FINAL PROJECT REPORT
OVERCOME OVER-SMOOTHING IN GRAPH NEURAL NETWORKS

Raphaël Romand-Ferroni
CentraleSupélec
raphael.romandferroni@student-cs.fr

Alexandre Selvestrel
CentraleSupélec
alexandre.selvestrel@student-cs.fr

ABSTRACT

This project aims to evaluate oversmoothing in graph neural networks (GNNs), focusing specifically on two methods introduced by Chen et al. (2019): AdaEdge and MADReg. Our goal is to help identify the most effective approach to limiting oversmoothing in GNNs. By improving our understanding of the impact of oversmoothing on these networks, our work could help improve predictions for existing architectures and guide the design of new GNN architectures. This could eventually pave the way for deeper networks capable of taking into account interactions between more distant nodes, thereby increasing representational power. The link to the GitHub repository can be find here: https://github.com/LaFerraille/Overcome-Over-Smoothing-in-Graph-Neural-Networks

1 INTRODUCTION

Over-smoothing is a phenomenon observed in graph neural networks (GNNs) where the network produces nearly identical outputs for all nodes. This issue arises because, as information from each node is averaged with that of its neighbors across successive layers, the distinct features of nodes begin to converge towards uniform values. This uniformity makes it difficult to distinguish between nodes, undermining their classification. The underlying cause of over-smoothing is the iterative averaging process inherent to each layer's operation. A critical balance must be maintained between enhancing the network's representational power—achieved as the network deepens and facilitates interaction among more distant nodes—and mitigating over-smoothing, which tends to increase with network depth due to the progressive homogenization of node features.

This project assesses over-smoothing in GNNs, and evaluates two methods to mitigate it: AdaEdge and MADReg. We also investigate how combining them together affects the outcome and how the depth of the network can change the results. These methods were originally proposed by Chen et al. (2019) and only tested with four layers GNN. We verify the reproducibility of their results and we asses how they change when we modify the number of layers. Our work extends the research conducted by Rusch et al. (2023) to quantify over-smoothing. As in the survey, we use the Dirichlet energy to measure oversmoothing, but also use MadGap because it is linked to the penalization of MadReg.

In our study, we are particularly interested in Graph Attention Networks (GATs) (Veličković et al., 2018), but also study more traditional Graph Convolutional Networks (GCNs) (Kipf & Welling, 2017), GraphSAGE (Hamilton et al., 2018) and eventually ChebNET model (Tang et al., 2023). GATs are a form of artificial neural network architecture tailored for graph-structured data, delivering state-of-the-art results in both inductive and transductive classification tasks. Contrary to a widespread misconception when they were first deployed, they do not help to mitigate oversmoothing (Wu et al., 2024) and this phenomenon can severely affect their performance. Yet oversmoothing in GATs is less often studied than in other GNNs. For example, the survey Rusch et al. (2023) does not cover them in its section on different methods for reducing oversmoothing.

2 PROBLEM DEFINITION

2.1 SOME NOTATIONS

- The set of nodes in the graph is called $\mathcal{V}$. We note Card$(\mathcal{V}) = n$.
- The set of neighbors of a node v is $\mathcal{N}(v)$ and the degree of v is noted deg(v).
- The set of edges in the graph is called E.
- Let $A \in \mathbb{R}^{n \times n}$ denote the adjacency matrix of the graph.
- Term-by-term multiplication of the elements of a matrix is denoted by $\odot$.
- Let $H \in \mathbb{R}^{n \times h}$ be the embedding matrix of the GNN output nodes (h is the embedding dimension).

2.2 OVERSMOOTHING MEASURES

As said in introduction, we measure oversmoothing through MAD, MADGap and Dirichlet Energy

2.2.1 MAD AND MADGAP

MAD stands for Mean Average Distance. It is always defined in relation to a target mask $M^{\text{tgt}} \in \{0,1\}^{n \times n}$. This mask defines the nodes whose embeddings we want to compare in our oversmoothing calculation. If we don't specify it, we're using the mask whose coefficients are all 1. To calculate the MAD, we first need the cosine distances between node embeddings. So we define the cosine distance matrix $D \in \mathbb{R}^{n \times n}$ by

$$\forall i, j \in [\![1, n]\!], \quad D_{i,j} = 1 - \frac{H_{i,:} H_{j,:}}{\|H_{i,:}\| \|H_{j,:}\|} \tag{1}$$

