Maintaining Accuracy of Graph Neural Networks with Increasing Depth

Ayush Jain aj3152@nyu.edu
Courant Institute of Mathematical Sciences
New York University
New York, NY, USA

Harsh Dubey hd2225@nyu.edu
Courant Institute of Mathematical Sciences
New York University
New York, NY, USA

Hemant Ramawat hr2378@nyu.edu
Courant Institute of Mathematical Sciences
New York University
New York, NY, USA

Abstract

Graphical Neural Networks (GNNs) have been observed to perform much better than neural networks for graph-based data. Though, as we scale up, we see a noticeable drop in performance while training GNNs with increasing depth unlike neural networks which show improvement in performance with increasing layers. Several studies have attributed the performance decline to oversmoothing and oversquashing which confirms that increasing depth renders the node embeddings from different classes indistinguishable. In this paper, we aim to systematically investigate these reasons behind the drop in accuracy with the increasing depth in GNNs and evaluate it's corresponding mitigation strategies.

Keywords: Graph Neural Networks, Oversmoothing, Oversquashing
Datasets: Cora, Karate Club, PubMed

1. Introduction

Deep learning frameworks so far have proved to be very successful in dealing with data involving grids and sequences. Though, it has been observed that a lot of data around us cannot be best represented as sequences and grids like social networks, molecules and communication networks. These complex domains are best represented by a graphical network which have a rich relational data structure formed with the help of nodes and their corresponding edges representing the relationships between them. We can utilize this rich relationship data by explicitly modeling the relationships to improve the performance.

Graph Neural Networks (GNNs) in recent years have emerged as a promising framework to analyse and learn graphical representations as they best utilize rich relational information from graphical networks (Xu et al., 2018). GNNs are a general framework for modeling complicated structural data having nodes (elements) and edges (connections) between them. As a result, GNNs are extensively used to simulate numerous graph-structured domains. GNN can be seen as a message-passing step where each node in a GNN layer updates its state or node embedding by collecting messages coming from its direct neighbors. The active research in the field led to development of advanced GNN, namely, graph convolutional networks, graph attention networks, and simple graph convolution networks. The main difference between all the GNN variations is how each node aggregates its neighbors' representations with its own. Most issues, on the other hand, require interaction between nodes that are not directly coupled, which is accomplished by stacking numerous GNN layers.

Despite active research, learning graphical networks still remains a challenging task as they have arbitrary form with complex topological structures unlike text and images which maintain spatial locality. There is also a missing reference point in graphs and also exhibit dynamic and multimodal features. Apart from the structural constraints, graph neural networks also face issues trying to scale up while dealing with large scale graphical networks as a significant performance deterioration is observed when the depth of graph increases. This behavior of GNNs is in complete contrast with neural networks which scale up very efficiently because they show an improvement in accuracy with increasing number of layers.

In this paper, we attempt to confirm the drop in accuracy with increasing depth in GNNs. After we have established this behavior, we'll try to explore and mitigate the potential reasons behind it. Firstly, we'll select and study the potential reasons studied so far behind the drop in accuracy and then discuss relevant measurement metrics to quantify and confirm the potential reasons selected. After, establishing the role of the selected potential reasons, we'll run experiments to maintain the accuracy with increasing depth in GNNs.

2. Background

2.1 Graph Operations

A GNN layer has two basic operations: it aggregates messages from all nearby nodes and combines them with the node's previous embedding. The functioning of a singe layer of a GNN can be explained from Figure 1.

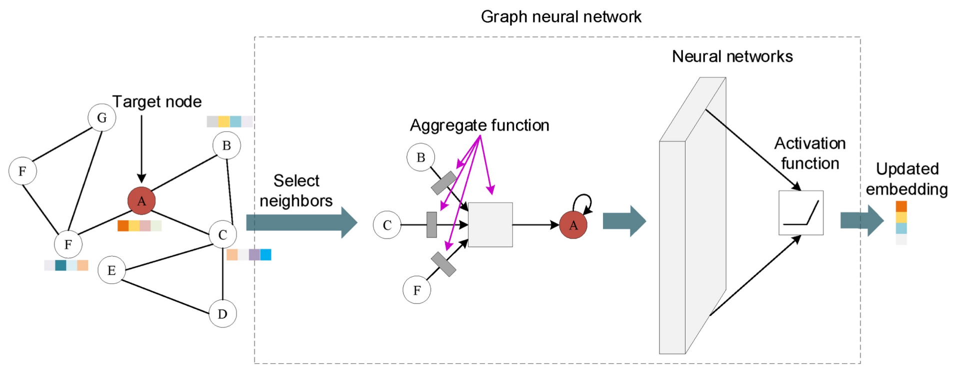


Figure 1: GNN Architecture (Zeng and Tang, 2021).

This process creates a single GNN layer, which is then reproduced several times by stacking GNN layers. By including neighbors of neighbors, and so on, as the number of layers grows, the receptive field of each node grows as well. Every node in a K-layer GNN has a receptive field for its K-hop neighborhood.

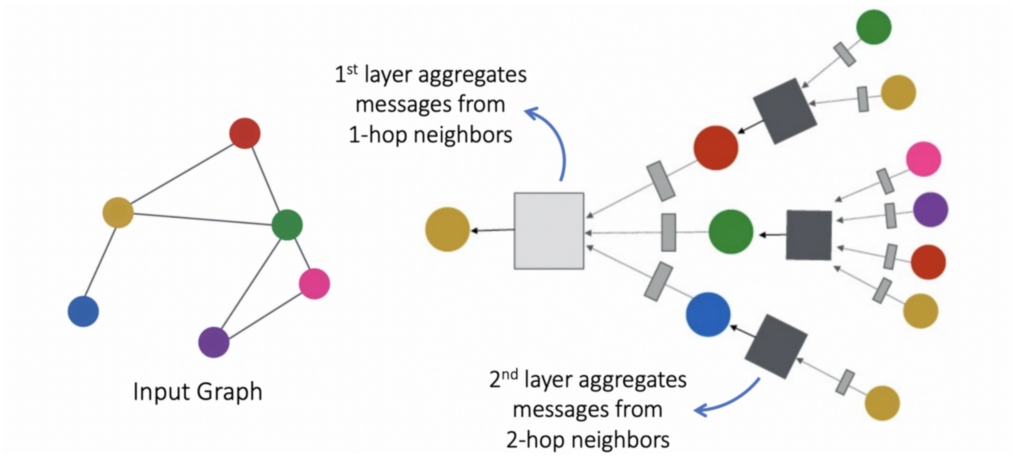

1st layer aggregates messages from 1-hop neighbors
Input Graph
2nd layer aggregates messages from 2-hop neighbors

Figure 2: Mechanism of a single layer of Graph Neural Networks (Leskovec, 2021).

