Oversmoothing Analysis of Node Classification

Eric Zander and Riya Ponraj

1 Introduction

Node classification is a ML task that assigns labels to nodes in a graph based on the properties of the nodes and their relationships. Graph Neural Networks are helpful for node classification tasks due to their ability to capture relationships between nodes in a graph. One common challenge in GNNs is the issue of oversmoothing which means node representations become indistinguishable as the number of message-passing layers increases. This report provides a comprehensive analysis of oversmoothing in node classification, focusing on the performance impact of varying the number of message-passing layers. There are two types of graph datasets: Homophilous and Heterophilous. Homophilous graphs are the graphs whose edges tend to connect nodes of the same class. Graphs without this property are called heterophilous. We are using Heterophilous graph datasets in this report. The datasets used here are obtained from the paper, "A Critical Look at the Evaluation of GNNs under Heterophily: Are We Really Making Progress?". The datasets include "Roman-empire," "Amazon-ratings," "Minesweeper," "Tolokers," and "Questions," each of which has distinct structural properties which are helpful to study oversmoothing in GNNs.

2 Background

To effectively analyze oversmoothing in node classification, it is essential to work with datasets like Heterophilous datasets that provide a challenge of nodes being connected to nodes of different labels rather than the same label. To address this need, five heterophilous datasets were used, each offering unique graph structures, class distributions, and node connectivity patterns. These datasets are used to benchmark GNN performance in heterophilous graph settings in the paper "A Critical Look at the Evaluation of GNNs under Heterophily: Are We Really Making Progress?" The criteria for using this dataset is, of course, the Heterophily (accounts for class imbalance and reliability), Diversity (obtained from various domains and different structural characteristics) and the Size (Each graph contains 10K to 50K nodes).

Description of the datasets: [1]

1. Roman-Empire - It is a word-level graph from the Roman Empire Wikipedia article. Each nodes represent words, and edges connect the sequential or syntactically related words. Each word is connected to another if the word follows or there is a dependency within the sentence. The task is to classify nodes into 18 syntactic roles using FastText word embeddings as features. The graph has

22.7K nodes, 32.9K edges, and an adjusted homophily of h_{adj} = -0.05, reflecting its heterophilous nature.

2. Minesweeper - This dataset simulates a 100x100 Minesweeper grid. The nodes are cells linked to adjacent cells. The task is to predict if a cell contains a mine or not, with 20% of nodes being mines. The graph's adjusted homophily is approximately 0, reflecting its randomness. The structure of this graph is significantly different from the other datasets due to its regularity

3. Amazon-ratings- This dataset models product co-purchasing relationships. The nodes are products, and edges link frequently co-purchased products. The task is to classify products into five rating categories using fastText embeddings. The 5-core subgraph of the largest component is used, focusing on the most relevant nodes for recommendation tasks.

4. Tolokers - It represents workers from the Toloka crowdsourcing platform. The nodes are workers, and the edges connect workers who collaborated on the same task. The goal is to predict banned workers using profile and task performance data. The graph contains 11.8K nodes with an average degree of 88.28, making it dense and challenging.

5. Questions - This dataset models users on the Yandex Q platform. The nodes are users, and the edges connect users who answer each other's questions. The goal is to predict which users remain active. Node features include FastText embeddings of user descriptions, with binary indicators for missing data. The graph contains 48.9K nodes and is highly unbalanced, with 97% of users active, and has an adjusted homophily of $h_{adj} = 0.02$.

3 Objective

1. Implement GNN-based models for node classification on heterophilous graph datasets.

2. Vary the message-passing layer design and analyze its impact on classification performance.

3. Analyze the relationship between the number of message-passing layers and node classification performance, with a focus on the occurrence of oversmoothing.