Then, we keep only the cosine distances that interest us for the calculation by defining:

$$D^{\text{tgt}} = D \odot M^{\text{tgt}} \tag{2}$$

We then average the non-zero values of D^{tgt} on each line i to obtain $\bar{D^{\text{tgt}}} \in \mathbb{R}^n$ defined by:

$$\forall i \in [\![1, n]\!], \quad \bar{D_i^{\text{tgt}}} = \frac{\sum_{j=1}^{n} D_{i,j}^{\text{tgt}}}{\sum_{j=1}^{n} \mathbb{1}_{\mathbb{R}^*}(D_{i,j}^{\text{tgt}})} \tag{3}$$

Taking the average of the non-zero values of this vector, we obtain the following:

$$\text{MAD}^{\text{tgt}} = \frac{\sum_{i=1}^{n} \bar{D}_i^{\text{tgt}}}{\sum_{i=1}^{n} \mathbb{1}_{\mathbb{R}^*}(\bar{D}_i^{\text{tgt}})} \tag{4}$$

MAD (relative to a 1-filled mask) describes how close all the embeddings are to each other. However, it is an imperfect measure of oversmoothing in that, as pointed out by Rusch et al. (2023), MAD can be zero without all embeddings being equal.
However, MAD allows us to define the notion of MADGap. It is defined as follows:

$$\text{MADGap} = \text{MAD}^{\text{rmt}} - \text{MAD}^{\text{neb}} \tag{5}$$

Where MAD^{neb} is the MAD of neighboring nodes (less than 3 edges apart), while MAD^{rmt} is the MAD of distant nodes (more than 8 edges apart). The idea behind MADGap is that we expect nearby nodes to have identical labels (or at least that it happens more often than with distant nodes). So, if there isn't too much oversmoothing, embeddings should vary more between distant nodes than between close ones. This would result in a high MADGap. But if oversmoothing is too high, all embeddings are roughly equal and the MadGap is close to 0 or even negative.

2.2.2 DIRICHLET ENERGY

The definition of Dirichlet energy $\mathcal{E}$ is as follows:

$$\mathcal{E} = \sum_{v_1 \in \mathcal{V}} \sum_{v_2 \in \mathcal{N}(v_1)} \left\| \frac{H(v_1)}{\sqrt{1+\deg(v_1)}} - \frac{H(v_2)}{\sqrt{1+\deg(v_2)}} \right\|_2^2 \tag{6}$$

where $H(v)$ is the embedding of node v.

As indicated by Rusch et al. (2023), the square root of the Dirichlet energy is indeed a node-similarity measure. This means that, unlike MAD or MADGap, when the Dirichlet energy is zero, all embeddings are equal. However, unlike MADGap, Dirichlet energy only takes into account the differences between embeddings of neighboring nodes. It therefore doesn't allow to compare variations in embeddings between long and short distances in the graph. Furthermore, since it is based on the L^2 norm and not on the cosine distance, the Dirichlet energy is not bounded by 1 and its order of magnitude may depend on the graph under study.

2.3 OBJECTIVE AND DIFFICULTIES OF THE PROJECT

Using the Dirichlet energy, MAD and MADGap oversmoothing measures, we aim to characterize the effectiveness of the oversmoothing limitation methods we are studying. The higher these three measures are, the less oversmoothing there is, which is why we aim to maximize them. We are also interested in the accuracy of the predictions made (we work exclusively on classification tasks). Indeed, the main reason for reducing oversmoothing is to improve the quality of predictions. We therefore hope that the methods studied will enable us to increase MAD, MADGap, Dirichlet energy and accuracy (for the same architecture). We're also looking at the impact of changing the number of layers on the quantities studied.
The multiplicity of oversmoothing measurements constitutes one of the difficulties of this project. Since they do not all measure exactly the same thing, the results can sometimes seem contradictory. In such situations, we feel that it is accuracy that should guide us and tell whether which oversmoothing measurement is "right". What's more, our oversmoothing measurements differed slightly at each run and therefore we needed to average them over 5 runs to get good results. However, we only had access to the power of our own GPUs, which was an important limitation during this project. Therefore, we did not stack more than 6 layers for each GNN (while Rusch et al. (2023) stacked 128 layers). Since oversmoothing is clearly visible as soon as we exceed 4 layers, our work remains useful for studying this phenomenon in moderately deep GNNs.