Mathematically,

$$a_v^{(k)} = \text{AGGREGATE}^{(k)}\left(\left\{h_u^{(k-1)} : u \in \mathcal{N}(v)\right\}\right)$$

$$h_v^{(k)} = \text{COMBINE}^{(k)}\left(h_v^{(k-1)}, a_v^{(k)}\right)$$

where $h_v^{(k)}$ represents the embedding of node v at layer k and $\mathcal{N}(v)$ denotes the neighbors of node v

The context is communicated through the graph's edges in this case. This formulation has the advantage of allowing the GNN to function on graphs of any shape or size. It also assures that the update step is permutation invariant, which means it is unaffected by the order of the nodes. The above style can be used to write most existing GNN algorithms, and different choices for aggregate and combine functions result in distinct GNN models. We can use various aggregation approaches (max, sum, mean) to get the embedding for the complete network once we have the embeddings for all nodes. While each network has its own set of benefits and drawbacks, there is no single best network. Model architecture is often an empirical option that depends on the dataset and task at hand (discussed in section 4.2.3).

2.2 Variants of Graph Neural Network

3. Related Work

GNNs's main component, a neighborhood aggregator, is essentially a low-pass smoothing operation that updates a node's representation iteratively by mixing itself with its neighbors' representations. Because the linked nodes tend to be similar, it appears to fit in with graph architectures. As the number of graph convolutional layers grows, all node representations across a graph converge to indistinguishable vectors, and GNNs perform poorly in the downstream applications. This is a well-known problem known as over-smoothing. GNN models are unable to dig deeper into multi-hop neighborhood structures and learn improved node representations because of this problem.

Moreover, over-smoothing was largely seen in short-range tasks — those with an expected engagement with nodes in small radius and where a node's successful prediction is heavily reliant on its immediate surroundings. Paper subject categorization and product category classification are examples of such challenges. Because the learning issues in these datasets rely mostly on short-range information, it's understandable that more layers beyond the problem radius are irrelevant. On the other hand, we see the problem of over-squashing in tasks that primarily rely on long-range information with a large radius of interest.

GNNs should contain at least N layers to allow a node to receive information from other nodes within a radius of N nodes, otherwise distant nodes will simply be unaware of each other. Long-range interaction problems clearly demand as many GNN layers as the interaction range to avoid under-reaching issue. The number of nodes in each node's receptive field, however, expands exponentially as the number of layers increases. Information from the exponentially increasing receptive field is compressed into fixed-length node vectors, causing over-squashing. As a result, the graph is unable to transmit messages from distant nodes, and the training data is used to learn only short-range signals.

Many improved measurement metrics have been developed in attempts to best quantify the problem of over-smoothing and over-squashing. Researchers have observed the graph between classification accuracy and depth, cosine similarity score between all node pairs and MAD (Mean Average Deviation) scores.

Regularizing the node distance, node/edge dropping, batch and pair normalizations have all been tried to alleviate the over-smoothing problem (Oono and Suzuki, 2020).

4. Investigation on Accuracy Deterioration

In this section, we try to empirically understand the accuracy drop with increasing depth in GNNs. We run the experiment to plot and visualize the node embeddings on Cora dataset. Then, we utilise this observation in the context of receptive field to investigate and understand the accuracy drop (Alon and Yahav, 2021).

4.1 Node Embedding Similarity

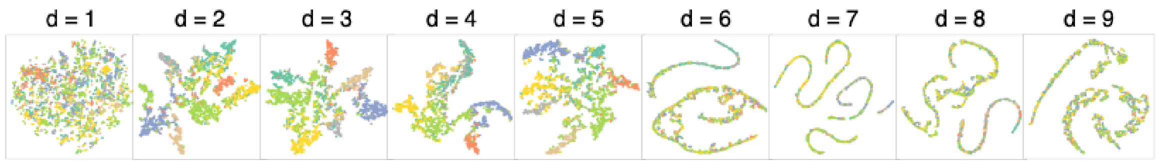


Figure 3: Plots of the node classification on Cora dataset during training where different color represents the different class of the graph (d denotes depth here).

We visualized the node embedding (Figure 3) in low dimension space for Cora dataset in GNNs (Liu et al., 2020). We ran the experiment starting from depth 1 running till depth 9. Each color represents a distinct class in the plots above. At depth of 1, we can observe that all the classes are easily distinguishable since the corresponding node embeddings are distinct. As we increase the depth, we can observe that it gets difficult to separate the distinct classes due to increased similarity between node embedding of among all nodes in the graphical network. This visualization confirms that as we increase the depth in GNNs, the node embedding tends to become more similar leading to the point that it becomes difficult to differentiate one node from other. Since, at higher depths, nodes lose their distinctiveness, it leads to accuracy drop which establishes the problem of oversmoothing (Cai and Wang, 2020).

4.2 Factors leading to Oversmoothing

Now, we'll try to study various factors which explicitly or implicitly contribute towards oversmoothing.

