REDUCING OVERSMOOTHING IN GRAPH NEURAL NETWORKS BY CHANGING THE ACTIVATION FUNCTION

Anonymous authors
Paper under double-blind review

ABSTRACT

The performance of Graph Neural Networks (GNNs) deteriorates as the depth of the network increases. That performance drop is mainly attributed to oversmoothing, which leads to similar node representations through repeated graph convolutions. We show that in deep GNNs the activation function plays a crucial role in oversmoothing. We explain theoretically why this is the case and propose a simple modification to the slope of ReLU to reduce oversmoothing. The proposed approach enables deep architectures without the need to change the network architecture or to add residual connections. We verify the theoretical results experimentally and further show that deep networks, which do not suffer from oversmoothing, are beneficial in the presence of the "cold start" problem, i.e. when there is no feature information about unlabeled nodes.

1 INTRODUCTION

Graph Neural Networks (GNNs) utilize message passing or neighborhood aggregation schemes to extract representations for nodes and their neighborhoods. GNNs have achieved good results on a variety of graph analytics tasks, such as node classification (Zhang et al., 2018; Hamilton et al., 2017), link prediction (Liben-Nowell & Kleinberg, 2003; Zhang & Chen, 2018) and graph classification (Klicpera et al., 2020). As a result, they play a key role in graph representation learning. One of the most prominent GNN models is the Graph Convolutional Network (GCN) (Kipf & Welling, 2017), which creates node representations, by averaging the representations (embeddings) of its immediate neighbors. Several studies have shown that, the performance of GCNs deteriorates, when their architecture becomes deeper (Li et al., 2018).

The success of deep CNNs on many tasks, like image classification, naturally led to several attempts towards building deep GNNs for node classification (Kipf & Welling, 2017; Li et al., 2018; Xu et al., 2018). Increasing the model's depth (and the number of parameters it has), would allow more accurate representational learning to occur. Most of the existing approaches have failed to develop a sufficiently deep architecture that achieves good performance. Therefore, there is a need to design new models which can efficiently scale to a large number of layers. The aim of this paper is to investigate the contributing factors, that compromise the performance of deep GNNs and develop a method to address them.

The performance drop of deep GNNs, is associated with several factors, including vanishing gradients, overfitting, as well as the phenomenon called oversmoothing (Li et al., 2018; Xu et al., 2018; Klicpera et al., 2018; Wu et al., 2019). Oversmoothing has been shown to be associated with graph convolution, a type of Laplacian operator. Li et al. (2018) proved that applying that operator repeatedly, makes node representations converge to a stationary point. At that point, all of the initial information (i.e. node features' information) is lost through the Laplacian smoothing. Consequently, oversmoothing hurts the performance by making node features indistinguishable across different classes.

In this work, we address the oversmoothing problem in deep GNNs. We provide what is to the best of our knowledge the first study regarding the role of the activation function and the learning rate per layer of the model to oversmoothing and propose a new method to address the problem. We confirm our hypotheses both experimentally and theoretically. The new method is shown to prevent node embeddings from converging to the same point, thus leading to better node representations. We summarize our main contributions as follows.

•**Role of Activation Function in Oversmoothing**: We prove theoretically the connection between

the activation function and oversmoothing. In fact, we show the relation between the slope of ReLU and the singular values of weight matrices, which are known to be associated with oversmoothing (Oono & Suzuki, 2020; Cai & Wang, 2020). We have also verified our theoretical results experimentally.

•**Role of Learning Rate in Oversmoothing**: Our analysis on the effect of the slope of ReLU to oversmoothing has a direct extension to the learning rates used per layer of the network. We conducted further experiments to study the effect of tuning the learning rates, showing that this approach could also reduce oversmoothing, but it is less practical.

•**The power of Deep GNNs:** We have performed extensive experiments using up to 64-layer networks, tackling oversmoothing with the proposed method. We further show the benefits that such deep GNNs can provide in the presence of reduced information, such as in a "cold start" situation, where node features are available only for the labeled nodes in a node classification setting.

2 Notations and Preliminaries

2.1 Notations

In order to illustrate the problem of oversmoothing, we consider the task of semi-supervised node classification on a graph. The graph to be analysed is $\mathcal{G}(\mathbb{V}, \mathbb{E}, \mathrm{X})$, with $|\mathbb{V}| = N$ nodes $u_i \in \mathbb{V}$, edges $(u_i, u_j) \in \mathbb{E}$ and $X = [x_1, ..., x_N]^T \in \mathbb{R}^{N \times C}$ denotes the initial node features. The edges form an adjacency matrix $A \in \mathbb{R}^{N \times N}$ where edge (u_i, u_j) is associated with element $A_{i,j}$. $A_{i,j}$ can take arbitrary real values indicating the weight (strength) of edge (u_i, u_j). Node degrees are represented through a diagonal matrix $D \in \mathbb{R}^{N \times N}$, where each element d_i represents the sum of edge weights connected to node i. During training, only the labels of a subset $V_l \in \mathbb{V}$ are available. The task is to learn a node classifier, that predicts the label of each node using the graph topology and the given feature vectors.

GCN, originally proposed by Kipf & Welling (2017), utilizes a feed forward propagation as:

$$H^{(l+1)} = \sigma(\hat{A} H^{(l)} W^{(l)}) \tag{1}$$

where $H^{(l)} = [h_1^{(l)}, ..., h_N^{(l)}]$ are node representations (or hidden vectors or embeddings) at the l-th layer, with h_i^l standing for the hidden representation of node i; $\hat{A} = \hat{D}^{-1/2}(A+I)\hat{D}^{-1/2}$ denotes the augmented adjacency matrix after self-loop addition, where $\hat{D}$ corresponds to the degree matrix; $\sigma(\cdot)$ is a nonlinear element-wise function, i.e. the activation function, ReLU; and $W^{(l)}$ is the trainable weight matrix of the l-th layer. A generic ReLU function with slope α is defined as ReLU(x) = $max(\alpha \cdot x, 0)$.