3 RELATED WORK

Oversmoothing in graph attention layers Veličković et al. (2018) has been mathematically demonstrated by Wu et al. (2024). This result shows the importance of work around oversmoothing minimization in GATs.
As indicated in the introduction, our work complements the survey conducted by Rusch et al. (2023). However, the latter did not deal with AdaEdge or MadReg, and limited itself to graph convolutional networks in its comparison of methods for reducing oversmoothing. To enable a comparison with the results of this article, we are working in particular on the Cora dataset, also used in the survey.
MadReg and AdaEdge techniques have been developed by Chen et al. (2019) but only tested with 4-layer GNNs. We check that we can reproduce their results and look at what happens when we change the number of layers. In addition, we also study the Dirichlet energy to describe oversmoothing, while only MAD and MADGap were used in the article.
Finally, although we haven't studied them, other penalties have already been introduced to reduce oversmoothing in GNNs. This is the case, for example, in Zhao et al. (2023)

4 METHODOLOGY

4.1 DATASETS

The Cora Dataset here illustrated in 1 consists of 2708 scientific publications classified into one of seven classes. The citation network consists of 5429 links. Each publication in the dataset is

described by a 0/1-valued word vector indicating the absence/presence of the corresponding word from the dictionary. The dictionary consists of 1433 unique words.

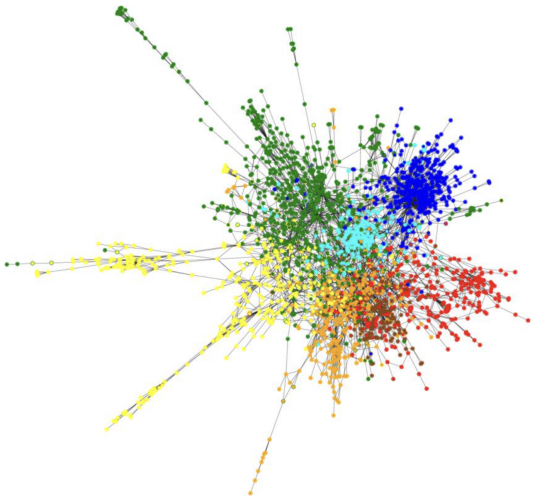

Figure 1: Graph representation of the Cora dataset

4.2 Models

4.2.1 Graph Convolutional Network (GCN)

The GCN model is a foundational graph neural network that applies a localized first-order approximation of spectral graph convolutions. The model aggregates neighbor node features linearly, which makes it particularly efficient and easy to scale to large graphs. Each layer in a GCN aggregates information from the direct neighbors and the node itself, effectively smoothing the features over the graph structure. GCNs are particularly prone to over-smoothing when many layers are stacked. This occurs because repeated aggregation tends to homogenize node features across the graph, leading to loss of useful information for differentiation of node representations. Studying GCNs can provide insights into the fundamental dynamics of over-smoothing in GNNs.

4.2.2 Graph Attention Network (GAT)

GAT introduces attention mechanisms to the aggregation process in GNNs. Each node computes coefficients that signify the importance of its neighbors' features. This mechanism allows nodes to weigh their neighbors' contributions differently, potentially mitigating the over-smoothing effect seen in simpler GNNs by focusing more on relevant neighbor features. Exploring GATs helps in understanding how attention mechanisms affect the distribution of node features across layers.

4.2.3 Enhanced Graph Attention Network (EnhancedGAT)

We propose a new model based on the traditional GAT model by incorporating multiple attention heads and nonlinear feedforward neural networks (MLPs) within each layer. This model aims to further enhance the feature discrimination capacity of each layer and tackle the over-smoothing problem through richer intermediate representations.

4.2.4 GraphSAGE

GraphSAGE extends GCN by incorporating a sampling strategy that computes embeddings by sampling a fixed number of neighbors instead of aggregating all neighbor nodes. GraphSAGE aims to reduce the computational burden and possibly control over-smoothing by limiting the scope of aggregation.

4.2.5 CHEBYSHEV NETWORKS (CHEBNET)

ChebNet utilizes Chebyshev polynomials of the Laplacian to generalize convolution in the spectral domain, providing a powerful way to capture multi-scale information on graphs.

