class edges are removed and
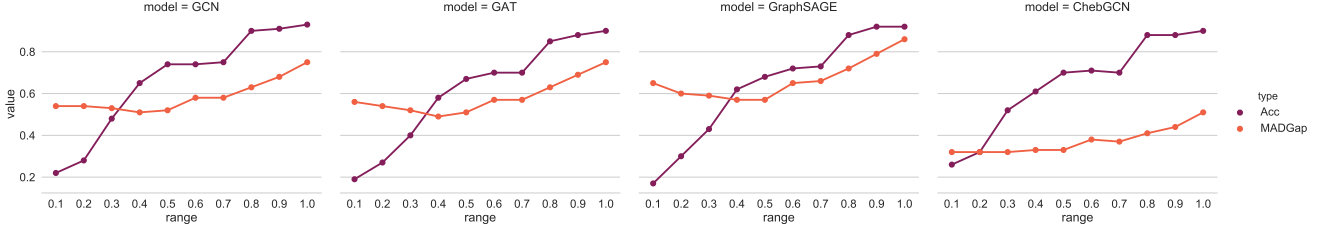
Figure 4: Performance (accuracy) and over-smoothness (MADGap) of node sets with different information-to-noise ratio (e.g., 0.1 means ratio≤ 0.1) on the *CORA* dataset (We display 4 out of 10 models results due to the limited space. We observe similar results in other models). All models have 2 layers. Results prove that nodes with higher information-to-noise ratio would have less over-smoothness degree and better prediction result.
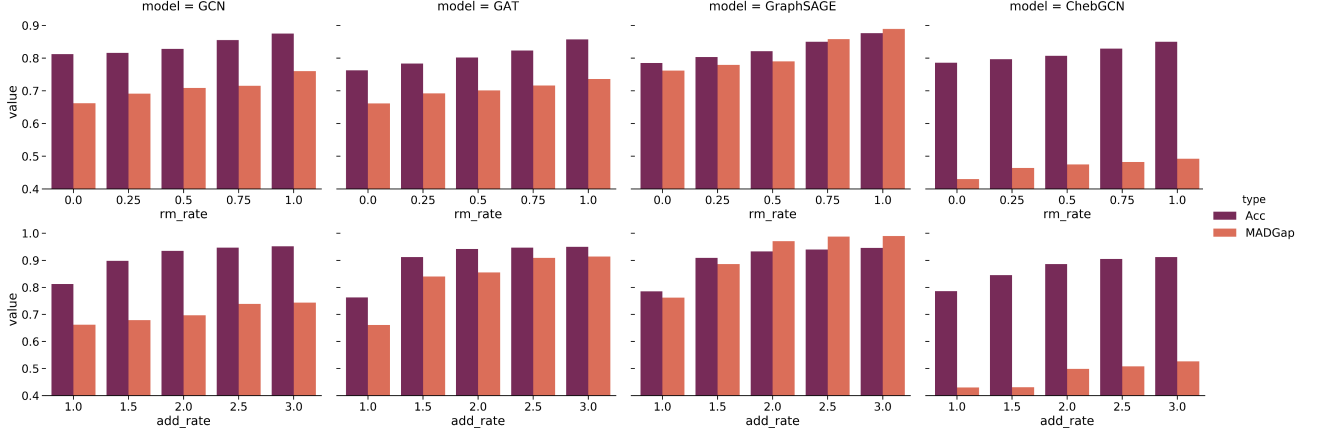

Figure 5: The gold label based topology adjustment experiment on the *CORA* dataset. We show the results of both removing inter-class edges (first row, where the X-axis represents the removing rate) and adding intra-class edges (second row, where the X-axis represents the intra-class edge ratio compared to the raw graph) on GCN, GAT, GraphSAGE and ChebGCN. Results show that both of these methods are very helpful for relieving the over-smoothing issue and improving model performance.

more intra-class edges are added, resulting in better model performance. Therefore, optimizing graph topology is helpful in relieving the over-smoothing problem and improving model performance.

In summary, we find that the graph topology has a great influence on the smoothness of graph representation and model performance. However, there is still discordance between the natural links and the downstream tasks. Most existing works mainly focus on designing novel GNN architectures but pay less attention to the established graph topology. Hence, we further investigate to improve the performance of GNNs by optimizing the graph topology.

Relieving Over-smoothing Problem from the Topological View

Inspired by the previous analysis, we propose two methods to relieve the over-smoothing issue from the topological view: (1) **MADReg:** we add a MADGap-based regularizer to the training objective; (2) Adaptive Edge Optimization (**AdaEdge**): we adjust the graph topology adaptively by iteratively training GNN models and conducting edge remove/add operations based on the prediction result. Neither

of these two methods is restricted to specific model architectures and can be used in the training process of general GNN models. Experiments demonstrate their effectiveness in a variety of GNNs.

MADReg: MADGap as Regularizer

In the previous experiments, we find that MADGap shows a significantly high correlation with model performance. Hence, we add MADGap to the training objective to make the graph nodes receive more useful information and less interference noise:

$$\mathcal{L} = \sum -l \, \log \, p(\hat{l} \,|\, \boldsymbol{X}, \boldsymbol{A}, \Theta) - \lambda \text{MADGap}, \quad (6)$$

where $\boldsymbol{X}$ is the input feature matrix, $\boldsymbol{A}$ is the adjacency matrix, $\hat{l}$ and l are the predicted and gold labels of the node respectively. Θ is the parameters of GNN and λ is the regularization coefficient to control the influence of MADReg. We calculate MADGap on the training set to be consistent with the cross-entropy loss.