2.2 Understanding Oversmoothing

GNNs achieve state-of-the-art performance in a variety of graph-based tasks. Despite their success, models like GCN (Kipf & Welling, 2017) and GAT (Velickovic et al., 2018) experience a performance drop, when stacking multiple layers. To a large extent, this is attributed to oversmoothing due to repeated graph convolutions. In their analysis Li et al. (2018), showed that graph convolution is a special form of Laplacian smoothing. In fact, they proved that the new representation of each node is formed by a weighted average of its own representation and that of its neighbors. This mechanism allows the node representations within each (graph) cluster, i.e. highly connected group of nodes, to become more similar and improves the performance on semi-supervised tasks on graphs. When stacking multiple layers, the smoothing operation is repeated multiple times leading to oversmoothing of node representations, i.e., the hidden representations of all nodes become similar, resulting in information loss.

2.3 Deep GNN limitations

Therefore, oversmoothing leads node representations to converge to a fixed point as the network's depth increases (Li et al., 2018). At that point, node representations contain information relevant to the graph topology and disregard the input features. Oono & Suzuki (2020) have generalized the idea in Li et al. (2018) by taking into consideration, that the ReLU activation function maps to a positive cone. They explain oversmoothing as the convergence to a subspace, instead of the

convergence to a fixed point. A similar approach is presented in (Cai & Wang, 2020), offering a different perspective to the oversmoothing problem using Dirichlet energy.
We now look closer to the proposal of Oono & Suzuki (2020), which we will use as a basis for our analysis. The main result in (Oono & Suzuki, 2020) is the definition of the convergence speed towards a subspace M, where the distance between node representations tends to zero. We denote as $d_M(X)$ the distance between the feature vector X and the subspace, where the oversmoothing is prevalent. When that distance approaches to zero it indicates that node representations have been over-smoothed. For this distance, Oono & Suzuki (2020) prove an interesting property.

Theorem 1 (Oono & Suzuki (2020)) *Let the largest singular value of the weight matrix W_{lh} be s_{lh} and $s_l = \prod_{h=1}^{H_l} s_{lh}$, where W_{lh} is the weight matrix of layer h and H_l be the network's depth, following the notation of the paper. Then it holds that $d_M(f_l(X)) \leq s_l d_M(X)$ for any $X \in \mathbb{R}^{N \times C}$, where f(·) is the forward pass of a GNN layer (i.e. $\sigma(AXW_{lh})$).*

The above theorem indicates, that the deeper the network the smaller the distance of node representations from the subspace M. If the maximum singular values are small then node representations asymptotically approach M, for any initial values of node features. Extending the above theorem, the authors conclude to the following estimate about the speed of convergence to the oversmoothing subspace.

Corollary 1 (Oono & Suzuki (2020)) *Let $s = sup_{l \in N_+} s_l$. then $d_M(X^{(l)}) = O((s\lambda)^l)$, where l is the layer number and if $s\lambda < 1$ the distance from oversmoothing subspace exponentially approaches zero. Where λ is the smallest non-zero eigenvalue of I - $\hat{A}$.*

According to the authors, sufficiently deep GCN will certainly suffer from oversmoothing under some conditions (details can be found in Oono & Suzuki (2020)). We build upon this result, aiming to develop a consistent approach that reduces oversmoothing and enables deep architectures.

2.4 Normalization in deep neural networks

Of relevance to oversmoothing are also methods that perform normalization for deep neural networks (Ioffe & Szegedy, 2015; Ba et al., 2016). In such methods, the output of each neuron is normalized, in order to keep a portion of the initial feature variance. Other recent work proposed Self Normalized Networks (SNN), that use a different activation function (SeLU) (Klambauer et al., 2017). These models perform self normalization inside each neuron, in order to keep the variance between consecutive layers stable. They have managed to enable deep Fully Connected models and achieve good performance.
A recent work regarding normalization in GNNs is the Pairnorm method (Zhao & Akoglu, 2020), which aims to keep constant the total pairwise distance between node representations. Compared to SNNs, Pairnorm performs normalization needing a constant value as hyper-parameter determined per dataset, while SNNs manage to normalize the output utilizing their activation function, which can be used in a family of neural networks. The activation function proposed in (Klambauer et al., 2017) (i.e. SeLU) has a saturating region to reduce the variance of data and a slope slightly greater than 1 in order to increase the data variance, when needed.

3 Understanding and dealing with oversmoothing

Using the mathematical definition of oversmoothing in Oono & Suzuki (2020), in this section we establish a connection with the training process of GNNs. In particular, we analyse the role of the activation function and the learning rate and we propose modified versions of GNNs to address the issue.

3.1 Theoretical Analysis

We start by establishing the connection between oversmoothing and variance reduction of node representations. Consider oversmoothing as the convergence to a subspace, where node representations

are almost the same. In that particular subspace, the initial variance of feature vectors has been massively reduced. This unwanted oversmoothing effect is 'harsh', due to the fact that ReLU performs a zero mapping, when the input is negative. We use the term 'harsh' to indicate the smoothing case, where representations are mapped to a single point (i.e. zero), instead of converging to a subspace. We use the notation of 'converging to a subspace' following the oversmoothing analysis presented in Oono & Suzuki (2020), where the subspace M (where oversmoothing is prevalent) is defined.
We will now show how the slope of the ReLU activation function affects oversmoothing, starting with some important assumptions: (a) the non-exploding gradients assumption, and (b) ReLU's probability not to output zero. We use these assumptions to extract bounds on the weights of the network and then use these bounds to determine the relationship between the largest singular value and the elements of each weight matrix. Given that relationship, we connect our analysis with existing literature regarding oversmoothing through Theorem 1.

Assumption 1 *For each layer l there exists a number G_l which is the upper bound to the gradients of the output of the subsequent layer (l+1) with respect to the weight elements of $W^{(l)}$, i.e.*

$$\frac{dO^{(l+1)}}{dw^{(l)}_{old_{i,j}}} \leq G_l, \qquad \forall w^{(l)}_{old_{i,j}} \tag{2}$$

where $O^{(l+1)}$ is the output of (l+1)-th layer.

This assumption needs to hold in all cases, in order to avoid the exploding gradient case, that would not allow the learning process to converge.