4.3 METHODS TO LIMIT OVERSMOOTHING

4.3.1 MADREG

As introduced in Chen et al. (2019), we propose to implement the MADGap-based regularizer to the training loss in order to alleviate the over-smoothing issue. As the MADGap is directly linked to the information-to-noise ratio and the noise brought by neighboring nodes, add a regularize to the training loss will surely improve this information gain and eventually the final results of the model. Hence, we add MADGap to our CrossEntropy training loss (in the case of Cora dataset) to make the graph nodes receive more useful information and less interference noise:

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

Here we compute MADGap on the training set solely to be consistent with the cross-entropy loss. In our training process, we set fixed values of λ equal to 0, 0.01 and 0.1. The results of the experiment and the impact of MADReg can be found at **??**.

4.3.2 ADAEDGE

We also compute the AdaEdge (Adaptive Edge Optimization) method from Chen et al. (2019) which directly optimize the graph topology based on the model predictions. Here is the detail of the procedure:

Algorithm 1 Adaptive Edge Optimization (AdaEdge

Require: GNN model constructor: GNN, AdjustGraph function: $AdjustGraph$, Adjacency matrix: A, Features matrix: X, Maximum iterations: max_iter
Ensure: Adjusted adjacency matrix and final accuracy
$(acc_0, pred_0, conf_0) \leftarrow GNN(A, X)$
$A' \leftarrow AdjustGraph(pred_0, conf_0)$
for $iter = 1$ to max_iter **do**
    $(acc_i, pred_i, conf_i) \leftarrow GNN(A', X)$
    if $acc_i \leq acc_{i-1}$ **then return** acc_{i-1}
    end if
    $A' \leftarrow AdjustGraph(pred_i, conf_i)$
end for
return acc_{max_iter-1}

The AdaEdge method initiates with training a Graph Neural Network (GNN) on the original graph. Post initial training, the algorithm employs the model's predictive outcomes to inform adjustments to the graph topology. This involves the elimination of edges connecting nodes with disparate class predictions (inter-class) and the establishment of edges between nodes sharing class predictions (intra-class). Following this adjustment, the GNN is retrained from the ground up on the updated graph structure.

This process of optimizing the graph topology is not a one-off event but is iteratively applied. Through multiple iterations, AdaEdge progressively refines the graph, allowing for an evolving representation that aligns more closely with the learning task. By doing so, AdaEdge leverages the predictive power of GNNs to shape the graph into a form that is inherently more conducive to accurate model performance, thereby enhancing the robustness and generalization of the GNN in handling the task at hand.

In our node classification task in the Cora dataset, we set a fixed number of training iterations at 5 to refine the graph topology. We set the confidence level of class prediction to 0.15 as the dataset

contains 7 classes which correspond to a random baseline of 0.1428 for each class prediction. We found it reasonable to set this confidence level to 0.15 in order to force the graph topology to evolve.

4.4 Training process

The experiments utilized a range of GNN architectures, including GCN (Graph Convolutional Network), GAT (Graph Attention Network), GraphSAGE, ChebNet (Chebyshev Spectral CNN), and an Enhanced GAT model with multi-head attention mechanisms and integrated MLP layers. The models were trained on a standardized dataset with predefined features for nodes and edges, aiming to maximize node classification accuracy and graph understanding.

The training configuration and parameters are as followed:

- **Epochs and Learning Rate**: Each model was trained for up to 200 epochs with a learning rate of 0.01. However in order to gain time, we used early stopping: if the performances on the validation set didn't increase during the last 10 epochs, we stop the training there.
- **Loss Function**: Cross-entropy loss was utilized to compute the error rate between predicted and true labels in the Cora dataset
- **MADReg Regularization Parameters**: Experiments were conducted with MADReg regularization parameters set to 0, 0.01, and 0.1 to control over-smoothing and information-to-noise ratio.
- **Layer Variations**: Models were tested with varying depths, specifically 2, 3, 4, 5, and 6 layers, to understand the impact of depth on model performance and over-smoothing.
- **AdaEdge Implementation**: Optionally, AdaEdge was applied to dynamically adjust graph topology based on model predictions during training, involving either removal of inter-class edges or addition of intra-class edges.

For each model configuration, multiple runs (5 per configuration) were executed to ensure statistical reliability. Metrics such as accuracy, MAD, MADGap, Dirichlet energy , and edge modification rates were recorded. Results were aggregated, computing the mean and standard deviation for each metric across runs to assess performance stability and effectiveness. All experimental outcomes were compiled into a comprehensive results dataframe, capturing essential metrics for each configuration.