AdaEdge: Adaptive Edge Optimization

As discussed in the previous section, after optimizing the topology based on gold label (adding the intra-class edges
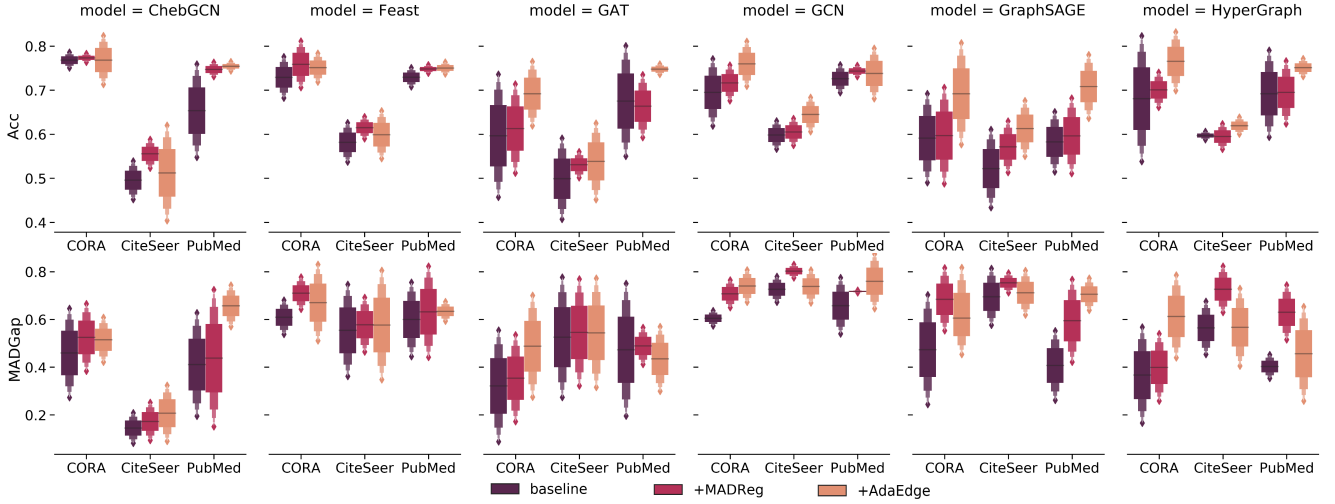
Figure 6: MADReg and AdaEdge results on the *CORA/CiteSeer/PubMed* datasets. The number of GNN layers is 4, where the over-smoothing issue is severe. The box plot shows the mean value and the standard deviation of the prediction accuracy and the MADGap values of 50 turns results (5 dataset splitting methods and 10 random seeds for each splitting following Shchur et al. (2018) and Sun, Koniusz, and Wang (2019). More details can be found in Appendix A). And we can find that the two proposed methods can effectively relieve the over-smoothing issue and improve model performance in most cases.

and removing the inter-class edges), the over-smoothing issue is notably alleviated, and the model performance is greatly improved. Inspired by this, we propose a self-training algorithm called AdaEdge to optimize the graph topology based on the prediction result of the model to adaptively adjust the topology of the graph to make it more reasonable for the specific task objective. Specifically, we first train GNN on the original graph and adjust the graph topology based on the prediction result of the model by deleting inter-class edges and adding intra-class edges. Then we retrain the GNN model on the updated graph from scratch. We perform the above graph topology optimization operation multiple times. The details of the AdaEdge algorithm are introduced in Appendix D.

Relieving Over-smoothing in High-order Layers

To verify the effectiveness of the two proposed methods, we conduct controlled experiments for all the 10 baseline GNN models on *CORA/CiteSeer/PubMed* datasets. We calculate the prediction accuracy and MADGap value for the GNN models with 4 layers, where the over-smoothing issue is serious. The results are shown in Figure 6. We present 6 out of 10 models results due to the space limit; the other models can be found in Appendix E. We can find that in the high-order layer situation where the over-smoothing issue is severe, the MADReg and AdaEdge methods can effectively relieve the over-smoothing issue and improve model performance for most models in all three datasets. The effectiveness of MADReg and AdaEdge further validates our assumption and provides a general and effective solution to relieve the over-smoothing problem.

Improving Performance of GNNs

In Table 3, we show the controlled experiments for GNN models trained on the original graph and the updated graph obtained by the AdaEdge method on all the 7 datasets. We select the best hyper-parameters when training GNN on the original graph and fix all these hyper-parameters when training on the updated graph. Experimental results show that the AdaEdge method can effectively improve the model performance in most cases, which proves that optimizing the graph topology is quite helpful for improving model performance. We analyze the cases of the AdaEdge method with little or no improvement and find that this is caused by the incorrect operations when adjusting the topology. Therefore, when the ratio of incorrect operations is too large, it will bring serious interference to the model training and bring in little or no improvement. Due to the space limit, the results of MADReg are shown in Appendix F. Typically, the baselines achieve their best performance with small number of GNN layers, where the over-smoothing issue is not severe. Under this condition, MADReg can hardly improve the performance by enlarging the MADGap value. However, when the over-smoothing issue becomes more severe while the GNN layer number grows larger, MADReg is still capable of improving the performance of the baselines significantly. Above all, both AdaEdge and MADReg are effective for improving GNNs performance, and AdaEdge generalizes better when the over-smoothing issue is not severe.

Related Work
Graph Neural Networks (GNNs)
GNNs have proven effective in various non-Euclidean graph structures, such as social network (Hamilton, Ying, and Leskovec 2017), biology network (Zitnik and Leskovec