4 Method

To test the impact of message-passing layer count on classification performance, we implemented graph convolutional networks (GCNs) and graph attention networks (GATs) with PyTorch Geometric's nn.GCNConv and nn.GATConv respectively. Configurations with a varying number of convolutional layers were tested on all five datasets. To measure their success, we considered accuracy, precision, recall, F1 score, and ROC-AUC. However, while weighted averages and one-vs-one approaches allowed for computation of precision, recall, F1 scores, and ROC-AUC for the multi-class datasets, we primarily

focused on accuracy in these cases and ROC-AUC for binary classification given the imbalanced nature of the Tolokers dataset in particular.

Additionally, we saved input embeddings provided by consecutive message passing layers. We then employed t-SNE for dimensionality reduction and visualized the results to illustrate the impact of message passing.

4.1 Model Architecture

Our network architectures were loosely based on those described by Platonov et al.[1] with modifications made to achieve similar results despite employing smaller models for more efficient testing of additional message passing layers.

They consist of a series of convolutional message passing layers followed by two linear layers as per a standard multi-layer perceptron (MLP). Layer normalization is employed alongside linear projection and skip connections between the message passing layers. For the activation function, Gaussian Error Linear Unit (GELU) is applied as by Platonov et al. after layer normalization as well as between the MLP hidden layers.

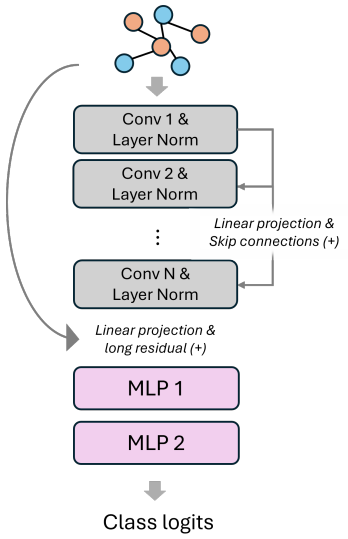

Figure 1: Overview of model architecture. Linear projections are created to support skip connections between convolutional layers as well as a long residual connection from the input to MLP blocks.

The use of linear projection and skip/residual connections afforded results approximating those indicated by Platonov et al. despite smaller message passing layers; we employed at most 5 message passing layers with incrementally decreasing dimensions (512, 256, 128, etc.) as opposed to many layers outputting dimensions of exclusively 512 as described. However, it is important to emphasize that the further layers can more directly access the information in earlier layers as a result. Consequently, the negative impact of largely unhelpful message passing layers for heterophilous datasets in particular will be more easily mitigated. This is important to consider when evaluating the significance of oversmoothing.

4.2 Embedding Analysis

To further analyze the impact of additional message passing layers, we collect embeddings output by each convolutional layer, project them into 2 dimensions with a dimensionality reduction method, and create scatter plots.

5 Experiment

We created GCNs and GATs as described with 2, 3, 4, and 5 message passing layers. At minimum, 2 convolutional layers are created to map input data to an embedding space and then the output to the input size of the MLP. For more than 2 layers, used incrementally halved embedding dimensions (512, 256, 128, and 64) to counterbalance the rapid growth in parameters.

Each model was trained for all five heterophilious datasets with the same train/test/val splits accessed via PyTorch Geometric. Finally, embeddings were retrieved for the models with the most message passing layers.

5.1 Classification Results

The classification metrics for each model are in Table 1 and visualized in Figure 2.

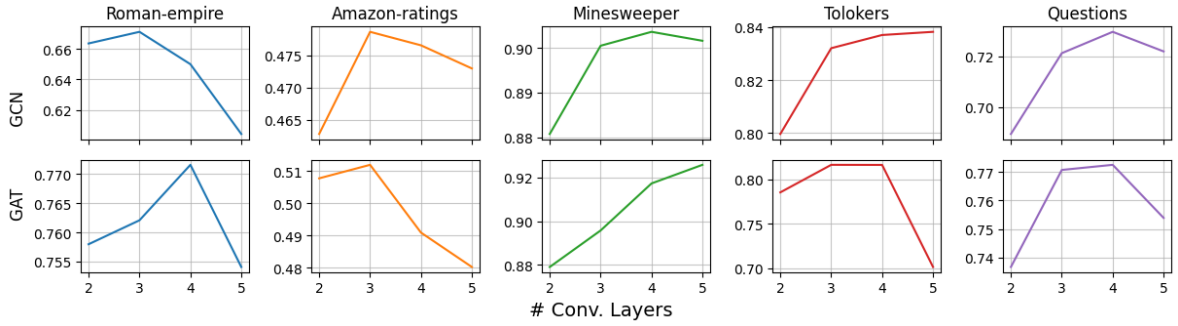

Figure 2: Plots of the relevant metrics per dataset and model type.