Assumption 2 *The probability, that the output of the ReLU function not equals zero at layer l is independent of the outputs of the previous layers, in a feed-forward ReLU neural network, i.e.*

$$P(ReLU(h^{(l)}) \neq \vec{0} | \forall h^{(j)}, j < l) \leq p, \qquad p \in [0, 1) \tag{3}$$

where $h^{(l)}$ is the node representation at layer l, ReLU is applied piece-wise on that representation and $\vec{0}$ is the all zero vector.

Following the results of Lu et al. (2019) we get that $P(ReLU(h^{(l)}) \neq \vec{0} | \forall h^{(j)}, j < l) = 1/2$ for feed-forward networks (FFs), on the condition that the output of the previous layers is positive. That condition does not necessary hold in GNNs, due to the role of the adjacency matrix in the aggregation scheme. In FFs, when the output of a layer is zero, so are the outputs of all subsequent layers. In GNNs there might be cases, where the output of layer l (a node representation) is zero but the aggregation using the adjacency matrix produces a non-zero representation in layer (l+1). Apart from this difference, GNNs can be considered a special case of FFs, with the proper wiring due to the adjacency matrix aggregation. Therefore, the above assumption holds also for GNNs.

Lemma 1 *For a network of depth dep the total gradient reaching to the l-th layer $\left(\text{i.e., } \frac{dJ}{dw^{(l)}_{old_{i,j}}}\right)$ in order to update $W^{(l)}$ is bounded by:*

$$\frac{dJ}{dw^{(l)}_{old_{i,j}}} \leq \alpha^{(dep-l)} \cdot G_l \tag{4}$$

Where J is the model's loss function (i.e. Cross Entropy), α stands for the ReLU slope and G_l is the upper bound of gradients of the output of the subsequent layer (l+1) with respect to the weight elements of $W^{(l)}$.

The proof of Lemma 1 is shown in Appendix A and is based on the chain rule for backpropagation. Given the derivative of the ReLU function and the upper bound G_l of Assumption 1, we can derive the bound of Equation 4. To compute the gradient with respect to weight element $w^{(l)}_{i,j}$ of layer l, we repeatedly differentiate nested ReLU functions, leading to a product of their slopes (or zero). In the final differentiation step, we get the gradient of the output of (l+1)-th layer with respect to $w^{(l)}_{i,j}$, which is bounded by G_l.

Lemma 2 *While model's loss, through gradients, flows backwards some weight elements do not receive updates, because we have dying ReLUs (i.e. ReLUs outputting zero) (Lu et al., 2019). The total number of weight elements getting updated at layer l is bounded by:*

$$\#\{w_{i,j}^{(l)}\} \leq p^{(dep-l)} \cdot d^2 \tag{5}$$

where d is the largest of the two dimensions of $W^{(l)}$, i.e. there are at most d^2 elements in $W^{(l)}$, if it is a square matrix.

The proof of Lemma 2 appears in Appendix B. Given Lemma 1, the gradient flowing backwards, with respect to a weight element $w_{i,j}^{(l)}$ contains a product of ReLU derivatives. In order for $w_{i,j}^{(l)}$ to get an update, all ReLU derivatives need to be non-zero, otherwise the gradient will be zeroed and no update will be applied to $w_{i,j}^{(l)}$. Additionally, due to Assumption 2 weight elements located in the lower weight matrices tend to receive fewer updates. This is due to the fact, that the further the gradient flows backwards the more ReLU derivative factors appear in it. Based on Assumption 2, the probability of all of them to be non-zero decreases.
Regarding the singular values (denoted by $s_l(\cdot)$) of a weight matrix $W^{(l)}$ at the l-th layer, the largest of them is given by: $max(s_l(W^{(l)})) = ||W^{(l)}||_2 \leq ||W^{(l)}||_F = \sqrt{\sum |w_{i,j}^{(l)}|^2}$, where $|| \cdot ||_F$ is the Frobenius norm and $|| \cdot ||_2$ is the spectral norm. Using the general weight updating rule $\left(\text{i.e., } w_{new_{i,j}}^{(l)} = w_{old_{i,j}}^{(l)} + \eta \cdot \frac{dJ}{dw_{old_{i,j}}^{(l)}}\right)$, where J is the model's loss and η is the learning rate, we arrive at the main theorem of this work.

Theorem 2 *The upper bound of the largest singular value of the weight matrix $W^{(l)}$ at layer l for a GNN model, utilizing a ReLU activation function, depends on the slope of the function. That bound is given per layer and shows the effect of each iteration of updates on the weight matrix. We denote with W_{old} and W_{new} the weight matrices before and after the update during an iteration of the training process respectively.*

$$max(s_l(W_{new}^{(l)})) \leq ||W_{old}^{(l)}||_F + \sqrt{3} \cdot p^{\left(\frac{dep-l}{2}\right)} \cdot d \cdot \alpha^{(dep-l)} \cdot B_l \tag{6}$$

where $B_l = \eta \cdot G_l$, dep is the network's depth, d is the largest dimension of the $W^{(l)}$ matrix, p is the upper bound of the probability of ReLU not to output zero and α is ReLU's slope.

The proof of Theorem 2 appears in Appendix C and uses the upper bound of the largest singular value by the Frobenius norm, expanded according to the weight update rule. Separating weight elements into two sets (updated and not updated) we identify the gradient values responsible for the weight updates. These values are bounded (Lemma 1), as is the number of updated elements (Lemma 2), leading to Equation 6. The resulting upper bound shows the connection of the slope of ReLU and the upper bound of the probability of ReLU not to output zero with the largest singular value of the weight matrix. That largest singular value is connected with the oversmoothing problem and as we mentioned above could act as a resource to reduce it.