Acc(%)	CORA		CiteSeer		PubMed		Amazon Photo		Amazon Comp.		Coauthor CS		Coauthor Phy.	
Model	baseline	+AE	baseline	+AE	baseline	+AE	baseline	+AE	baseline	+AE	baseline	+AE	baseline	+AE
GCN	$81.2_{\pm0.8}$	$82.3^{**}_{\pm0.8}$	$69.3_{\pm0.7}$	$69.7^{**}_{\pm0.9}$	$76.3_{\pm0.5}$	$77.4^{**}_{\pm0.5}$	$90.6_{\pm0.7}$	$91.5^{**}_{\pm0.5}$	$81.7_{\pm0.7}$	$82.4^{**}_{\pm1.1}$	$89.8_{\pm0.3}$	$90.3^{**}_{\pm0.4}$	$92.8_{\pm1.6}$	$93.0^{**}_{\pm1.1}$
ChebGCN	$78.6_{\pm0.6}$	$80.1^{**}_{\pm0.5}$	$67.4_{\pm1.0}$	$67.8^{*}_{\pm1.2}$	$76.7_{\pm0.1}$	$77.5^{**}_{\pm0.6}$	$89.6_{\pm1.6}$	$89.4_{\pm1.2}$	$80.8_{\pm2.4}$	$81.3^{**}_{\pm1.1}$	$90.5_{\pm0.4}$	$90.7^{*}_{\pm0.3}$	\	\
HyperGraph	$80.5_{\pm0.6}$	$81.4^{**}_{\pm0.8}$	$67.9_{\pm0.5}$	$68.5^{**}_{\pm0.5}$	$77.4_{\pm0.2}$	$77.3_{\pm0.7}$	$87.5_{\pm0.7}$	$88.6^{**}_{\pm0.3}$	$58.7_{\pm22.1}$	$61.4^{**}_{\pm26.8}$	$86.9_{\pm0.5}$	$87.3^{**}_{\pm0.4}$	$91.9_{\pm2.0}$	$92.2^{**}_{\pm1.4}$
FeaSt	$80.4_{\pm0.7}$	$81.6^{**}_{\pm0.7}$	$69.3_{\pm1.1}$	$69.4^{*}_{\pm1.0}$	$76.6_{\pm0.6}$	$77.2^{**}_{\pm0.4}$	$90.5_{\pm0.6}$	$90.8^{**}_{\pm0.6}$	$80.8_{\pm1.3}$	$81.7^{**}_{\pm0.9}$	$88.4_{\pm0.2}$	$88.9^{**}_{\pm0.2}$	\	\
GraphSAGE	$78.5_{\pm1.7}$	$80.2^{**}_{\pm1.2}$	$68.4_{\pm0.9}$	$69.4^{**}_{\pm0.8}$	$75.2_{\pm1.1}$	$77.2^{**}_{\pm0.8}$	$90.1_{\pm1.4}$	$90.6^{**}_{\pm0.5}$	$80.2_{\pm1.0}$	$81.1^{**}_{\pm1.0}$	$90.1_{\pm0.4}$	$90.3^{**}_{\pm0.4}$	$93.0_{\pm0.4}$	$92.7_{\pm0.2}$
GAT	$76.3_{\pm3.1}$	$77.9^{**}_{\pm2.0}$	$68.9_{\pm0.6}$	$69.1^{*}_{\pm0.8}$	$75.9_{\pm0.5}$	$76.6^{**}_{\pm0.2}$	$89.7_{\pm1.7}$	$90.8^{**}_{\pm0.9}$	$81.4_{\pm1.5}$	$81.1^{**}_{\pm1.6}$	$85.5_{\pm1.9}$	$86.6^{**}_{\pm1.6}$	$91.1_{\pm1.0}$	$91.4^{**}_{\pm1.0}$
ARMA	$74.9_{\pm10.6}$	$76.4^{**}_{\pm5.6}$	$65.3_{\pm4.1}$	$66.1^{**}_{\pm4.3}$	$68.5_{\pm11.4}$	$68.9^{**}_{\pm12.2}$	$86.4_{\pm3.0}$	$87.0^{**}_{\pm1.9}$	$63.8_{\pm18.9}$	$71.7^{**}_{\pm8.1}$	$90.6_{\pm1.1}$	$90.9^{**}_{\pm0.6}$	$92.2_{\pm1.8}$	$92.6^{**}_{\pm1.0}$
HighOrder	$76.6_{\pm1.2}$	$72.5^{**}_{\pm4.1}$	$64.2_{\pm1.0}$	$63.3^{**}_{\pm1.0}$	$75.0_{\pm2.6}$	$76.9^{**}_{\pm1.3}$	$26.1_{\pm12.4}$	$30.3^{**}_{\pm10.2}$	$26.3_{\pm12.7}$	$23.9^{**}_{\pm13.4}$	$84.2_{\pm1.0}$	$85.6^{**}_{\pm0.7}$	$90.8_{\pm0.8}$	$90.9_{\pm0.6}$
DNA	$58.2_{\pm14.4}$	$60.1^{*}_{\pm10.8}$	$60.9_{\pm2.7}$	$61.3^{**}_{\pm2.2}$	$65.8_{\pm7.8}$	$66.8^{**}_{\pm9.6}$	$89.1_{\pm1.3}$	$89.8^{**}_{\pm0.6}$	$78.2_{\pm2.9}$	$79.8^{**}_{\pm2.0}$	$88.2_{\pm0.9}$	$90.0^{**}_{\pm0.6}$	$93.0_{\pm0.5}$	$93.3^{**}_{\pm0.4}$
GGNN	$47.3_{\pm6.1}$	$44.7^{**}_{\pm3.5}$	$55.5_{\pm2.8}$	$47.9^{**}_{\pm3.4}$	$66.1_{\pm4.4}$	$69.5^{**}_{\pm1.2}$	$74.1_{\pm12.3}$	$80.6^{**}_{\pm7.2}$	$42.4_{\pm26.7}$	$61.5^{**}_{\pm20.8}$	$86.6_{\pm1.4}$	$88.2^{**}_{\pm0.8}$	$91.2_{\pm1.2}$	$91.6^{**}_{\pm0.7}$

Table 3: Controlled experiments of AdaEdge (+AE) on all the 7 datasets. We show the mean value, the standard deviation and the t-test significance of 50 turns results. * means statistically significance with $p < 0.05$ and ** means $p < 0.01$. Darker color means larger improvement. The missing results are due to the huge consumption of GPU memory of large graphs.

2017), business graph (McAuley et al. 2015) and academic graph (Sen et al. 2008). Recently, many novel GNN architectures have been developed for graph-based tasks. Veličković et al. (2018) propose the graph attention network to use self-attention to aggregate information from neighboring nodes. Hamilton, Ying, and Leskovec (2017) propose a general inductive framework to generate node embedding by sampling and aggregating features from the neighboring nodes. There are also other GNNs proposed, such as ARMA (Bianchi et al. 2019), FeaSt (Verma, Boyer, and Verbeek 2018), HyperGraph (Bai, Zhang, and Torr 2019) and so on. Xu et al. (2018) propose jumping knowledge networks to help the GNN model to leverage the information from high-order neighbours for a better node representation. However, all these models focus on improving the information propagation and aggregation operation on the static graph while paying less attention to the graph topology. In this work, we propose to explicitly optimize the graph topology to make it more suitable for the downstream task.