Overall, it is $5(models) \times 5(layers) \times 3(regularization) \times 2(AdaEdge) = 150$ different training configuration that we tested. The training process took nearly **7 hours** to run entirely on a unique GPU.

5 Results

We have stored the results of the 150 different training configurations in a csv file. We use this dataframe to compute all the visualisations and the following analysis.

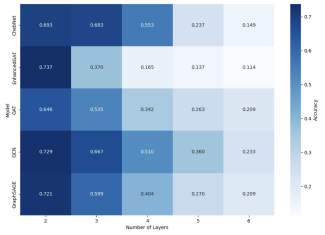

Figure 2: Effect of layers on model accuracy

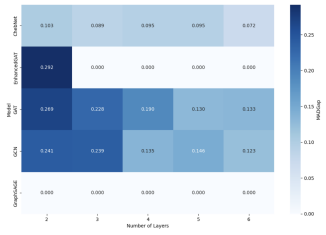

Figure 3: Effect of layers on MADGap

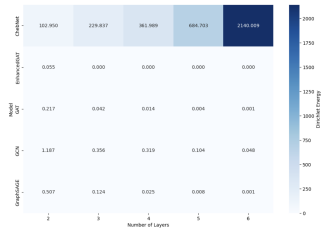

Figure 4: Effect of layers on Dirichlet Energy

In 2, 3 and 4 we conduct a detailed examination of the influence exerted by the number of layers on the performance metrics of various Graph Neural Network (GNN) models. The first subplot on

the left highlights a trend across models where accuracy uniformly declines as the number of layers increases. This observation suggests that model depth, contrary to enhancing performance, may introduce complexities that do not translate to better predictive capabilities. Indeed, as the number of layers grows, the models tend to suffer from the over-smoothing effect illustrated in the MADGap reduction on the right subplot.

The MADGap similarly exhibits a downward trend with additional layers. This decrease implies a growing inclination toward over-smoothing, where nodes' features become overly homogeneous and thus less effective for discriminative tasks. The parallel decline of accuracy and MADGap across all examined GNN architectures points to a consistent challenge of deep GNNs: as depth increases, the networks struggle to maintain the diversity of features, underscoring the intricate balance between a GNN's depth and its functional expressiveness. The clear, linear degradation in both metrics as the network depth increases underscores the critical need for strategies that counteract the diminishing utility of additional layers, such as implementing regularization techniques like MADReg that are specifically designed to combat over-smoothing and preserve feature distinctiveness.

For GraphSAGE, which is known for its ability to sample and aggregate features efficiently, and for EnhancedGAT, that employs attention mechanisms across 4 heads to weigh the importance of nodes' features, the null MADGap might suggest that its aggregation mechanism is leading to uniform feature representations. This could be a sign of over-smoothing, where the aggregation over several layers makes the node features converge to a point where inter-class and intra-class node representations are too similar.

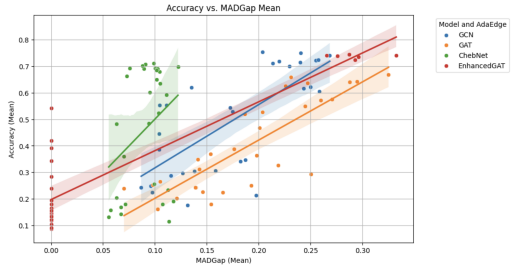

Figure 5: Accuracy vs. MADGap Mean

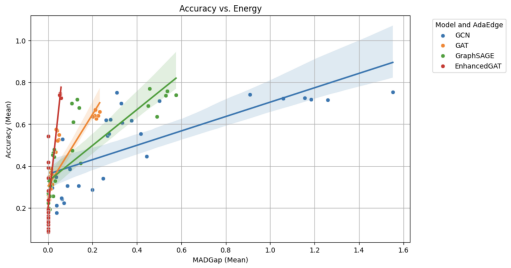

Figure 6: Accuracy vs. Dirichlet Energy

The figure 5 highlights the relationship between the MADGap and the accuracy of the models. The positive correlation between the MADGap and the accuracy suggests that the MADGap can be a good indicator of the model's performance. The regression lines show the trend for each model, with the EnhancedGAT model having the steepest slope, indicating that it has the highest sensitivity to changes in the MADGap. This plot can help in understanding how the MADGap can be used to evaluate the performance of different models and guide the selection of the best model for a given task. GraphSAGE has been excluded from 5 as it has null MADGap, making it difficult to conclude on any link between performance and this metric here.