3.2 Alleviating Oversmoothing

We transfer the idea of SeLU to GNNs, focusing on the part that increases the variance, because the repetition of the Laplacian operator acts as a variance reducer. In order to avoid oversmoothing in deep GNNs using this approach, we need to identify a 'sweet' spot, where variance reduction from graph convolution counteracted as needed by the slope of the activation function. Typically in GNNs, ReLU is used with a slope value $\alpha = 1$. As a result, the second exponential factor in Equation 6 can be ignored (it is always equal to 1). This pushes the bound for the largest singular value of weight matrices towards a fixed low value, when the architecture of the network gets deeper (dep increases), and especially at the lower layers (small l) of the network. This is due to the first exponential factor of (6) converging to zero for large values of dep.
The restriction of the largest singular values to low values, increases the speed of convergence to the oversmoothng subspace, as stated in Corollary 1 from Oono & Suzuki (2020). According to the corollary, the speed depends on λ and s. The former parameter (λ) is a property of the data (largest eigenvalue of the adjacency matrix), while the latter (s) is the product of the largest singular values

of the weight matrices.
Realizing the importance of the slope of ReLU in the training process, we move on to propose a modification of the activation function that reduces oversmoothing. Following (Lu et al., 2019), we proceed our analysis with $p = 1/2$ (see Assumption 2), which simplifies calculations. In particular, we observe that a slope of 2 makes the second exponential factor prevail over the first one, leading to a new combined factor of $2^{\left(\frac{dep-l}{2}\right)}$. This in turn increases the upper bound for the largest singular value, restricting the influence of the layer index (l) and depth (dep) in Equation 6. It is worth noting, that the upper bound should not be constant across all layers (which could be achieved by setting a corresponding value for the slope), because different layers of the network have different roles and the goal of a GNN is to bring intra-class representations close, while keeping inter-class representations apart.
Using the results of (Oono & Suzuki, 2020) we can further connect the choice of slope for the activation function to the speed of convergence to the oversmoothing subspace. Hence, the proposed method increases the bound for each largest singular value, which in turn decreases the speed of convergence to the oversmoothing subspace, according to Corollary 1.

3.3 Modifying the slope of ReLU: Limitations

Oono & Suzuki (2020) have proved that any GNN deep enough will eventually reach the oversmoothing subspace. In theory, this also holds for our method, when the GNN gets very deep. Could this be avoided by increasing the slope of ReLU further? Unfortunately not, as we cannot make the slope too large. It is known, that by increasing the slope of ReLU too much, one may face the problem of exploding gradients (Bengio et al., 1994), that impedes the learning procedure. In fact, if we increase the slope of ReLU too much, the upper bound in Equation 6 becomes too loose and the performance degrades. Our proof suggests, that a slope equal to 2 avoids oversmoothing and our experiments have shown that it also avoids the exploding gradients problem.

3.4 Modifying the learning rate, instead of the slope of ReLU

Instead of changing the slope of ReLU, we could opt to modify the Learning Rate (LR) per layer in order to increase the upper bound in Equation 6. In particular, a different learning rate (η) per layer leads to a different B_l (= $\eta \cdot G_l$) per layer, counteracting the problematic first exponential factor of Equation 6.
However, determining the right LR per layer is a non-trivial task. Intuitively, LR should be larger in the lower layers and smaller in the upper ones. Keeping the slope equal to 1, the first problematic exponential term is smaller (note $p < 1$) in the lower layers reducing the value of the upper bound. To avoid this, one needs to tune LR carefully. Additionally, LR affects heavily the learning process and large LR values might lead to oscillations and poor learning performance. Modifying the slope and the learning rate combined led to negligible improvements. Preliminary results with modified LR values are shown in Appendix D.

3.5 Why do we need deep GNNs?

Oversmoothing is only a problem for deep GNNs, leading to the question of whether and when deep GNNs are really needed. Most of the existing benchmark datasets for various graph analytics tasks do not seem to justify the need for deep networks. Due to their homophilic nature, useful information for each node resides in its close neighbors (usually 2 or 3 hops away). A task where deeper architectures could be needed is the "cold start" problem; namely the situation where many node features are missing. The problem of missing features is called "cold start", because it resembles the situation of a new product/user arriving to a recommender system. The system has no prior knowledge about the new arrival, yet has to make some recommendations. In this scenario, the hope is that deeper GNNs could recover features from more distant nodes, in order to create informative representations.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

Datasets: Aligned to most of the literature, we use four well-known benchmark datasets: *Cora, CiteSeer, Pubmed* and the less homophilic dataset *Texas*. The statistics of the datasets are reported in the Appendix. For *Cora, CiteSeer* and *Pubmed* we use the same data splits as in Kipf & Welling (2017), where all nodes except the ones used for training and validation are used for testing. For *Texas* we use the same splits as in (Pei et al., 2020).
Models: We utilize two different GNN models as our base models for the proposed methodology; namely GCN (Kipf & Welling, 2017) and GAT (Velickovic et al., 2018). We compare the results of these models with and without the modification of the slope of ReLU for varying number of layers. We do not compare against methods utilizing residual connections, because the aim of this work is to show that oversmoothing could be avoided without "short-circuiting" initial information to latter layers.
Hyperparameters: We set the number of hidden units (of each layer) of GCN and GAT to 128 for both models across all datasets. The L_2 regularization is set with penalty $5 \cdot 10^{-4}$ for both models. We vary the depth between 2 and 64 layers. The number of attention heads for GAT is set to 1.
Configuration: Each experiment is run 10 times and we report the average performance over these runs. We train all models for 200 epochs using Cross Entropy as a loss function. For our experiments we used an RTX 3080ti GPU.

4.2 EXPERIMENTAL RESULTS

Reducing oversmoothing:
Table 1 presents the classification performance of the two base models (GCN and GAT) on all four datasets, with and without the modified slope of ReLU (called Slope2GNN) for varying number of layers. Additionally, it presents results using SeLU, instead of ReLU, since it also modifies the slope of the function, while additionally normalizing node representations, as explained in section 2.4. To the best of our knowledge, this is the first time SeLU is used with GNNs.
Table 1 shows that methods with a modified activation function reduce oversmoothing significantly. As expected, baselines maintain a high performance for shallow architectures in homophilic datasets, where oversmoothing is not a problem. The modified activation functions (Slope2GNN and SeLU) consistently improve the testing accuracy as the number of layers ($\#L$) increases. Note that even the proposed method suffers performance degradation for very deep GNNs, due to the unavoidable nature of the oversmoothing (Oono & Suzuki, 2020). GCN and GAT average over information in the close neighborhood of the node, assuming homophily, which is not the case in the *Texas* dataset. Interestingly, low homophily in this dataset makes oversmoothing appear at a larger depth compared to other datasets. This is because connected nodes do not have similar representations and the model needs more Laplacian operations to oversmooth them.