4.2.1 Sampling strategy

Sampling strategies decides which nodes are of interest for a given node among it's direct and indirect neighbors. Since sampling strategy essentially defines what nodes will have a contribution in the node embedding of a given node, it plays an importance role in tackling oversmoothing. There are many good choices of sampling strategies as,

- Sampling by random node selection: The most obvious technique to make a sample graph is to choose a set of nodes N at random, and then generate a sample graph from the set of nodes N. This algorithm is known as Random Node (RN) sampling.
- Edges can be selected evenly at random in the same way that nodes can be. This algorithm is known as Random Edge (RE) sampling. as Random Edge (RE) sampling
- Sampling by exploration: The general idea behind this family of sampling algorithms is to pick a node at random and then examine the nodes in its neighborhood. This algorithm is known as Random Node Neighbor.

4.2.2 Receptive field

The complete set of nodes in range within K hops makes up the receptive field for a given node in GNN with depth K.

We can observe in Figure 4 that as we increase the depth or number of layers for a given node in GNNs, the node's receptive field increases exponentially due to the network effect in play. Therefore, at higher depths, for any two given node, the most of the nodes in the receptive field of any two nodes overlaps to a great extent.

Since, two nodes has a huge overlap in their corresponding receptive fields, it contributes to oversmoothing as the node embedding for a given node is dependent on the information being received from nodes within it's receptive fields. Therefore, two nodes with similar receptive fields will have similar node emebedding given that both nodes apply same aggregation strategy with equal weight given to all nodes present in their receptive field.

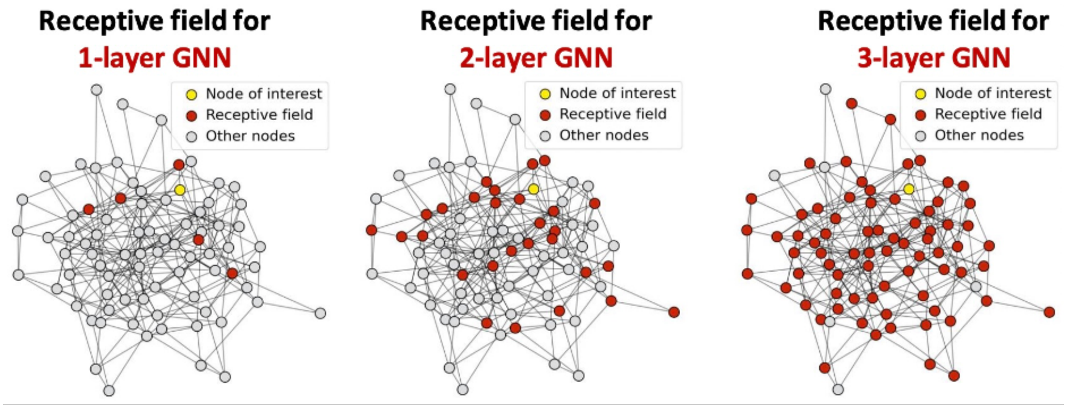

Receptive field for 1-layer GNN
Receptive field for 2-layer GNN
Receptive field for 3-layer GNN
Node of interest
Receptive field
Other nodes

Figure 4: Receptive field of nodes at different depths (Leskovec, 2021).

4.2.3 Aggregation strategy

After the receptive field is determined for a node, we have to figure out how to aggregate the information coming from all the nodes present in the receptive field. It also plays a critical role in determining the node embedding for a every node in GNNs.

Graph Isomorphism Network (GIN): A basic yet powerful model for graphs that generalizes the WL isomorphism test. It effectively use a sum aggregator rather than a max or mean aggregator to capture a graph's whole multi-set of nodes.

$$h_v^{(k)} = \mathbf{MLP}^{(k)}\left(\left(1+\epsilon^{(k)}\right)\cdot h_v^{(k-1)} + \sum_{u\in\mathcal{N}(v)} h_u^{(k-1)}\right)$$

Graph Convolutional Network (GCN): The aggregate and combine functions are combined in the following updating equation. The use of symmetric mean aggregation or normalization, rather than max pooling, is the main distinction from GraphSAGE (Li et al., 2018).

$$h_v^{(k)} = \mathbf{ReLU}\left(W^{(k)}\cdot \sum_{u\in\mathcal{N}(v)\cup\{v\}} \frac{1}{\sqrt{d_u d_v}} h_u^{(k-1)}\right)$$

where d_v represents the degree of node v

Graph Attention Network (GAT): This method uses a non-isotropic aggregation method in which each edge in the graph is given an importance score (or attention). This identifies how much surrounding nodes contribute to updating a node's embedding. The trained attentions can also be utilized to better evaluate and interpret the graph, which is another advantage of this method.

$$e_{uv}^{(k)} = \text{LeakyReLU}\left(a \cdot \text{CONCAT}\left[W^{(k)}h_u^{(k-1)}, W^{(k)}h_v^{(k-1)}\right]\right)$$
$$\alpha_{uv}^{(k)} = \text{SOFTMAX}_{u \in \mathcal{N}(v) \cup \{v\}}\left(e_{uv}^{(k)}\right)$$
$$h_v^{(k)} = \sigma\left(\sum_{u \in \mathcal{N}(v) \cup \{v\}} \alpha_{uv}^{(k)} W^{(k)} h_u^{(k-1)}\right)$$

Simple Graph Convolution (SGC): The non-linearity in each GCN layer is not significant, according to this research, and the majority of the advantage comes from neighborhood aggregation. As a result, the activation functions between layers are removed. In the weight parameters W, the resulting K-layer network becomes linear, although it has the same receptive field as a K-layer GCN.

$$h_v^{(k)} = W^{(k)} \cdot \sum_{u \in \mathcal{N}(v) \cup \{v\}} \frac{1}{\sqrt{d_u d_v}} h_u^{(k-1)}$$
$$\text{where } d_v \text{ represents the degree of node } v$$

GraphSAGE: One of the first papers to offer an inductive learning framework for graphs, that is, the capacity to generalize to unknown nodes during inference. The pooling aggregator is the most widely used choice for the aggregate function, according to the original publication. Every node embedding is transformed using a fully linked layer, and information from all neighbors is aggregated using an element-wise max-pooling process. The combine function concatenates the aggregate and prior embedding outputs before passing them via a fully connected layer with ReLU activation.

$$a_v^{(k)} = \text{MAX}\left(\left\{\text{ReLU}\left(W_{\text{pool}}^{(k)} \cdot h_u^{(k-1)}\right) : u \in \mathcal{N}(v)\right\}\right)$$
$$h_v^{(k)} = \text{ReLU}\left(W^{(k)} \cdot \text{CONCAT}\left[h_v^{(k-1)}, a_v^{(k)}\right]\right)$$

4.2.4 Node importance

Node importance is another concept which has the capability to relieve oversmoothing by modifying the node embeddings. Generally in Graph Neural Networks we give equal weight to the information coming from different nodes managed in the aggregation step. Node importance suggests to provide varying weight to the information coming from nodes present in receptive field based on node importance or priority instead of simply having equal weight for all nodes.