In this particular case, the Dirichlet energy is of interest because it gives conclusive results on GraphSAGE. Indeed, the higher it is, the greater the accuracy of the model. MadGap and Dirichlet energy also seem to point in the same direction for the other models, except for ChebNet. For ChebNet, Dirichlet energy diverges as the number of layers increases, for reasons we don't understand (while accuracy decreases). This shows that Dirichlet energy is not a perfect measure of oversmoothing either. In what follows, we'll focus exclusively on MADGap, which seems to be more stable than Dirichlet energy.

In table 1 we show the mean value and the standard deviation of the accuracy and MADGap values for each of the model for different number of layers (from 2 to 6). Darker color means larger improvement thanks to MADReg penalization. We show that the MADReg method for $\lambda = 0.1$ and $\lambda = 0.01$ comes with better performance of the model and less over-smoothing (controlled by MADGap).

In the Appendix, the different barplots show the effect of MADReg and AdaEdge on the accuracy of the models for each layer. We spot that the effect of MADReg and AdaEdge is more pronounced

LAYER	Metric	GCN			GAT			ChebNet			EnhancedGAT			GraphSAGE		
		$\lambda = 0$	$\lambda = 0.01$	$\lambda = 0.1$	$\lambda = 0$	$\lambda = 0.01$	$\lambda = 0.1$	$\lambda = 0$	$\lambda = 0.01$	$\lambda = 0.1$	$\lambda = 0$	$\lambda = 0.01$	$\lambda = 0.1$	$\lambda = 0$	$\lambda = 0.01$	$\lambda = 0.1$
Layer2	acc(%)	0.7224 ± 0.0348	**0.7408 ± 0.0482**	0.7242 ± 0.0631	0.6258 ± 0.0352	**0.6418 ± 0.0577**	0.6412 ± 0.0514	0.6916 ± 0.0176	0.6802 ± 0.0228	**0.6980 ± 0.0311**	0.7352 ± 0.0324	0.7248 ± 0.0378	**0.7404 ± 0.0298**	0.7584 ± 0.0198	0.7400 ± 0.0230	0.6354 ± 0.1598
	MADGap	0.2552 ± 0.0387	**0.2686 ± 0.0314**	0.2577 ± 0.0365	0.2257 ± 0.0481	**0.2947 ± 0.0294**	0.2879 ± 0.0869	0.1002 ± 0.0122	0.1046 ± 0.0174	**0.1223 ± 0.0167**	0.2963 ± 0.0460	0.2927 ± 0.0563	0.2656 ± 0.0459	0.0001 ± 0.0000	0.0001 ± 0.0000	0.0002 ± 0.0002
Layer3	acc(%)	0.6998 ± 0.0920	0.6164 ± 0.1195	**0.7112 ± 0.0464**	0.5262 ± 0.0895	**0.5710 ± 0.1575**	0.5488 ± 0.1065	0.6924 ± 0.0489	0.6498 ± 0.0530	**0.7078 ± 0.0367**	0.2386 ± 0.1090	0.3922 ± 0.1173	**0.5418 ± 0.1895**	0.6102 ± 0.1593	0.6984 ± 0.0305	**0.7172 ± 0.0585**
	MADGap	0.2295 ± 0.0537	**0.2434 ± 0.0514**	0.2138 ± 0.0839	0.2037 ± 0.0397	**0.2600 ± 0.0882**	0.2448 ± 0.0524	0.0760 ± 0.0043	**0.1017 ± 0.0353**	0.0909 ± 0.0355	0.0000 ± 0.0000	0.0002 ± 0.0002	0.0001 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000
Layer4	acc(%)	0.4458 ± 0.0898	0.5438 ± 0.1923	**0.5534 ± 0.1587**	0.3124 ± 0.0901	0.3270 ± 0.0732	**0.3630 ± 0.0957**	0.5912 ± 0.0933	**0.6016 ± 0.0721**	0.4852 ± 0.1384	0.1796 ± 0.1126	0.1580 ± 0.0399	**0.1942 ± 0.0890**	0.3294 ± 0.1088	**0.4786 ± 0.2210**	0.2568 ± 0.0745
	MADGap	0.1039 ± 0.0769	**0.1727 ± 0.0979**	0.1047 ± 0.0321	0.1430 ± 0.0684	**0.2186 ± 0.1307**	0.1979 ± 0.0850	0.1110 ± 0.0619	0.0952 ± 0.0502	0.0941 ± 0.0404	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000
Layer5	acc(%)	0.5282 ± 0.1475	0.3864 ± 0.1530	0.3066 ± 0.1321	0.2498 ± 0.0475	**0.3492 ± 0.1126**	0.2522 ± 0.1059	0.2552 ± 0.1346	0.1904 ± 0.0799	**0.3594 ± 0.1439**	0.1546 ± 0.0952	0.1332 ± 0.0958	0.1190 ± 0.0272	0.3248 ± 0.1564	0.2820 ± 0.1251	0.3056 ± 0.1659
	MADGap	0.1751 ± 0.0916	0.1049 ± 0.0421	0.1586 ± 0.0531	0.1928 ± 0.0885	0.1413 ± 0.0244	0.0981 ± 0.0646	**0.0995 ± 0.0285**	0.1178 ± 0.0545	0.0702 ± 0.0390	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000
Layer6	acc(%)	0.2436 ± 0.0873	**0.2962 ± 0.0891**	0.2234 ± 0.0841	0.2242 ± 0.1044	0.1794 ± 0.0357	0.1604 ± 0.0463	0.1796 ± 0.0818	0.1432 ± 0.0740	0.1154 ± 0.0359	0.1190 ± 0.0249	**0.1362 ± 0.0952**	0.1330 ± 0.0809	0.2234 ± 0.0999	0.1878 ± 0.0643	0.1990 ± 0.0480
	MADGap	0.0868 ± 0.0487	**0.1269 ± 0.0527**	0.0975 ± 0.0618	0.1656 ± 0.0433	0.1542 ± 0.0774	0.1029 ± 0.0620	0.0717 ± 0.0195	0.0671 ± 0.0223	**0.1135 ± 0.0524**	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000