The effect of deeper networks on different activation functions:
Focusing on the use of SeLU, although it seems to make the models resistant to oversmoothing, the effect seems to be lost for GCNs with 64 layers. This may be due to the sensitivity of the function on the choice of slope value, which is lower than 2. Moreover, the saturating region (part of SeLU used to reduce variance of data, if needed) of SeLU might further reduce the variance of node representations, acting in favor of oversmoothing. Finally, SNNs aim to keep the variance of the model stable, ignoring the effect of the Laplacian smoothing in GNNs.
To further examine the behavior of each method, as the number of layers increases, in Figure 1, we present a more detailed progression of the accuracy of GCNs, as the networks get deeper on the *Cora* dataset. The effect of oversmoothing and its reduction by the two methods that modify ReLU is very apparent here. Furthermore, the simpler Slope2GNN approach seems to be more resistant to the increase in the depth of the network.

The need for deep networks:
Having shown the benefit of using the modified activation functions in deep GNNs, we move to a set of experiments that aims to highlight also the value of using such deep architectures. In particular, we report our experimental results in the "cold start" scenario, as described in section 3.5. The cold-start datasets that we use in these experiments are generated by removing feature vectors from unlabeled nodes and replacing them with all-zero vectors. For each combination of GNN and

Accuracy (%)									
# Layers	Method	Cora		CiteSeer		Pubmed		Texas	
		GCN	GAT	GCN	GAT	GCN	GAT	GCN	GAT
2	Original	81.34	**77.64**	70.55	**68.84**	77.62	**77.67**	**56.22**	**58.92**
	Slope2GNN	**82.05**	76.81	**70.57**	68.42	**78.27**	76.62	51.78	55.24
	SeLU	81.59	76.08	70.28	67.32	77.73	76.54	54.16	56.00
4	Original	78.41	**78.11**	63.20	64.55	**77.78**	**76.43**	**47.46**	47.03
	Slope2GNN	**79.33**	77.74	**65.79**	**66.61**	77.47	75.76	46.05	**47.14**
	SeLU	78.77	78.00	65.26	65.79	77.11	76.27	46.03	47.14
8	Original	31.33	48.01	25.54	34.35	63.32	70.36	**52.32**	**48.86**
	Slope2GNN	**76.93**	75.17	**59.82**	**61.77**	**76.01**	**75.22**	47.35	45.62
	SeLU	75.71	**75.40**	58.60	61.26	75.73	75.19	46.59	47.13
16	Original	13.7	16.59	18.03	18.97	18.00	30.57	9.95	11.89
	Slope2GNN	**74.47**	**74.87**	**54.71**	55.31	**75.92**	73.49	47.03	**47.89**
	SeLU	74.34	73.78	54.43	**57.77**	75.09	**74.13**	**47.69**	47.24
32	Original	13.00	13.00	7.89	7.89	18.00	18.00	16.22	16.22
	Slope2GNN	**68.69**	**64.67**	44.08	53.09	**73.54**	**73.21**	46.92	47.78
	SeLU	67.66	46.21	**55.24**	**55.19**	69.45	51.65	**50.92**	**50.59**
64	Original	13.00	13.00	7.70	7.70	18.00	18.00	16.22	16.22
	Slope2GNN	**62.69**	**17.89**	**41.10**	**20.37**	**68.63**	31.87	42.92	34.81
	SeLU	17.46	13.97	22.59	17.32	36.01	**32.53**	**55.14**	**54.81**

Table 1: Performance comparison of vanilla GCN and GAT against SeLU and Slope2GNN enhanced versions of the same models, in *Cora, CiteSeer, Pubmed, Texas*. Average test node classification accuracy (%) for networks of different depth. With **bold** is the best performing model for each depth and each dataset.

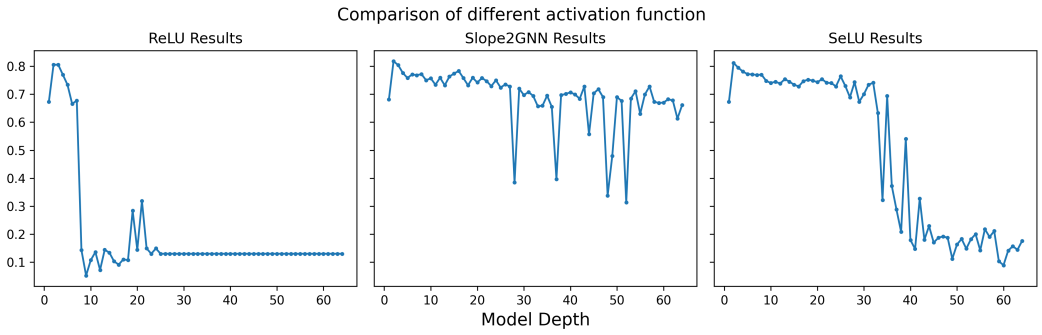

Figure 1: Comparison between three different activation functions for a GCN on *Cora* dataset. Y-axis shows model accuracy on test nodes, while varying the model's depth (shown in x-axis).

activation function, we present in Table 2 the best performance achieved and the depth at which the model attains that performance. The main observation in these results is the improvement in terms of accuracy with the use of deeper GNNs. This improvement is only attainable with the modified activation functions that reduce the effect of oversmoothing. The vanilla versions of GCN and GAT cannot go further than the performance of their shallow versions. Worth-mentioning is also the fact that the simple Slope2GNN approach manages to benefit from the deeper architectures even in the hard *Texas* dataset. This is not the case for SeLU, which may require tuning of the slope of the activation function, as discussed above.