Pareja et al. (2019) propose the EvolveGCN that uses the RNN to evolve the graph model itself over time. Fey (2019) allow for a selective and node-adaptive aggregation of the neighboring embeddings of potentially differing locality. Yang et al. (2019b) propose a new variation of GCN by jointly refining the topology and training the fully connected network. These existing works about dynamic graph rely on the adaptive ability of the model itself and focus on special GNN architecture (e.g., GCN), while our AdaEdge method optimizes the graph topology with a clear target (adding intra-class edges and removing inter-class edges) and can be used in general GNN architectures. Rong et al. (2019) propose DropEdge method to drop edges randomly at each training epoch for data augmentation while our AdaEdge method adjusts edges before training to optimize the graph topology.

Over-smoothing Problem in GNNs

Previous works (Li, Han, and Wu 2018; Zhou et al. 2018) have proven that over-smoothing is a common phenomenon in GNNs. Li, Han, and Wu (2018) prove that the graph convolution of the GCN model is actually a special form of Laplacian smoothing. Deng, Dong, and Zhu (2019) propose that smoothness is helpful for node classification and design methods to encourage the smoothness of the output distribution, while Yang et al. (2019a) propose that nodes may be mis-classified by topology based attribute smoothing and try to overcome this issue. In this work, we prove that smoothing is the essential feature of GNNs, and then classify the smoothing into two kinds by the information-to-noise ratio: reasonable smoothing that makes GNN work, and over-smoothing that causes the bad performance. From this view, the methods from Deng, Dong, and Zhu (2019) and Yang et al. (2019a) can be regarded as improving reasonable smoothing and relieve over-smoothing, respectively. Besides, Li et al. (2019a) propose to use LSTM in GNN to solve over-smoothing issue in text classification. However, existing works usually mention the over-smoothing phenomenon, but there lacks systematic or quantitative research about it.

Conclusion and Future Work

In this work, we conduct a systematic and quantitative study of the over-smoothing issue faced by GNNs. We first design two quantitative metrics: MAD for smoothness and MADGap for over-smoothness. From the quantitative measurement results on multiple GNNs and graph datasets, we find that smoothing is the essential nature of GNNs; over-smoothness is caused by the over-mixing of information and the noise. Furthermore, we find that there is a significantly high correlation between the MADGap and the model performance. Besides, we prove that the information-to-noise ratio is related to the graph topology, and we can relieve the over-smoothing issue by optimizing the graph topology to make it more suitable for downstream tasks. Followingly, we propose two methods to relieve the over-smoothing issue in GNNs: the MADReg and the AdaEdge methods. Extensive results prove that these two methods can effectively relieve the over-smoothing problem and improve model performance in general situations.

Although we have shown optimizing graph topology is an effective way of improving GNNs performance, our proposed AdaEdge method still suffers from the wrong graph adjustment operation problem. How to reduce these operations is a promising research direction.

Acknowledgement

References

Bai, S.; Zhang, F.; and Torr, P. H. 2019. Hypergraph Convolution and Hypergraph Attention. *arXiv preprint arXiv:1901.08150*.

Bianchi, F. M.; Grattarola, D.; Livi, L.; and Alippi, C. 2019. Graph Neural Networks with Convolutional Arma Filters. *arXiv preprint arXiv:1901.01343*.

Defferrard, M.; Bresson, X.; and Vandergheynst, P. 2016. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. In *Advances in Neural Information Processing Systems*, 3837–3845.

Deng, Z.; Dong, Y.; and Zhu, J. 2019. Batch virtual adversarial training for graph convolutional networks. *arXiv preprint arXiv:1902.09192*.

Fey, M., and Lenssen, J. E. 2019. Fast Graph Representation Learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*.

Fey, M. 2019. Just Jump: Dynamic Neighborhood Aggregation in Graph Neural Networks. *arXiv preprint arXiv:1904.04849*.

Hamilton, W. L.; Ying, Z.; and Leskovec, J. 2017. Inductive Representation Learning on Large Graphs. In *Advances in Neural Information Processing Systems*, 1024–1034.

Kipf, T. N., and Welling, M. 2017. Semi-supervised Classification with Graph Convolutional Networks. In *5th International Conference on Learning Representations, ICLR 2017*.

Li, Y.; Tarlow, D.; Brockschmidt, M.; and Zemel, R. S. 2016. Gated Graph Sequence Neural Networks. In *4th International Conference on Learning Representations, ICLR 2016*.

Li, W.; Li, S.; Ma, S.; He, Y.; Chen, D.; and Sun, X. 2019a. Recursive graphical neural networks for text classification. *arXiv preprint arXiv:1909.08166*.

Li, W.; Xu, J.; He, Y.; Yan, S.; Wu, Y.; and Sun, X. 2019b. Coherent Comment Generation for Chinese Articles with a Graph-to-Sequence Model. In *Proceedings of the 57th Conference of the Association for Computational Linguistics*, 4843–4852.

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

Maehara, T. 2019. Revisiting graph neural networks: All we have is low-pass filters. *arXiv preprint arXiv:1905.09550*.

McAuley, J.; Targett, C.; Shi, Q.; and Van Den Hengel, A. 2015. Image-based Recommendations on Styles and Substitutes. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 43–52. ACM.

Morris, C.; Ritzert, M.; Fey, M.; Hamilton, W. L.; Lenssen, J. E.; Rattan, G.; and Grohe, M. 2019. Weisfeiler and Leman go Neural: Higher-order Graph Neural Networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, 4602–4609.

Pareja, A.; Domeniconi, G.; Chen, J.; Ma, T.; Suzumura, T.; Kanezashi, H.; Kaler, T.; and Leisersen, C. E. 2019. Evolvegcn: Evolving graph convolutional networks for dynamic graphs. *arXiv preprint arXiv:1902.10191*.

Rong, Y.; Huang, W.; Xu, T.; and Huang, J. 2019. The truly deep graph convolutional networks for node classification. *arXiv preprint arXiv:1907.10903*.

Sen, P.; Namata, G.; Bilgic, M.; Getoor, L.; Galligher, B.; and Eliassi-Rad, T. 2008. Collective classification in network data. *AI magazine* 29(3):93–93.