5.2 Embedding Visualization

After training and testing classification, we saved convolutional layer outputs of the models with the most message passing layers and plotted them. The results are depicted in Figure 4. We chose t-SNE for dimensionality reduction due to its capacity for preserving local structure in spite of high dimensionality.

We also tested models with many smaller message passing layers. An example of the results is shown in Figure 3.

6 Conclusion

The classification results generally indicate that, while the benefits of additional message passing layers vary based on the model and data, performance generally degrades as oversmoothing renders input features too similar for meaningful distinction.

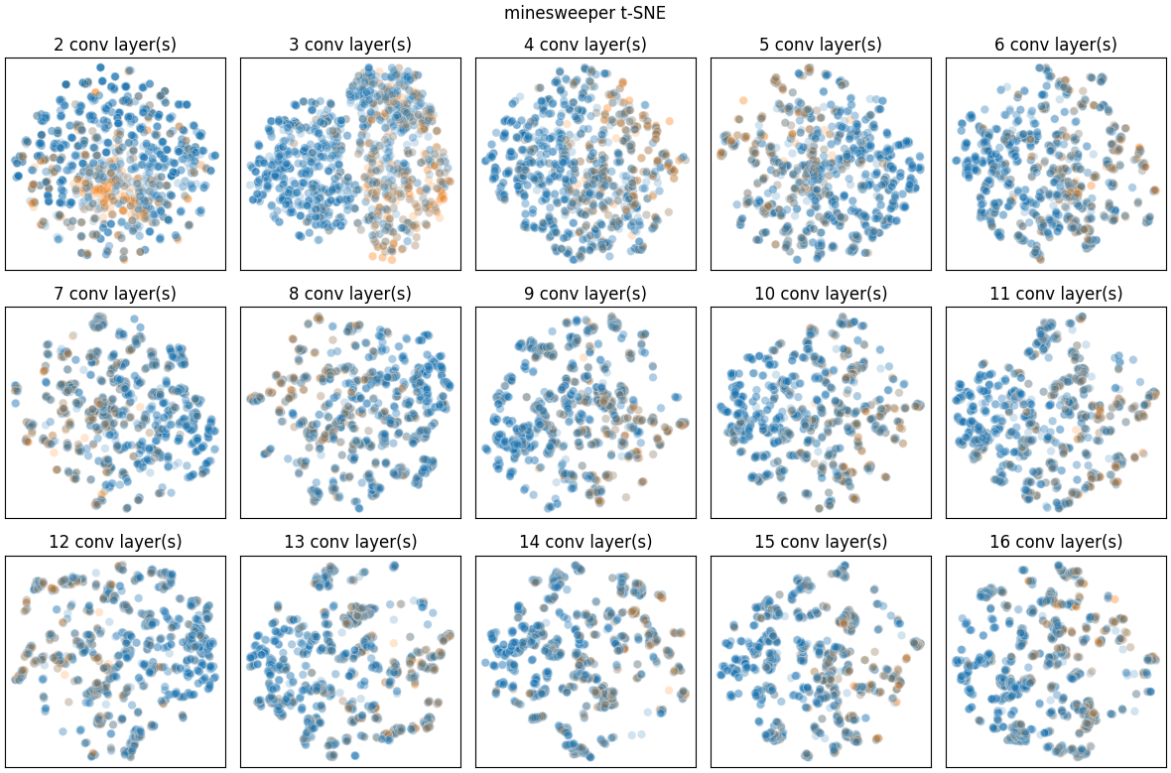

Figure 3: Minesweeper embeddings with more message passing layers.