Table 1: Accuracy and MADGap metrics comparison across different GNN models with and without MADReg regularization.

for models with two layers and three layers, this shows the trade-off between regularization and adaptive edge removal and addition in the context of graph neural networks.

6 Conclusion

Our work confirms that MADReg and AdaEdge are methods that can limit oversmoothing in GNNs. However, these methods do not prevent accuracy from decreasing sharply when the number of layers is increased by a large extent. In fact, this is the case with most methods for limiting oversmoothing in GNNs. Only G^2-GCN seemed not to suffer from this problem in Rusch et al. (2023). An extension of our work would be to combine MadGap and AdaEdge with the other methods presented in Rusch et al. (2023).

References

Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, and Xu Sun. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view, 2019.

William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs, 2018.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2017.

T. Konstantin Rusch, Michael M. Bronstein, and Siddhartha Mishra. A survey on oversmoothing in graph neural networks, 2023.

Shanshan Tang, Bo Li, and Haijun Yu. Chebnet: Efficient and stable constructions of deep neural networks with rectified power units via chebyshev approximations, 2023.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2018.

Xinyi Wu, Amir Ajorlou, Zihui Wu, and Ali Jadbabaie. Demystifying oversmoothing in attention-based graph neural networks, 2024.

Weichen Zhao, Chenguang Wang, Congying Han, and Tiande Guo. Exploring over-smoothing in graph attention networks from the markov chain perspective, 2023. URL `https://openreview.net/forum?id=EdH_fkyhAO`.