Node importance also helps in relieving the problem of oversquashing. Since, the number of nodes increases exponentially as the number of layers increase. If the number of nodes in receptive field are too high and if each node is given equal weight during the aggregation step, then the contribution of every single node becomes very minute, thereby leading to lack of uniqueness in node embeddings of all nodes at higher depth rendering them indistinguishable.

Node importance helps to solve the problem of oversquashing pretty well as it ensures not all nodes present in receptive field of a given node in GNN are simply provided equal weight in aggregation step. Rather, it works on creating node importance and provides a distinct weight to each node in alignment with the node's importance.

5. Measurement Metrics

In this section, we carried a quantitative analysis on the factors leading to oversmoothing in order to quantitatively prove the empirical analysis carried in the investigation stage (Chen et al., 2019).

5.1 Accuracy Score

We ran the experiment to calculate the validation accuracy score by running the GCN on Cora dataset. Similarly, we ran the same experiment at different depths or layers to observe the behavior between validation accuracy and number of layers as seen in Figure 5.

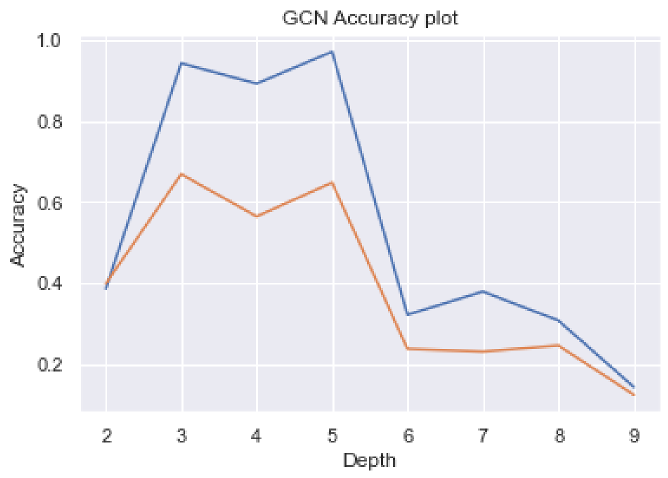


Figure 5: Line plot between validation accuracy score and number of layers.

In Figure 5, we can observe information gain without compromising distinctiveness in node embeddings from layer 2 till layer 5 which leads to improvement in performance. Although, from layer 6 onwards, the receptive field increases exponentially which leads to overlapping receptive fields for majority of the nodes, thereby rendering the node embedding of all nodes indistinguishable and making the problem of oversmoothing visible which is confirmed by the sharp accuracy drop in the plot from layer 5.

5.2 Cosine Similarity

Cosine Similarity is a metric that measures how similar two or more vectors are. The cosine of the angle between vectors is the cosine similarity. The vectors are usually non-zero and belong to an inner product space. Cosine similarity is determined using only the dot product and magnitude of each vector, and is thus only affected by the terms that the two vectors share, whereas Euclidean has a term for every non-zero dimension in either vector. Cosine, unlike Euclidean, offers some useful semantics for ranking related papers based on mutual term frequency.

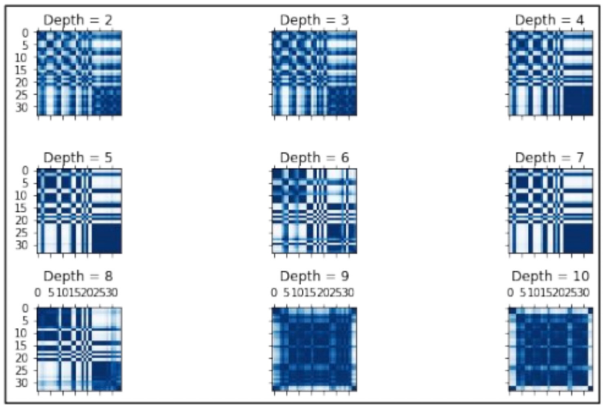


Figure 6: Correlation matrix between Cosine Similarity and Depth, Each plot shows the cosine similarity for all the potential pairs of nodes on the Karate Club Dataset

In Figure 6, we can clearly observe that the correlation in the cosine similarity of all the node pairs present in GNN increases to a point that they become indistinguishable as as the depth of GNN increases. This confirms the problem of oversmoothing studied empirically in investigation stage.

5.3 Mean Similarity Score

In Cosine similarity metric, we calculated the cosine similarity in two node embedding of the nodes present in all the distinct node pairs (in the given receptive field of a given GNN). Mean similarity score is the mean of the cosine similarity calculated for all node pairs in GNN.

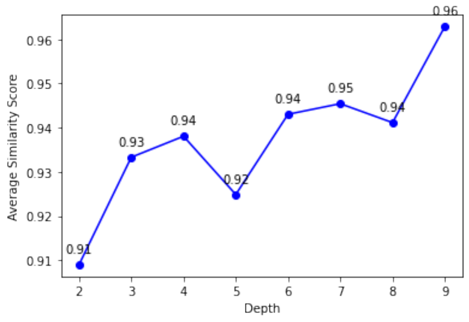


Figure 7: Plot between depth and average similarity score.