Generally, GATs outperformed GCNs in accuracy for the multiclass datasets and ROC-AUC for all binary classification datasets besides the Tolokers dataset. These results are similar to those achieved by Platonov et al.; their GCN achieved 83.64 on the Tolokers dataset, and their GAT achieved 83.70.

References

[1] Oleg Platonov, Denis Kuznedelev, Michael Diskin, Artem Babenko, and Liudmila Prokhorenkova. A critical look at the evaluation of gnns under heterophily: Are we really making progress?, 2024.

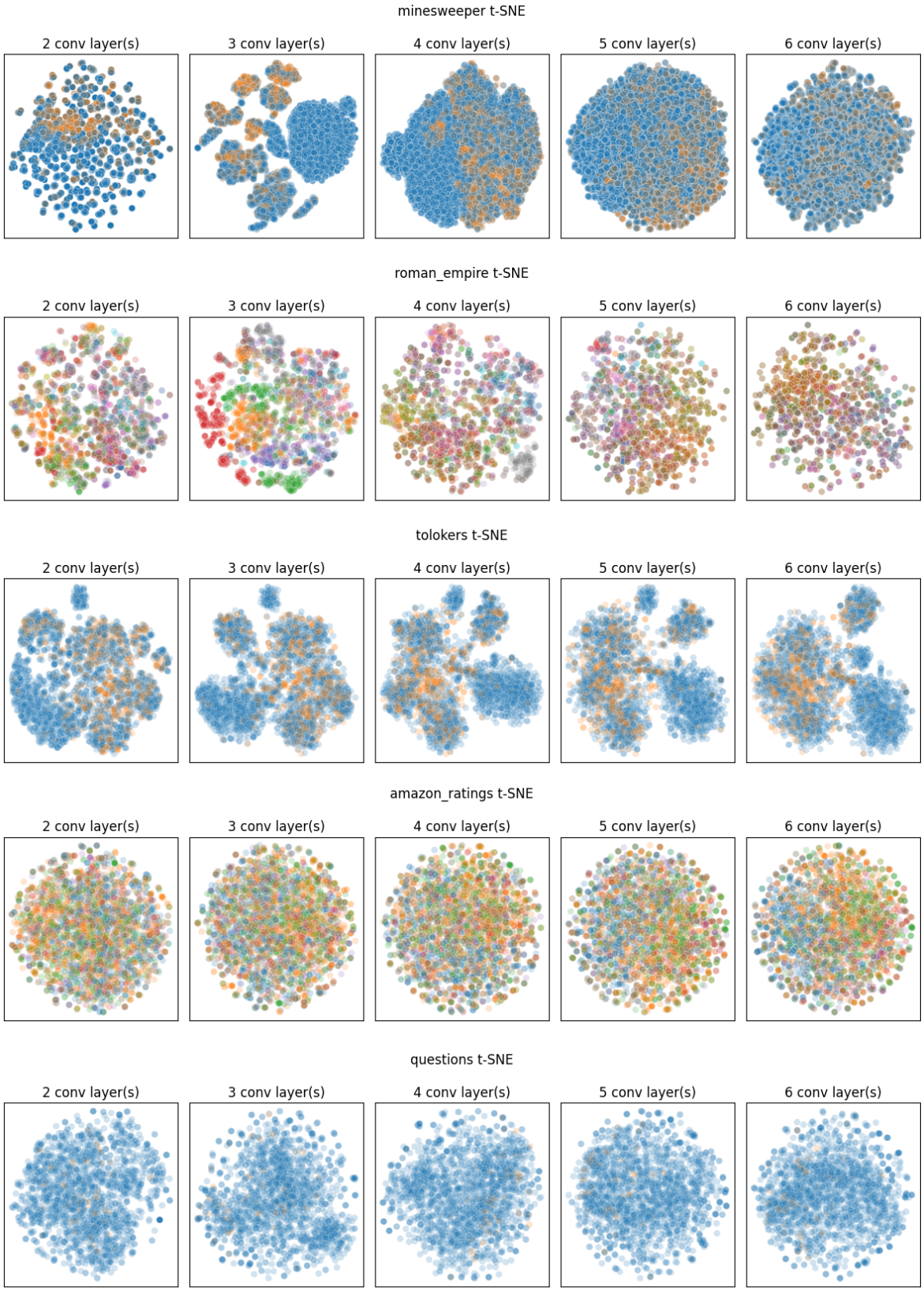

Figure 4: Input embeddings from different numbers of message passing layers.

Model	# Layers	Accuracy	Precision	Recall	F1	ROCAUC
Roman-empire						
GCN	2	**0.6636**	0.6293	0.6636	0.6272	0.9166
	3	**0.6712**	0.6546	0.6712	0.6368	0.9155
	4	**0.6500**	0.6375	0.6500	0.6179	0.9119
	5	**0.6043**	0.5592	0.6043	0.5591	0.8835
GAT	2	**0.7580**	0.7504	0.7580	0.7526	0.9599
	3	**0.7621**	0.7568	0.7621	0.7582	0.9594
	4	**0.7716**	0.7674	0.7716	0.7683	0.9594
	5	**0.7541**	0.7510	0.7541	0.7517	0.9567
Amazon-ratings						
GCN	2	**0.4628**	0.4453	0.4628	0.4232	0.6920
	3	**0.4787**	0.4431	0.4787	0.4554	0.7122
	4	**0.4766**	0.4673	0.4766	0.4553	0.7114
	5	**0.4730**	0.4402	0.4730	0.4527	0.7087
GAT	2	**0.5078**	0.5017	0.5078	0.4998	0.7315