5 Related Work

Li et al. (2018) were the first to introduce the problem of oversmoothing in GNNs. Subsequent work, (Oono & Suzuki, 2020; Cai & Wang, 2020) further analyzed and theoretically proved the existence

Model		GCN		GAT	
Dataset	Method	Accuracy (%)	#L	Accuracy (%)	#L
Cora	Original	64.56	4	62.20	3
	Slope2GNN	**73.90**	12	**72.23**	29
	SeLU	73.80	26	72.00	24
CiteSeer	Original	43.50	5	41.92	4
	Slope2GNN	**49.70**	14	**49.12**	31
	SeLU	47.52	24	48.10	27
Pubmed	Original	64.90	4	57.20	4
	Slope2GNN	**73.81**	27	72.51	25
	SeLU	72.40	29	**72.60**	30
Texas	Original	16.22	1	16.22	1
	Slope2GNN	**31.89**	21	**19.46**	7
	SeLU	16.22	1	16.22	1

Table 2: Comparison of different models and activation functions on the "cold start" problem. We show accuracy percentage (%) for the test set using GCN and GAT as backbone GNN models. Only the features of the nodes in the training set are available to the model. We also show at what depth (i.e. # Layers) each model achieves its best accuracy.

of the problem and showed that it is unavoidable as the depth increases. Xu et al. (2018) proposed Jumping Knowledge Networks (JK-Networks) as the first attempt to address the problem, by injecting skip connections to the GNNs. The model kept information from lower layers and combined them directly with the final layer, before labeling the node. Following a similar approach, Chen et al. (2020) proposed the use of residual connections between layers to enable deep architectures and alleviate the problem. Both of these approaches explicitly inject part of the initial information into higher layers of the network, implicitly downgrading the importance of intermediate layers. On the other hand, our method enables deep architectures without the need to inject initial information and allows the model to learn how to compress the initial information and what amount of it to maintain. We do not compare against these methods, because the aim of this work is to show that oversmoothing could be avoided without "short-circuiting" initial information to latter layers.

DropEdge (Rong et al., 2020) and DropNode (Do et al., 2021) are two alternative methods that address oversmoothing by altering the graph topology. In particular, they remove either edges or nodes at random, in order to slow down the message passing speed. In contrast to such approaches, our method does not change the graph topology, which may have unpredictable effects, but rather relies on a small adjustment of the slope of the activation function. An additional advantage of the approach proposed in this paper is that it does not have any hyper-parameters to be tuned, in contrast to all of the alternatives mentioned above. Hyperparameter tuning is also required by the Pairnorm method, which was mentioned in section 2.4.

6 Conclusion

In this paper we have shown the important role that the slope of the ReLU activation function plays in the oversmoothing problem in GNNs. We have further proposed a simple modification that reduces drastically the problem. We have illustrated the benefits of the approach in a set of experiments with different datasets and GNNs of different depth. Additionally, we showed the improvement in accuracy one can achieve through deep architectures that do not suffer from oversmoothing. This was evident in a set of experiments that simulated the "cold start" problem of missing node features in GNNs.

The simple link between the activation function and oversmoothing unveiled in this paper, opens up a range of interesting hypotheses to be investigated in the future. In particular, we would like to study the effect of changing the activation function in various GNN architectures and learning methods. Additionally, we would like to study alternative activation functions that could provide further benefits. Finally, we aim to study in more detail the approach of varying LR per layer.

REFERENCES

Lei Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *CoRR*, abs/1607.06450, 2016. URL http://arxiv.org/abs/1607.06450.

Yoshua Bengio, Patrice Y. Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Trans. Neural Networks*, 5(2):157–166, 1994. doi: 10.1109/72.279181. URL https://doi.org/10.1109/72.279181.

Chen Cai and Yusu Wang. A note on over-smoothing for graph neural networks. *CoRR*, abs/2006.13318, 2020. URL https://arxiv.org/abs/2006.13318.

Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pp. 1725–1735. PMLR, 2020. URL http://proceedings.mlr.press/v119/chen20v.html.

Tien Huu Do, Duc Minh Nguyen, Giannis Bekoulis, Adrian Munteanu, and Nikos Deligiannis. Graph convolutional neural networks with node transition probability-based message passing and dropnode regularization. *Expert Syst. Appl.*, 174:114711, 2021. doi: 10.1016/j.eswa.2021.114711. URL https://doi.org/10.1016/j.eswa.2021.114711.

William L. Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pp. 1024–1034, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/5dd9db5e033da9c6fb5ba83c7a7ebea9-Abstract.html.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Francis R. Bach and David M. Blei (eds.), *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pp. 448–456. JMLR.org, 2015. URL http://proceedings.mlr.press/v37/ioffe15.html.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=SJU4ayYgl.

Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. *CoRR*, abs/1706.02515, 2017. URL http://arxiv.org/abs/1706.02515.

Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. Personalized embedding propagation: Combining neural networks on graphs with personalized pagerank. *CoRR*, abs/1810.05997, 2018. URL http://arxiv.org/abs/1810.05997.

Johannes Klicpera, Janek Groß, and Stephan Günnemann. Directional message passing for molecular graphs. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=B1eWbxStPH.

Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In Sheila A. McIlraith and Kilian Q. Weinberger (eds.), *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pp. 3538–3545. AAAI Press, 2018. URL https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16098.

David Liben-Nowell and Jon M. Kleinberg. The link prediction problem for social networks. In *Proceedings of the 2003 ACM CIKM International Conference on Information and Knowledge Management, New Orleans, Louisiana, USA, November 2-8, 2003*, pp. 556–559. ACM, 2003. doi: 10.1145/956863.956972. URL https://doi.org/10.1145/956863.956972.

Lu Lu, Yeonjong Shin, Yanhui Su, and George E. Karniadakis. Dying relu and initialization: Theory and numerical examples. *CoRR*, abs/1903.06733, 2019. URL http://arxiv.org/abs/1903.06733.

Kenta Oono and Taiji Suzuki. Graph neural networks exponentially lose expressive power for node classification. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=S1ldO2EFPr.

Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. Geom-gcn: Geometric graph convolutional networks. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=S1e2agrFvS.

Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. Dropedge: Towards deep graph convolutional networks on node classification. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=Hkx1qkrKPr.

Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL https://openreview.net/forum?id=rJXMpikCZ.

Felix Wu, Amauri H. Souza Jr., Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Q. Weinberger. Simplifying graph convolutional networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pp. 6861–6871. PMLR, 2019. URL http://proceedings.mlr.press/v97/wu19e.html.

Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In Jennifer G. Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pp. 5449–5458. PMLR, 2018. URL http://proceedings.mlr.press/v80/xu18c.html.

Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pp. 5171–5181, 2018. URL https://proceedings.neurips.cc/paper/2018/hash/53f0d7c537d99b3824f0f99d62ea2428-Abstract.html.

Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. In Sheila A. McIlraith and Kilian Q. Weinberger (eds.), *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pp. 4438–4445. AAAI Press, 2018. URL https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17146.

Lingxiao Zhao and Leman Akoglu. Pairnorm: Tackling oversmoothing in gnns. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=rkecl1rtwB.

APPENDIX

A: LEMMA 1 PROOF

Lemma 1 *For a network of depth dep the total gradient reaching to the l-th layer* $\left(\text{i.e., } \frac{dJ}{dw^{(l)}_{old_{i,j}}}\right)$ *in order to update* $W^{(l)}$ *is bounded by:*

$$\frac{dJ}{dw^{(l)}_{old_{i,j}}} \leq \alpha^{(dep-l)} \cdot G_l$$

Where J is the model's loss function (i.e. Cross Entropy), α *stands for the ReLU slope and* G_l *is the upper bound of gradients of the output of the subsequent layer (l+1) with respect to the weight elements of* $W^{(l)}$*.*

Proof: Firstly, let us define the gradient of the output of a ReLU function with respect to the input as follows:

$$\frac{d(ReLU(x))}{dx} = \begin{cases} \alpha, & x > 0 \\ 0, & x \leq 0 \end{cases}$$

Let us consider now a function of nested ReLUs, like the neural networks under investigation, i.e. $f(x) = g(\sigma(g(\sigma(..g(\sigma(g(x)))..))))$, where $\sigma(\cdot)$ is the ReLU function applied repeatedly n times, and $g(\cdot)$ is a function that multiplies its input with a value. Then the gradient of $f(\cdot)$ with respect to x (x is a vector) is given as:

$$\frac{d(f(x))}{dx} = \begin{cases} \alpha^n \cdot \frac{d(g(x))}{dx}, & g(x) > 0, g(\sigma(\cdot)) > 0 \\ 0, & \text{otherwise} \end{cases} \tag{7}$$

This formula indicates the relation between slope, layer index, depth and largest singular value (we only care about the largest). Let as use the notion of J for the loss function of the network (i.e. Cross Entropy). The relation comes from the following formulas (for each weight element in the weight matrix):

$$w^{(l)}_{new_{i,j}} = w^{(l)}_{old_{i,j}} + \eta \cdot \frac{dJ}{dw^{(l)}_{old_{i,j}}}$$

$$\frac{dJ}{dw^{(l)}_{old_{i,j}}} = \alpha^{(depth-l)} \cdot \frac{dO^{(l+1)}}{w^{(l)}_{old_{i,j}}} \text{ or } 0$$

Where the 0 value comes from Equation 7. GNNs' forward pass is similar to $f(\cdot)$, hence performing the backpropagation leads to gradient calculation of the form of Equation 7. So the update rule adds (or subtracts) to each weight a big number (contains exponential factor). We prove, that $\frac{dJ}{dw_{old_{i,j}}}$ tends to be closer to zero as the layer index gets smaller, because when the gradient travels backwards the more distant it travels the more probable it is to die. There exist many components (gradients of ReLU functions) in the total gradient so it is probable one of them to be zero and the total gradient to be zero.
Given Assumption 1, we get an upper bound regarding the gradient of the output of the subsequent layer with respect to each weight element of $W^{(l)}$ $\left(\text{i.e., } \frac{dO^{(l+1)}}{w^{(l)}_{old_{i,j}}} \leq G_l\right)$.
So we will have that:

$$\eta \cdot \frac{dJ}{dw_{old_{i,j}}} \leq \eta \cdot \alpha^{(depth-l)} \cdot G_l \leq \alpha^{(depth-l)} \cdot B_l \tag{8}$$

where B_l is an upper bound on the product of the learning rate η with G_l, i.e. $B_l = \eta \cdot G_l$.

B: LEMMA 2 PROOF

Lemma 2 *While model's loss, through gradients, flows backwards some weight elements do not receive updates, because we have dying ReLUs (i.e. ReLUs outputting zero) (Lu et al., 2019). The total number of weight elements getting updated at layer l is bounded by:*

$$\#\{w_{i,j}^{(l)}\} \leq p^{(dep-l)} \cdot d^2$$

where d is the largest of the two dimensions of $W^{(l)}$, i.e. there are at most d^2 elements in $W^{(l)}$, if it is a square matrix.

Proof: Let us assume that the number of weights that get updated is reduced as the gradient flows backward. This is due to the fact that, the further the gradient 'travels' the more gradients of ReLUs will exist within it. Hence the greater is the probability at least one of them to be zero. In fact, we define the probability of a weight element to get updated and the number of weight elements to be updated in layer l as:

$$P\{w_{i,j}^{(l)} = updated\} = p^{(depth-l)}$$

$$\#\{w_{i,j}^{(l)}\} = p^{(depth-l)} \cdot (i \cdot j) \leq p^{(depth-l)} \cdot d^2 \tag{9}$$

Where i, j are the dimensions of the weight matrix and d is the largest of them.
The probability for a weight element to get updated has the aforementioned form, because we have used the Assumption 2 regarding the probability of a ReLU unit not to output zero, which is at most p. Thus, the probability of getting updated is the probability of the gradient flowing backwards to reach the weight element, which in turn means not to have any ReLU component equal to zero.
The total number of elements that get updated is upper bounded (using union bound) on the sum of the probabilities of all elements of the weight matrix. In fact, every weight matrix has $i \cdot j$ elements where i, j are the matrix dimensions and $d = max(i, j)$.