Shchur, O.; Mumme, M.; Bojchevski, A.; and Günnemann, S. 2018. Pitfalls of Graph Neural Network Evaluation. *arXiv preprint arXiv:1811.05868*.

Sun, K.; Koniusz, P.; and Wang, J. 2019. Fisher-Bures Adversary Graph Convolutional Networks. In *Proceedings of the Thirty-Fifth Conference on Uncertainty in Artificial Intelligence*, 161.

Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Lio, P.; and Bengio, Y. 2018. Graph Attention Networks. In *6th International Conference on Learning Representations, ICLR 2018*.

Verma, N.; Boyer, E.; and Verbeek, J. 2018. Feastnet: Feature-steered Graph Convolutions for 3d Shape Analysis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2598–2606.

Xu, K.; Li, C.; Tian, Y.; Sonobe, T.; Kawarabayashi, K.; and Jegelka, S. 2018. Representation learning on graphs with jumping knowledge networks. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018*, 5449–5458.

Yang, L.; Chen, Z.; Gu, J.; and Guo, Y. 2019a. Dual Self-Paced Graph Convolutional Network: Towards Reducing Attribute Distortions Induced by Topology. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019*, 4062–4069.

Yang, L.; Kang, Z.; Cao, X.; Jin, D.; Yang, B.; and Guo, Y. 2019b. Topology Optimization based Graph Convolutional Network. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019*, 4054–4061.

Yang, Z.; Cohen, W. W.; and Salakhutdinov, R. 2016. Revisiting Semi-supervised Learning with Graph Embeddings. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016*, 40–48.

Zhou, J.; Cui, G.; Zhang, Z.; Yang, C.; Liu, Z.; and Sun, M. 2018. Graph Neural Networks: A Review of Methods and Applications. *arXiv preprint arXiv:1812.08434*.

Zitnik, M., and Leskovec, J. 2017. Predicting Multicellular Function through Multi-layer Tissue Networks. *Bioinformatics* 33(14):i190–i198.

A Experimental Settings

In this section, we will introduce the graph task datasets and the baseline GNN models considered in this work first. And then we will introduce the experiment details and notations used in this paper.

	Class	Feature	Node	Edge
CORA	7	1,433	2,708	5,278
CiteSeer	6	3,703	4,732	13,703
PubMed	3	500	44,338	500
Coauthor CS	15	6,805	18,333	81,894
Coauthor Phy.	5	8,415	34,493	247,962
Amazon Comp.	10	767	13,381	245,778
Amazon Photo	8	745	7,487	119,043

Table 4: Statistical information about datasets.

Experiment Dataset

We conduct experiment on 7 node classification datasets in 3 types:

- **Citation Network** The *CORA*, *CiteSeer*, *PubMed* datasets (Sen et al. 2008) are citation networks which are usually taken as be benchmarks of graph related studies (Li, Han, and Wu 2018; Maehara 2019). The nodes are papers in computer science field with bag-of-word features of paper title. The edges represent the citation relation amaong papers and the label is paper category.

- **Coauthor Network** Coauthor CS and Coauthor Physics are co-authorship graphs based on the Microsoft Academic Graph from the KDD Cup 2016 challenge 3.[5] Here, nodes are authors, that are connected by an edge if they co-authored a paper; node features represent paper keywords for each authors papers, and class labels indicate most active fields of study for each author.

- **Amazon Network** Amazon Computers and Amazon Photo (McAuley et al. 2015) are segments of the Amazon co-purchase graph, where nodes represent goods, edges indicate that two goods are frequently bought together, node features are bag-of-words encoded product reviews, and class labels are given by the product category.

Datasets related statistical information are shown in the Table 4. All these graphs are undirected graphs with no edge weight.

Experimental Details

Previous works (Veličković et al. 2018; Bai, Zhang, and Torr 2019; Li et al. 2019b) on GNN study usually run experiment multi-turns to eliminate random effects. And Shchur et al. (2018) pointed out that in the semi-supervised node classification task, the train/valid/test split of dataset has a significant influence on the final result. Following Shchur et al.; Sun, Koniusz, and Wang (2018; 2019) we apply the 20/30/rest splitting method, which means we randomly sample 20 nodes in each category for training set and 30 for validation set; all the rest nodes are as test data. In order to ensure the credibility of the results, we select 5 random train/valid/test split of dataset and run 10 turns with different random seeds in each split. Then we measure the average number of all the 50 turns' results.

Besides, in order to avoid the random effects caused by dataset split or initial seeds and observe the influence of the proposed methods more clearly, we use the same random dataset split and initial seed list for the baseline method and the proposed method in each controlled experiment, and the dataset split and seed list was randomly generated before each controlled experiment. We also fix all the other hyperparameters in each controlled experiment.

Notations Used in This Paper

Given a undirected graph $G = (V, E)$, where the V is the node set with $|V| = n$ and E is the edge set. The adjacency matrix is denoted by $A = [a_{ij}] \in \mathbb{R}^{n \times n}$. The raw node features are denoted by $X = [x_1, x_2, \ldots, x_n]^\top \in \mathbb{R}^{n \times c}$ and each $x_i \in \mathbb{R}^c$ is the input feature vector for the i-th node. Each node has a corresponding label l indicating the node class. In the semi-supervised node classification task, the gold labels of nodes in the trainset are known and the target is to predict the labels for nodes in the testset. We use GNN_k to represent a k-layer (propagate step) graph neural network and the label predicted by this GNN is represented by $\hat{l}$. Besides, the hidden states of all node after j layer(s) is denoted by $H_j \in \mathbb{R}^{n \times h_j}$ (h_j represents the hidden-size of GNN j-th layer) and h_{ji} denotes the hidden state vector of the i-th node representation after j layers.

B MAD Global Values on More Datasets

In Figure 9, we display the MAD values of various GNNs with different layers on *CiteSeer* and *PubMed* dataset. We can observe that in these two datasets, the MAD values of all baseline GNN models decreases with the increase of model layer.