A Appendix

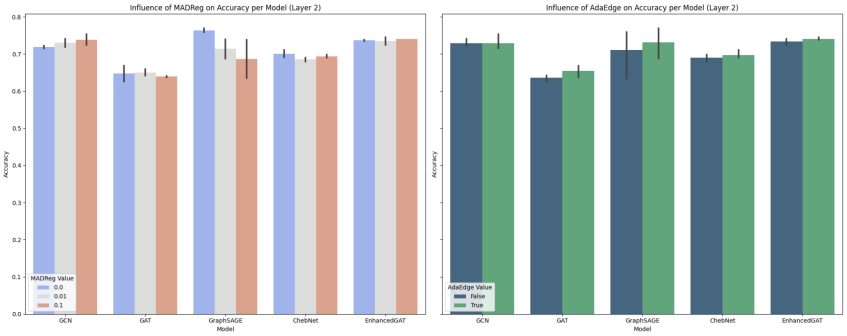

Figure 7: Layer 2

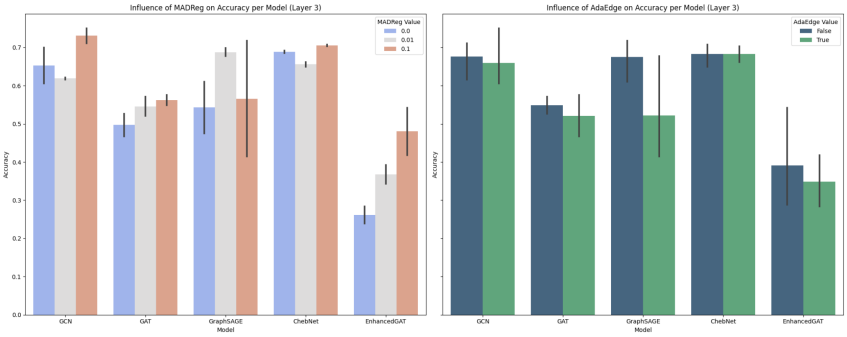

Figure 8: Layer 3

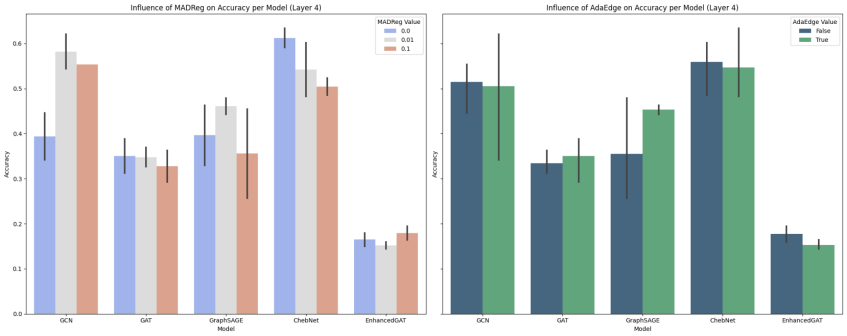

Figure 9: Layer 4

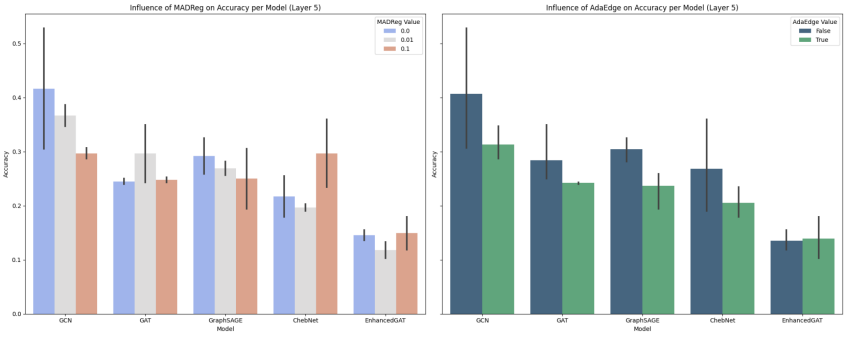

Figure 10: Layer 5

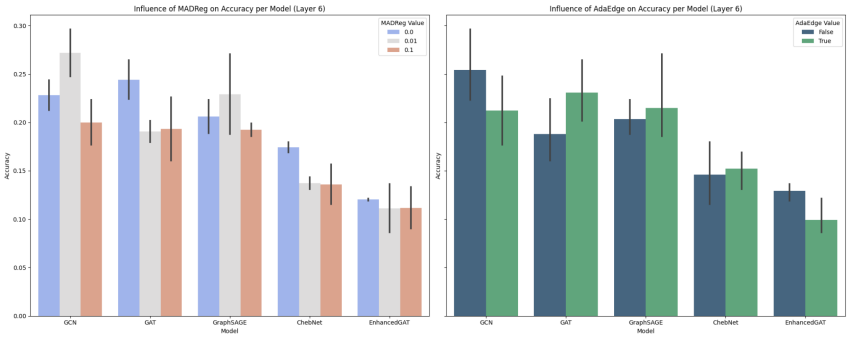

Figure 11: Layer 6