C: THEOREM 2 PROOF

Theorem 2 *The upper bound of the largest singular value of the weight matrix $W^{(l)}$ at layer l for a GNN model, utilizing a ReLU activation function, depends on the slope of the function. That bound is given per layer and shows the effect of each iteration of updates on the weight matrix. We denote with W_{old} and W_{new} the weight matrices before and after the update during an iteration of the training process respectively.*

$$max(s_l(W_{new}^{(l)})) \leq ||W_{old}^{(l)}||_F + \sqrt{3} \cdot p^{(\frac{dep-l}{2})} \cdot d \cdot \alpha^{(dep-l)} \cdot B_l$$

where $B_l = \eta \cdot G_l$, dep is network's depth, d is the largest dimension of $W^{(l)}$ matrix, p is the upper bound of the probability of ReLU not to output zero and α is ReLU's slope.

Proof: Regarding the singular values of a matrix (denoted by s_l), it is known that:

$$max(s_l(W^{(l)})) = ||W^{(l)}||_2 \leq ||W^{(l)}||_F = \sqrt{\sum |w_{i,j}^{(l)}|^2}$$

the matrix here being $W^{(l)}$, the weight matrix at layer l. Based on Lemma 1, a new weight, during the weight update process will be given by: $w_{new} \sim \alpha^{(depth-l)} \cdot B_l$. Since only $\#\{w_{i,j}^{(l)}\}$ weight elements are getting updated per layer (Lemma 2), in each iteration the Frobenius norm increases by a value, that depends on layer index, depth and slope. Specifically, if we define $\alpha^{(depth-l)} \cdot B_l = K_l$, in order to simplify the formulas, we have:

$$max(s_l(W_{new}^{(l)})) \leq ||W_{new}^{(l)}||_F = \sqrt{\sum |w_{new_{i,j}}^{(l)}|^2} \rightarrow$$

$$max(s_l(W^{(l)}_{new})) \leq \sqrt{\sum_{updated} |w^{(l)}_{new_{i,j}}|^2 + \sum_{not_updated} |w^{(l)}_{new_{i,j}}|^2} \xrightarrow{8}$$

$$max(s_l(W^{(l)}_{new})) \leq \sqrt{\sum_{updated} |w^{(l)}_{old_{i,j}} + K_l|^2 + \sum_{not_updated} |w^{(l)}_{old_{i,j}}|^2} \rightarrow$$

$$max(s_l(W^{(l)}_{new})) \leq \sqrt{\sum_{updated} \left(|w^{(l)}_{old_{i,j}}|^2 + |K_l|^2 + 2w^{(l)}_{old_{i,j}} K_l\right) + \sum_{not_updated} |w^{(l)}_{old_{i,j}}|^2}$$

Using the fact that $w^{(l)}_{old_{i,j}} <= K_l$, because K_l contains an exponential term and weights are initialized to values close to zero we get:

$$max(s_l(W^{(l)}_{new})) \leq \sqrt{\sum_{updated} \left(|w^{(l)}_{old_{i,j}}|^2 + 3|K_l|^2\right) + \sum_{not_updated} |w^{(l)}_{old_{i,j}}|^2} \xrightarrow{\sqrt{A+B} \leq \sqrt{A} + \sqrt{B}}$$

$$max(s_l(W^{(l)}_{new})) \leq \sqrt{\sum_{updated} |w^{(l)}_{old_{i,j}}|^2 + \sum_{not_updated} |w^{(l)}_{old_{i,j}}|^2} + \sqrt{\sum_{updated} 3|K_l|^2} \xrightarrow{Lemma2(9)}$$

In the second square root we sum over the updated weight elements and K_l is independent of them. Hence, we get:

$$max(s_l(W^{(l)}_{new})) \leq ||W^{(l)}_{old}||_F + \sqrt{3|K_l|^2 \cdot \#\{w^{(l)}_{i,j}\}}$$

$$max(s_l(W^{(l)}_{new})) \leq ||W^{(l)}_{old}||_F + \sqrt{3} \cdot |K_l| \cdot p^{\left(\frac{depth-l}{2}\right)} \cdot d$$

$$max(s_l(W^{(l)}_{new})) \leq ||W^{(l)}_{old}||_F + \sqrt{3} \cdot p^{\left(\frac{depth-l}{2}\right)} \cdot d \cdot \alpha^{(depth-l)} \cdot B_l$$

So the slope of the function determines the upper bound of the Frobenius norm while training. In turn this norm is directly connected as an upper bound to the largest singular value of the matrix. In the aforementioned proof we have used previously defined Lemmas and Assumptions.

D: MODIFYING LEARNING RATE: LR2GNN

As discussed in section 3.4, an alternative to changing the slope of ReLU is to modify the learning rate. In order to test this hypothesis and compare the results to the modified activation function, we performed some preliminary experiments with GCN. In these experiments, we determined a different learning rate per layer, using the validation part of each dataset and the chosen values are shown in Table 3. We did this only for the first 8 layers, as the process of tuning the rate proved very cumbersome and time-consuming. The main observation seems to be the improved performance over the vanilla GCN for the three simpler datasets, coupled with a low accuracy for the *Texas* dataset. The comparison against the modified activation function seems inconclusive, but the latter approach wins due to its simplicity. If one opted for the modified learning rate, a method to automatically adapt the rate in different layers would be needed. Deeper architectures extend the complexity and the pool, from which we would have to find the proper values of learning rates. Worth to mention, combining different values of LR per layer with Slope2GNN results in negligible improvements.

Layer index	0	1	2	3	4	5	6	7
Learning Rate	$3 \cdot 10^{-3}$	10^{-4}	10^{-5}	$5 \cdot 10^{-5}$	10^{-5}	10^{-4}	10^{-4}	10^{-4}

Cora Accuracy: **76.43**	CiteSeer Accuracy: **64.96**	Pubmed Accuracy: **78.82**
Texas Accuracy: **16.32**		

Table 3: Preliminary results of modifying the learning rate of GCN on *Cora, CiteSeer, Pubmed* and *Texas* datasets. The graph displays test accuracy of an 8-layer GCN, using the aforementioned learning rates.

E: Datasets Statistics

Datasets	# Nodes	# Edges	# Classes	# Features
Cora	2708	5429	7	1433
CiteSeer	3327	4732	6	3703
Pubmed	19717	44338	3	500
Texas	183	309	5	1703

Table 4: The statistics of all datasets used in this work.