	3	**0.5120**	0.5064	0.5120	0.5045	0.7405
	4	**0.4909**	0.4832	0.4909	0.4828	0.7311
	5	**0.4802**	0.4671	0.4802	0.4672	0.7236
Minesweeper						
GCN	2	0.8360	0.8232	0.8360	0.8267	**0.8808**
	3	0.8560	0.8516	0.8560	0.8535	**0.9005**
	4	0.8604	0.8566	0.8604	0.8582	**0.9036**
	5	0.8564	0.8546	0.8564	0.8555	**0.9016**
GAT	2	0.8384	0.8265	0.8384	0.8299	**0.8791**
	3	0.8496	0.8503	0.8496	0.8499	**0.8958**
	4	0.8704	0.8653	0.8704	0.8671	**0.9173**
	5	0.8776	0.8730	0.8776	0.8746	**0.9258**
Tolokers						
GCN	2	0.7905	0.7570	0.7905	0.7564	**0.7996**
	3	0.8092	0.7882	0.8092	0.7735	**0.8321**
	4	0.8160	0.7968	0.8160	0.7891	**0.8371**
	5	0.8170	0.7979	0.8170	0.7945	**0.8383**
GAT	2	0.7874	0.7525	0.7874	0.7155	**0.7854**
	3	0.7820	0.7644	0.7820	0.7707	**0.8163**
	4	0.8010	0.7749	0.8010	0.7741	**0.8162**
	5	0.7816	0.6109	0.7816	0.6858	**0.7017**
Questions						
GCN	2	0.9702	0.9412	0.9702	0.9555	**0.6894**
	3	0.9704	0.9595	0.9704	0.9571	**0.7212**
	4	0.9706	0.9650	0.9706	0.9568	**0.7296**
	5	0.9703	0.9596	0.9703	0.9565	**0.7219**
GAT	2	0.9707	0.9615	0.9707	0.9582	**0.7366**
	3	0.9706	0.9597	0.9706	0.9591	**0.7708**
	4	0.9704	0.9590	0.9704	0.9592	**0.7726**
	5	0.9701	0.9578	0.9701	0.9591	**0.7539**

Table 1: Test set model evaluation metrics by dataset and type (GCN vs. GAT).