C Infomation-to-noise Ratio Experiment on More Datasets

In Figure 7, we display the model performance and the MADGap value of node sets with different infomation-to-noise ratio at 2-order neighbours in *CiteSeer* and *PubMed* dataset. We can find that in these four GNN models, the model performance and the MADGap value rises with the increase of the intra-class node ratio, which will bring more useful information for the graph nodes. So it is the information-to-noise ratio that largely affects the node representation over-smoothness thus has a influence on the model performance.

D AdaEdge Algorithm

The details of AdaEdge Algorithm is displayed in Algorithm 1. Besides, we use several heuristic approaches to adjust graph topology more wisely: (1) Operation edge for nodes with high prediction confidence (the max value after softmax operation); (2) Operation edge for nodes belong to classes with high prediction precision; (3) Skip operation by a certain probability to control the sparsity of the added and removed edges; (4) Operation edge for nodes with certain degrees.

---

[5]https://kddcup2016.azurewebsites.net

LAYER	Metric	CORA		CiteSeer		PubMed	
		GCN	+MADReg	GCN	+MADReg	GCN	+MADReg
Layer1	acc(%)	$54.6_{\pm14}$	$54.9^{*}_{\pm13}$	$39.7_{\pm12}$	$41.6^{**}_{\pm12}$	$52.8_{\pm20}$	$57.1^{**}_{\pm16}$
	MADGap	$0.13_{\pm0.04}$	$0.14^{*}_{\pm0.04}$	$0.07_{\pm0.04}$	$0.10^{**}_{\pm0.03}$	$0.33_{\pm0.20}$	$0.44^{**}_{\pm0.21}$
Layer2	acc(%)	$80.8_{\pm0.8}$	$80.6_{\pm0.88}$	$68.7_{\pm0.96}$	$67.1^{**}_{\pm1.7}$	$76.6_{\pm0.66}$	$75.2^{**}_{\pm1.2}$
	MADGap	$0.59_{\pm0.03}$	$0.66^{**}_{\pm0.04}$	$0.51_{\pm0.08}$	$0.53^{*}_{\pm0.07}$	$0.74_{\pm0.03}$	$0.73_{\pm0.05}$
Layer3	acc(%)	$79.0_{\pm1.2}$	$79.9^{**}_{\pm1.1}$	$65.9_{\pm1.6}$	$65.4^{*}_{\pm2.2}$	$76.2_{\pm0.65}$	$76.5^{*}_{\pm0.64}$
	MADGap	$0.57_{\pm0.03}$	$0.59^{*}_{\pm0.04}$	$0.52_{\pm0.04}$	$0.53^{*}_{\pm0.03}$	$0.70_{\pm0.04}$	$0.74^{**}_{\pm0.07}$
Layer4	acc(%)	$70.8_{\pm12}$	$72.1^{**}_{\pm8.2}$	$55.6_{\pm11}$	$62.2^{**}_{\pm1.9}$	$67.3_{\pm11}$	$69.4^{**}_{\pm9.9}$
	MADGap	$0.52_{\pm0.05}$	$0.53^{*}_{\pm0.04}$	$0.47_{\pm0.07}$	$0.54^{**}_{\pm0.04}$	$0.63_{\pm0.22}$	$0.71^{**}_{\pm0.13}$
Layer5	acc(%)	$46.9_{\pm9.7}$	$48.0^{*}_{\pm11}$	$35.8_{\pm10}$	$46.5^{**}_{\pm7.7}$	$55.1_{\pm9.4}$	$56.1^{**}_{\pm9.6}$
	MADGap	$0.24_{\pm0.07}$	$0.28^{*}_{\pm0.04}$	$0.331_{\pm0.18}$	$0.46^{**}_{\pm0.07}$	$0.45_{\pm0.22}$	$0.51^{*}_{\pm0.24}$
Layer6	acc(%)	$36.2_{\pm9.1}$	$38.5^{**}_{\pm8.2}$	$28.9_{\pm9.0}$	$37.5^{**}_{\pm4.7}$	$49.5_{\pm9.1}$	$52.3^{*}_{\pm10.5}$
	MADGap	$-0.22_{\pm0.20}$	$-0.20^{**}_{\pm0.16}$	$0.18_{\pm0.23}$	$0.36^{**}_{\pm0.07}$	$0.12_{\pm0.25}$	$0.18^{*}_{\pm0.24}$

Table 5: Results of MADReg method for GCN with different layers on *CORA/CiteSeer/PubMed* dataset. Darker color means larger improvement over the baseline.

E Supplementary Result of Relieving Over-smoothing

In Figure 8, we display more models results of relieving over-smoothing in high layer GNNs.