We can observe in Figure 7 that the average/mean similarity score is rising as we increase the depth. The empirical analysis done in investigation stage claimed that the accuracy decareases with increasing depth because as depth increases, the node embedding becomes indistinguishable, thereby leading to oversmoothing. Additionally, Figure 7 also suggests that the nodes are getting indistinguishable as the depth increases (due to increased similarity score), thereby confirming the claims made in investigation stage quantitatively.

5.4 Row difference

Row difference is another metric that helps us to calculate the overall similarity score considering all the distinct node pairs present in a given reception field. Mathematically, it is the squared mean of the difference in the Euclidean distance of the nodes from all the distinct pairs available in a given reception field.

$$\text{row-diff}(\mathbf{H}^{(k)}) = \frac{1}{n^2} \sum_{i,j \in [n]} \left\| \mathbf{h}_i^{(k)} - \mathbf{h}_j^{(k)} \right\|_2$$

After calculating the row difference as suggested above for the given graph with increasing depth or layers, we can observe clearly that the row difference/distance slumps after depth = 5. This sharp drop in row-difference after depth/layers = 5 confirms that the rows are losing their uniqueness and eventually becomes indistinguishable from each other leading to oversmoothing issues as claimed in the investigation stage.

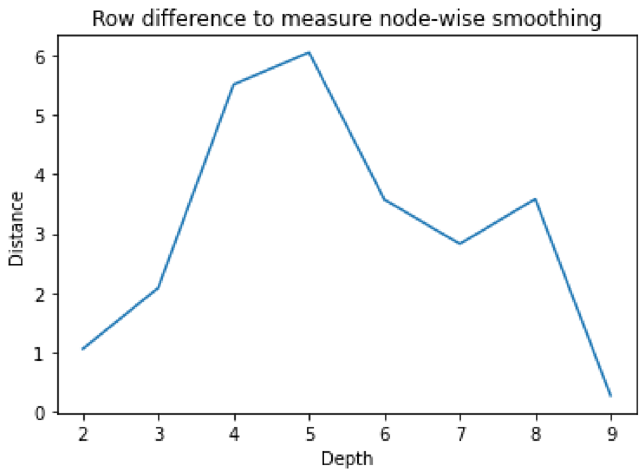


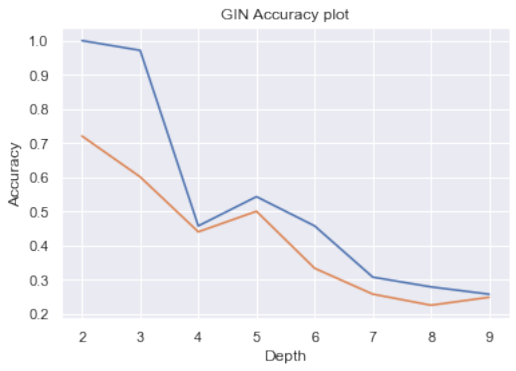
Figure 8: Plot between row difference and depth.

6. Mitigation strategies

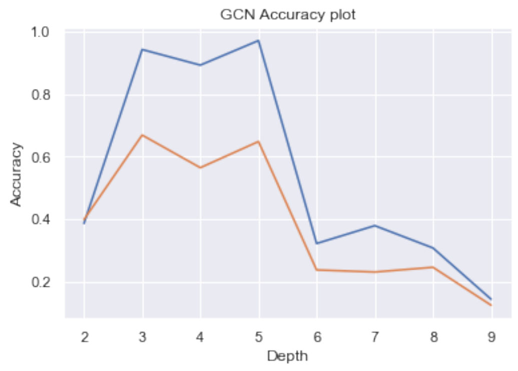
In previous section we did an empirical and a quantitative analysis on oversmoothing effect present in GNNs with increasing depth. In this section, we will explore and implement various mitigation strategies to maintain accuracy by relieving oversmoothing in GNNs.

6.1 Aggregation strategies

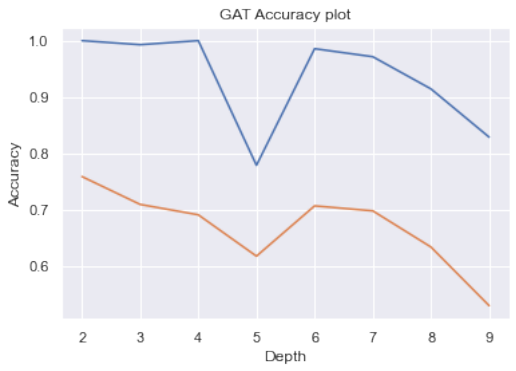
As we discussed in the Section 4.2.3 Aggregation Strategy which shared multiple variants of Graph Neural Network and their underlying reasoning. In this section we try to implement all those variants (as shared below) and discuss if different variants of GNN can maintain the accuracy and their caveats.



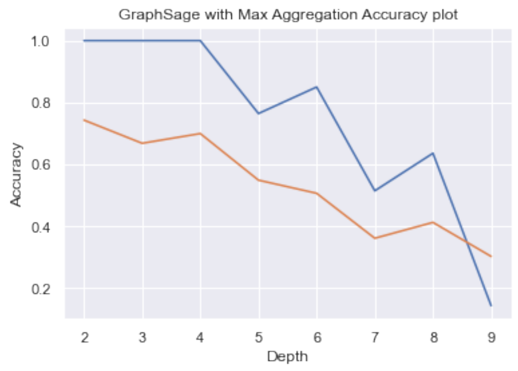
(a) Plot between Accuracy (Training and Validation) and Depth for GIN.



(b) Plot between Accuracy (Training and Validation) and Depth for GCN.



(c) Plot between Accuracy (Training and Validation) and Depth for GAT.



(d) Plot between Accuracy (Training and Validation) and Depth for GraphSage.

Figure 9: Plots between accuracy and number of layers for all variants of GNN.