F Error Analysis of MADReg

In Table 5, we show the results for GCN with different number of layers on the *CORA/CiteSeer/PubMed* datasets and we observe that the MADReg method can relieve the over-smoothing problem and improve model performance effectively especially for GNNs with more layers where the over-smoothing issue is more severe. In the shallow GCN, our MADReg method can not effectively improve the model performance because the model has already been well trained and the over-smoothing issue is not serious.
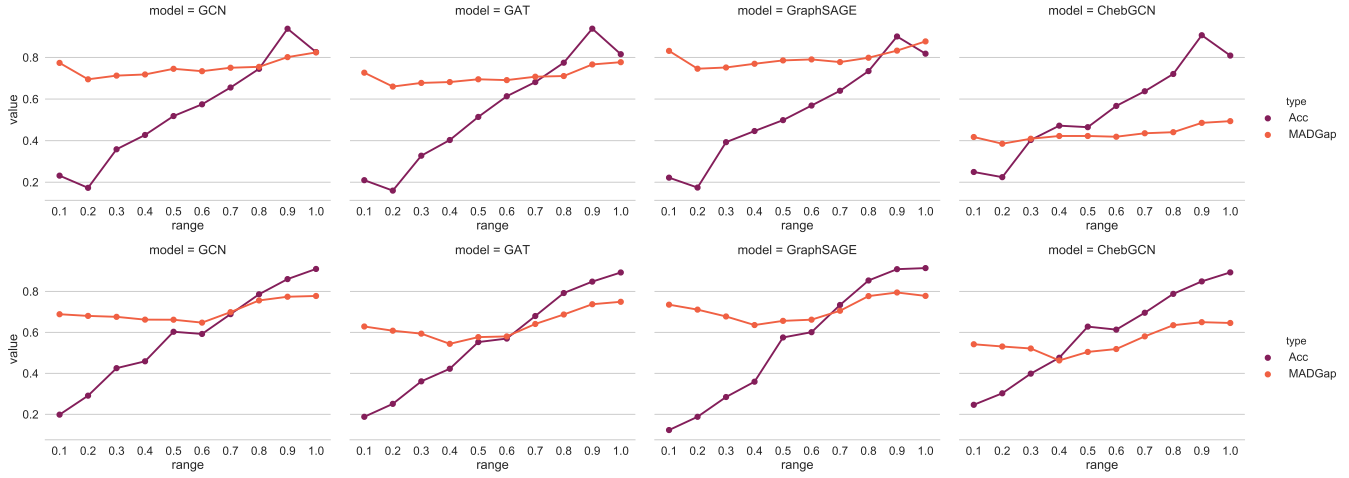
Figure 7: Infomation-to-noise Ratio Experiment on *CiteSeer* (top plot) and *PubMed* (bottom plot) datasets.
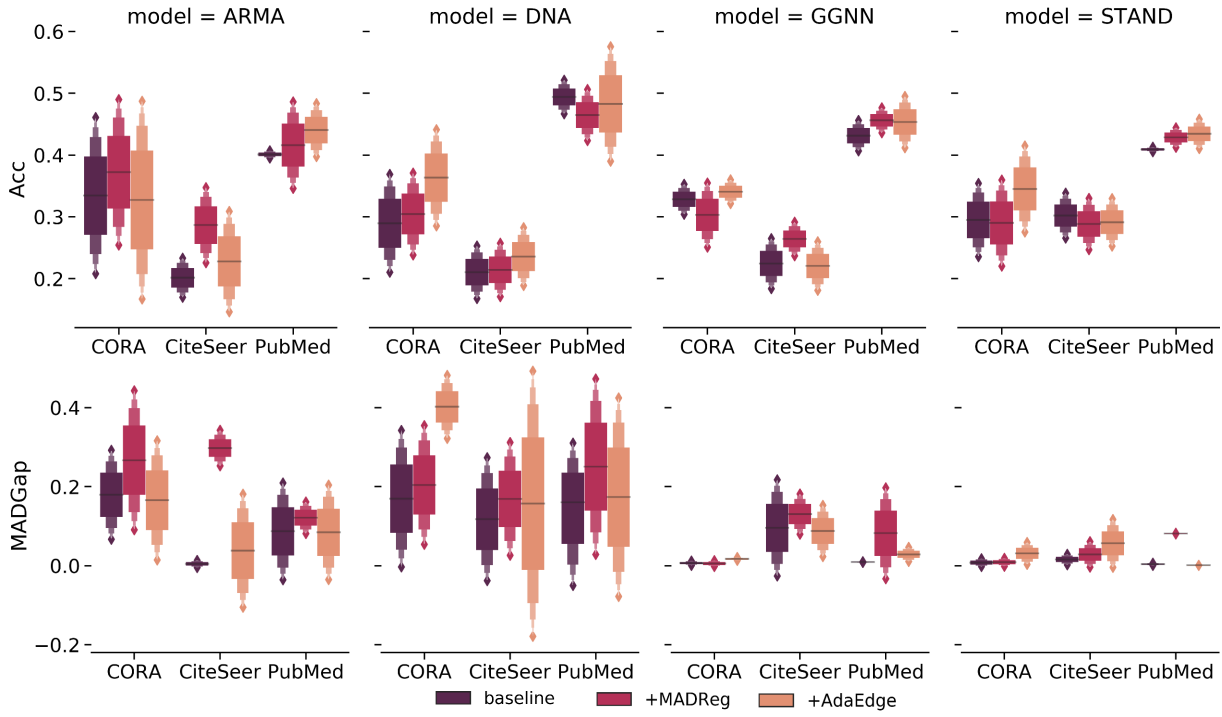
Figure 8: More Results on the *CORA/CiteSeer/PubMed* datasets. The number of model layer is 4, where the over-smoothing issue is serious. The box graph shows the mean value and the standard deviation of the prediction accuracy and the MADGap values. And we can find that the two proposed methods can effectively relieve the over-smoothing issue and improve model performance in most cases.

CiteSeer (top plot)

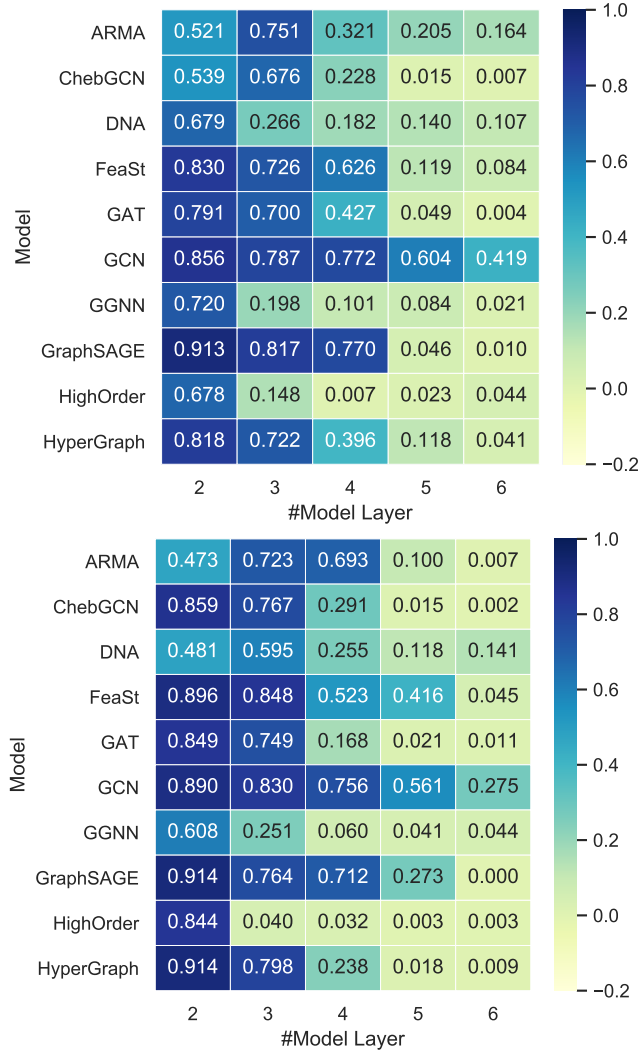| Model	2	3	4	5	6
ARMA	0.521	0.751	0.321	0.205	0.164
ChebGCN	0.539	0.676	0.228	0.015	0.007
DNA	0.679	0.266	0.182	0.140	0.107
FeaSt	0.830	0.726	0.626	0.119	0.084
GAT	0.791	0.700	0.427	0.049	0.004
GCN	0.856	0.787	0.772	0.604	0.419
GGNN	0.720	0.198	0.101	0.084	0.021
GraphSAGE	0.913	0.817	0.770	0.046	0.010
HighOrder	0.678	0.148	0.007	0.023	0.044
HyperGraph	0.818	0.722	0.396	0.118	0.041