We can make some observations from Figure 9 which shows the plots between accuracy and depth for Graph Isomorphism Networks (GIN), Graph Convolutional Networks (GCN), Graph Attention Networks (GAT) and GraphSage. Here, we can see that the training and validation accuracy beyond depth of 2 for GIN, GCN and GAT falls faster than GraphSage as the depth increases.

We know that GIN, GCN and GAT utilizes the mean aggregation strategy whereas GraphSage utilizes the max aggregation strategy which is the key difference GraphSage when compared with others. Therefore, GraphSage just keeps the maximum node embedding among its and its neighbors node embedding whereas GIN, GCN and GAT takes the summation of all input node embedding to calculate the node embedding for a given node in GNN.

Even though it leads to information loss in GraphSage as the node embedding of only one incoming node (which is maximum) is accepted by GraphSage, it improves the accuracy as the randomization is increased during the aggregation step which eventually helps to maintain the accuracy by relieving the oversmoothing.

We can also understand from the superior performance of the GraphSage that having distinct weights of incoming nodes works better than taking the equal weights of all incoming nodes which happens in mean aggregation strategy. We can also conclude from this study that aggregation strategy is very critical in maintaining the accuracy by relieving the oversmoothing as the depth increases.

6.2 Normalization (PairNorm)

PairNorm is yet another promising design which works well in relieving oversmoothing by normalizing total pairwise distance.

To avoid node-wise oversmoothing, keep the total pairwise squared distance (TPSD) constant across layers. When combined with GraphSage with max aggregation strategy, non-connected pairs can be pushed away (Zhao and Akoglu, 2020), mathematically,

$$\text{TPSD}(\dot{\mathbf{X}}) := \sum_{i,j\in[n]} \|\dot{\mathbf{x}}_i - \dot{\mathbf{x}}_j\|_2^2$$

Our goal is to ensure that,

$$\text{TPSD}(\mathbf{X}) = \text{TPSD}(\dot{\mathbf{X}})$$

Now, we carry out the Center and Scale operations on PairNorm as,

$$\tilde{\mathbf{x}}_i^c = \tilde{\mathbf{x}}_i - \frac{1}{n}\sum_{i=1}^{n}\tilde{\mathbf{x}}_i \qquad \text{(Center)}$$

$$\dot{\mathbf{x}}_i = s \cdot \frac{\tilde{\mathbf{x}}_i^c}{\sqrt{\frac{1}{n}\sum_{i=1}^{n}\|\tilde{\mathbf{x}}_i^c\|_2^2}} = s\sqrt{n} \cdot \frac{\tilde{\mathbf{x}}_i^c}{\sqrt{\|\tilde{\mathbf{X}}^c\|_F^2}} \qquad \text{(Scale)}$$

After the operations $\dot{\mathbf{X}}$ has the row-wise mean of 0 which confirms it is now centered and $\text{TPSD}(\dot{\mathbf{X}}) = 2n||\dot{\mathbf{X}}||_F^2 = 2n^2 s^2$, where s is a hyperparameter.

The complete procedure for PairNorm can be visualized as,

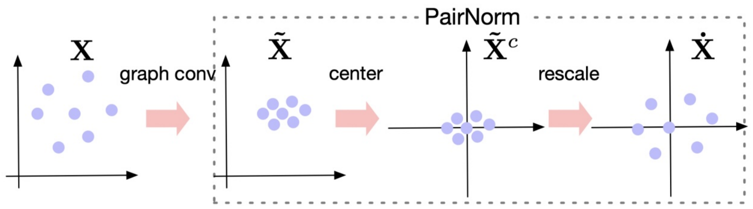
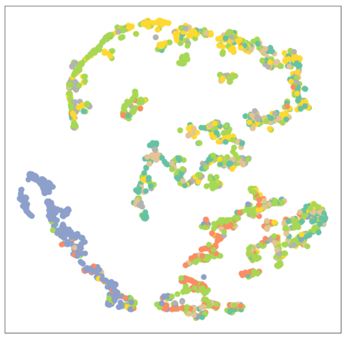
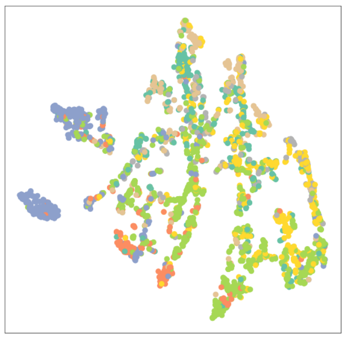


Figure 10: Visualizing the operations in PairNorm

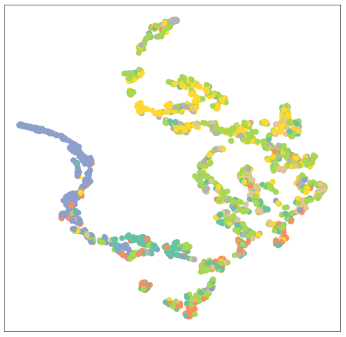
We ran the experiment for PairNorm on Cora dataset and we received the following results as seen in Figure 11.



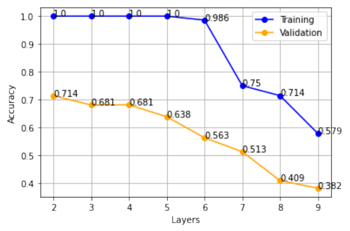
(a) Pair Norm Node Embeddings



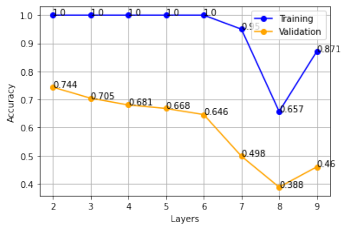
(b) Pair Norm Scaled-Individually Node Embeddings



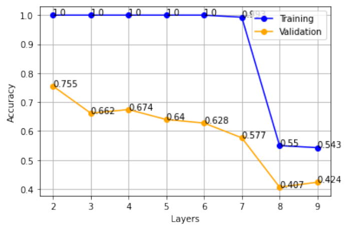
(c) Pair Norm Scaled-and-Centered-Simultaneously Node Embeddings



(d) Plot between Accuracy (Training and Validation) and Depth for Pair Norm



(e) Plot between Accuracy (Training and Validation) and Depth for Pair Norm Scaled Individually



(f) Plot between Accuracy (Training and Validation) and Depth for Pair Norm Scaled and Centered Simultaneously

Figure 11: Node embeddings and plots between accuracy and number of layers for different versions of Pair Norm.

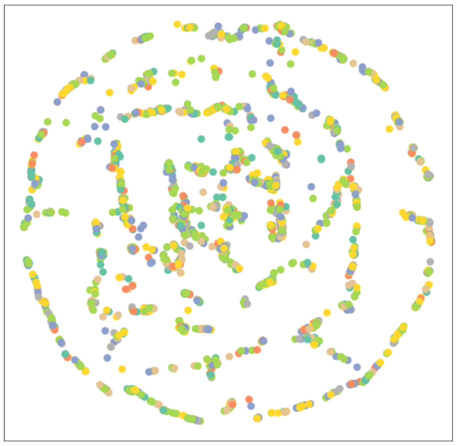
We can observe from Figure 11 that different versions of Pair Norm maintained accuracy at larger depths compared to the techniques implemented so far. Among Pair Norm techniques, Pair Norm Scaled-and-Centered-Simultaneously version maintained the accuracy the best up to the maximum depth of 7.

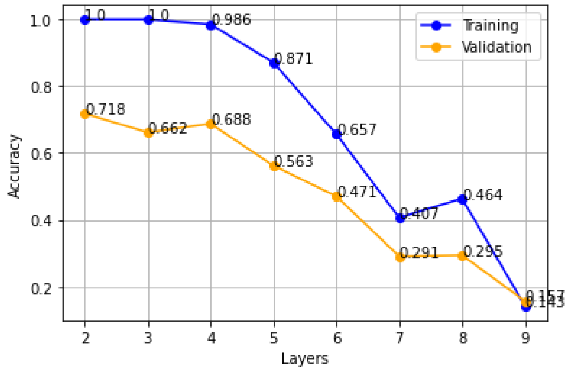
6.3 Sampling Method (DropEdge)

DropEdge is a randomized sampling method which samples neighbors at random by dropping certain percentage (dropout rate) of neighbors. This induces randomization in the sampling method which results in reduction in overlap of receptive field of the neighboring nodes in the graph. This acts as a data augmenter and also a message passing reducer. We applied the DropEdge Sampling to GraphSage with *max* aggregation (Rong et al., 2020).

By dropping out a certain rate of edges by random, DropEdge includes more diversity into the input data to prevent over-fitting, and reduces message passing in graph convolution to alleviate over-smoothing.

We can observe in Figure 12 that due to induced randomization with Drop Edge, our model is performing better than baseline models.


(a) Drop Edge embeddings

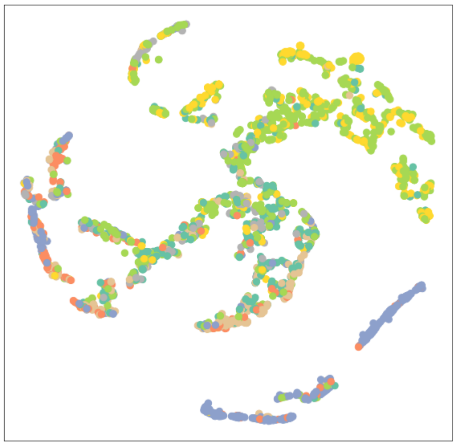


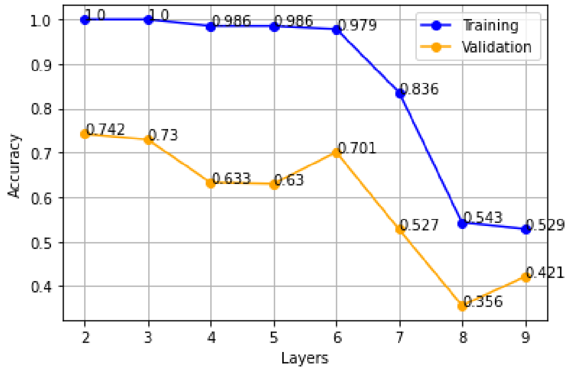
(b) Plot between Accuracy (Training and Validation) and Depth

Figure 12: Node embeddings and plots between accuracy and number of layers for Drop Edge with drop rate 0.5

6.4 Normalization + Sampling Method (PairNorm + DropEdge)

In this strategy, we are inducing randomization along with PairNorm normalization. This randomization prevent the model from overfitting. Additionally, it performs better than the baseline models as seen in Figure 13.

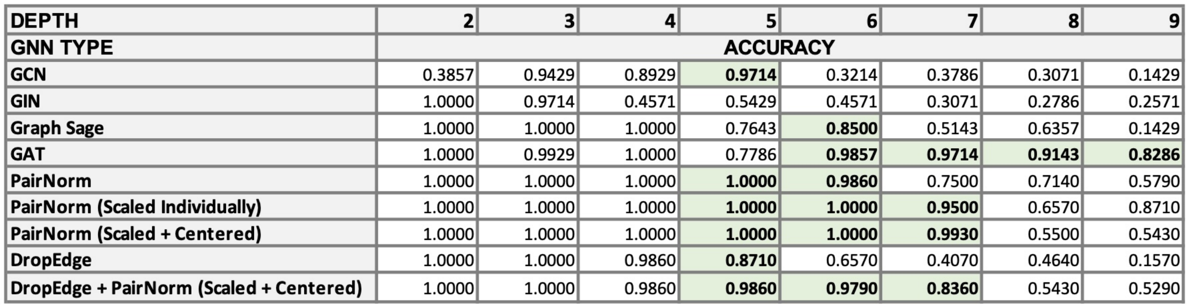

(a) Drop Edge + Pair Norm embeddings



(b) Plot between Accuracy (Training and Validation) and Depth

Figure 13: Node embeddings and plots between accuracy and number of layers for Drop Edge with drop rate 0.5 + PairNorm