#Model Layer

PubMed (bottom plot)

Model	2	3	4	5	6
ARMA	0.473	0.723	0.693	0.100	0.007
ChebGCN	0.859	0.767	0.291	0.015	0.002
DNA	0.481	0.595	0.255	0.118	0.141
FeaSt	0.896	0.848	0.523	0.416	0.045
GAT	0.849	0.749	0.168	0.021	0.011
GCN	0.890	0.830	0.756	0.561	0.275
GGNN	0.608	0.251	0.060	0.041	0.044
GraphSAGE	0.914	0.764	0.712	0.273	0.000
HighOrder	0.844	0.040	0.032	0.003	0.003
HyperGraph	0.914	0.798	0.238	0.018	0.009

#Model Layer

Figure 9: MAD results on *CiteSeer* (top plot) and *PubMed* (bottom plot) datasets.

Algorithm 1 AdaEdge

Require: The GNN model $\mathbf{GNN_k}$, the feature matrix $\boldsymbol{X}$, the raw adjacency matrix $\boldsymbol{A}$, the node size N, the operation order $order$, the limit number num_+ and num_-, the limit confidence $conf_+$ and $conf_-$ and max training times max_t.

1: **function** ADDEDGE($\mathbf{A}, \mathbf{pred}, \mathbf{conf}$)
2:      add_count $\leftarrow 0$
3:      $\mathbf{A}' \leftarrow \mathbf{A}$
4:      **for** node1 $n_i \in [0, N)$ **do**
5:          **for** node2 $n_j \in [n_i, N)$ **do**
6:              **if** $\mathbf{A}_{ij}$==0 and $\mathbf{pred}[n_i]$==$\mathbf{pred}[n_j]$ and $\mathbf{conf}[n_i] \geq conf_+$ and $\mathbf{conf}[n_j] \geq conf_+$ **then**
7:                  $\mathbf{A}'_{ij} \leftarrow 1, \mathbf{A}'_{ji} \leftarrow 1$
8:                  add_count $\leftarrow$ add_count + 1
9:                  **if** add_count $\geq num_+$ **then**
10:                      reutrn $\mathbf{A}'$
11:                  **end if**
12:              **end if**
13:          **end for**
14:      **end for**
15:      reutrn $\mathbf{A}'$
16: **end function**
17:
18: **function** REMOVEDGE($\mathbf{A}, \mathbf{pred}, \mathbf{conf}$)
19:      rmv_count $\leftarrow 0$
20:      $\mathbf{A}' \leftarrow \mathbf{A}$
21:      **for** edge $(n_i, n_j) \in \mathbf{A}$ **do**
22:          **if** $\mathbf{A}_{ij}$==1 and $\mathbf{pred}[n_i]$!=$\mathbf{pred}[n_j]$ and $\mathbf{conf}[n_i] \geq conf_-$ and $\mathbf{conf}[n_j] \geq conf_-$ **then**
23:              $\mathbf{A}'_{ij} \leftarrow 0, \mathbf{A}'_{ji} \leftarrow 0$
24:              rmv_count $\leftarrow$ rmv_count + 1
25:              **if** rmv_count $\geq num_-$ **then**
26:                  reutrn $\mathbf{A}'$
27:              **end if**
28:          **end if**
29:      **end for**
30:      reutrn $\mathbf{A}'$
31: **end function**
32:
33: **function** ADJUSTGRAPH($\mathbf{pred}, \mathbf{conf}$)
34:      **if** $order$=='add_first' **then**
35:          $\mathbf{A}' \leftarrow$ ADDEDGE($\boldsymbol{A}, \mathbf{pred}, \mathbf{conf}$)
36:          $\mathbf{A}' \leftarrow$ REMOVEEDGE($\mathbf{A}', \mathbf{pred}, \mathbf{conf}$)
37:      **else**
38:          $\mathbf{A}' \leftarrow$ REMOVEEDGE($\boldsymbol{A}, \mathbf{pred}, \mathbf{conf}$)
39:          $\mathbf{A}' \leftarrow$ ADDEDGE($\mathbf{A}', \mathbf{pred}, \mathbf{conf}$)
40:      **end if**
41:      reutrn $\mathbf{A}'$
42: **end function**
43:
44: **AdaEdge**
45: $\mathrm{acc_0}, \mathbf{pred_0}, \mathbf{conf_0} \leftarrow \mathbf{GNN_k^0}(\boldsymbol{A}, \boldsymbol{X})$
46: $\mathbf{A}_0' \leftarrow$ ADJUSTGRAPH($\mathbf{pred_0}, \mathbf{conf_0}$)
47: **for** iter times $i \in [1, max_t)$ **do**
48:      $\mathrm{acc_i}, \mathbf{pred_i}, \mathbf{conf_i} \leftarrow \mathbf{GNN_k^i}(\mathbf{A}_{i-1}', \boldsymbol{X})$
49:      **if** $\mathrm{acc_i} \leq \mathrm{acc_{i-1}}$ **then**
50:          reutrn $\mathrm{acc_{i-1}}$
51:      **end if**
52:      $\mathbf{A}_i' \leftarrow$ ADJUSTGRAPH($\mathbf{pred_i}, \mathbf{conf_i}$)
53: **end for**
54: reutrn $\mathrm{acc_{max_t - 1}}$