7. Results



DEPTH	2	3	4	5	6	7	8	9
GNN TYPE	ACCURACY							
GCN	0.3857	0.9429	0.8929	**0.9714**	0.3214	0.3786	0.3071	0.1429
GIN	1.0000	0.9714	0.4571	0.5429	0.4571	0.3071	0.2786	0.2571
Graph Sage	1.0000	1.0000	1.0000	0.7643	**0.8500**	0.5143	0.6357	0.1429
GAT	1.0000	0.9929	1.0000	0.7786	**0.9857**	**0.9714**	**0.9143**	**0.8286**
PairNorm	1.0000	1.0000	1.0000	**1.0000**	**0.9860**	0.7500	0.7140	0.5790
PairNorm (Scaled Individually)	1.0000	1.0000	1.0000	**1.0000**	**1.0000**	**0.9500**	0.6570	0.8710
PairNorm (Scaled + Centered)	1.0000	1.0000	1.0000	**1.0000**	**1.0000**	**0.9930**	0.5500	0.5430
DropEdge	1.0000	1.0000	0.9860	**0.8710**	0.6570	0.4070	0.4640	0.1570
DropEdge + PairNorm (Scaled + Centered)	1.0000	1.0000	0.9860	**0.9860**	**0.9790**	**0.8360**	0.5430	0.5290

Figure 14: Accuracy vs Depth for different GNNs

As per the initial hypothesis, we suspected oversmoothing as the major underlying factor behind the the accuracy drop. We identified several key factors that could cause oversmoothing, namely, Sampling Strategy, Receptive Field, Aggregation Strategy, Node Importance and Normalization. We ran several experiments to test our hypothesis and to determine the impact of these factors in accuracy drop. Further, we tried to mitigate the problem of oversmoothing by observing and changing these parameters.

For sampling strategy and receptive field, we used DropEdge sampling strategy to induce randomization leading to different receptive fields in neighboring nodes. We observed the improvement in accuracy compared to baseline model.

For aggregation strategy, we used several variants of Graph architectures(GCN, GIN, and GraphSage) and aggregation. We observed the most improvement in performance for GraphSage with max aggregation strategy.

For Node Importance strategy, we utilized Graph Attention Network architecture to sample neighboring nodes based on node importance which gave best results as compared to all other tested models. Aggregating information from neighbors based on their significance in their locality plays an important role.

For Normalization, we implemented different versions of Pair Norm normalization(Pair Norm, Pair Norm Scaled-Individually, and Pair Norm Scaled-and-Centered-Simultaneously), which performed better than the baseline models.

From the above experiments, we observed that handling one or several of these key factors and making strategies that attempt to improve the model based on the combination of these factors could lead to maintaining and improving accuracy for larger depths.

8. Conclusion

In this study, we address and explore solutions for the accuracy drop with increasing depth problem that exists in graph neural networks. We take empirical and quantitative approach in investigating potential reasons behind the accuracy drop. After establishing and proving oversmoothing and oversquashing as causes for accuracy drop, we conducted several experiments and successfully devised multiple mitigation strategies that prove it is possible to maintain the accuracy in Graph Neural Networks with increasing depth.

9. Future Scope

- **Random + Importance Sampling**: In the mitigation strategies, we employed the DropEdge random sampling strategy. Additionally, we are planning to devise a sampling strategy focused on utilizing the power of both random and importance sampling. We'll be using node centrality measure to compute node importance and thereby find the good neighborhood for a node in importance sampling.
- **Finding Good Neighborhood**: While in homophily graphs, the adjacent neighbors are important and informative, there are several scenarios and applications where non-neighboring nodes can have critical information. For example, if we consider the people on social media as nodes in a graph, and topics and channels they follow as node attributes. A person can have immediate connections that are close in relationship yet non-informative as they don't have the similar interests but have a second degree connection that non-adjacent yet more informative as they have similar interests, therefore could add relevant information. In such a case finding good neighborhood is important.
- **Skip Connection**: At larger depths, there could be nodes that have relevant information but there could several nodes in its path that do not. In such a case, we can use skip connections to skip such non-informative nodes in the path.

10. Code Repository: ml2565.git

Acknowledgments

We would like to thank Prof. Rajesh Ranganath and Mark Goldstein for helping us to understand Graph Neural Networks by guiding us post lectures and over mail to resolve the roadblocks. This project was carried out as a part of the curriculum for Machine Learning course at NYU Courant Institute of Mathematical Science.

References

Uri Alon and Eran Yahav. On the bottleneck of graph neural networks and its practical implications. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=i80OPhOCVH2`.

Chen Cai and Yusu Wang. A note on over-smoothing for graph neural networks. *arXiv preprint arXiv:2006.13318*, 2020.

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. URL `https://arxiv.org/abs/1909.03211`.

Jurij Leskovec. Cs224w: Machine learning with graphs, fall 2021, 2021. URL `http://web.stanford.edu/class/cs224w`.

Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning, 2018. URL `https://arxiv.org/abs/1801.07606`.

Meng Liu, Hongyang Gao, and Shuiwang Ji. Towards deeper graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery &amp Data Mining*. ACM, aug 2020. doi: 10.1145/3394486.3403076. URL `https://doi.org/10.1145%2F3394486.3403076`.

Kenta Oono and Taiji Suzuki. Graph neural networks exponentially lose expressive power for node classification. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=S1ldO2EFPr`.

Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. Dropedge: Towards deep graph convolutional networks on node classification. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=Hkx1qkrKPr`.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks?, 2018. URL `https://arxiv.org/abs/1810.00826`.

Yufan Zeng and Jiashan Tang. Rlc-gnn: An improved deep architecture for spatial-based graph neural network with application to fraud detection. *Applied Sciences*, 11(12), 2021. ISSN 2076-3417. doi: 10.3390/app11125656. URL `https://www.mdpi.com/2076-3417/11/12/5656`.

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