Journal of Machine Learning Research (2023) Submitted ; Published

A brief study of over-smoothness in deep Graph Convolutional Networks

Abstract

Graph representation learning has emerged as a powerful technique for capturing the structure and properties of complex networks. In recent years, Graph Convolutional Networks (GCN) and Graph Attention Networks (GAT) have gained significant attention due to their ability to learn meaningful node embeddings. However, deep GCNs often suffer from over-smoothing problem. In our report, we investigate into the problem in GAT and GCN models by incorporating different advanced techniques, such as Graph Convolutional Network with Initial Residual and Identity Mapping (GCNII), DropEdge, and Differentiable Group Normalization (DGN). Our experimentation results demostrate that the integrations between DGN and various GNN architectures successfully ameliorate the performance of the models.

Keywords: Over-smoothing, GCN, GAT, DropEdges, Group Normalisation

1. Introduction

Training a deep Graph Neural Network (GNN) has been recognised as a notoriously challenging task in the field of graph representation learning (Zhou et al., 2018). Besides the standard plights in training deep architectures such as vanishing gradient, over-fitting and information squashing problem, the problem of over-smoothing is also restricting the ability of the GNN in node classification tasks. Over-smoothing happens when the node features of different classes become indistinguishable as the results of several recursive neighbourhood aggregation (Li et al., 2018). This behavior prevents deep GNNs from effectively modeling the higher-order dependencies from multi-hop neighbors, and substantially decreases the discriminative power of node embeddings which ends up with a degraded performance in graph learning task like node classification.

While numerous theories and miscellaneous methods have been developed to understand and address the problem, it is far to say sufficient and still lacks integral analysis. In our project, we primarily focus on investigating the over-smoothing issue in two GNN arhictectures, GCN (Kipf and Welling, 2016) and GAT (Veličković et al., 2018). Among the previously proposed approaches that mitigates over-smoothness, we pick up several representative tricks (we will discuss further in section 2) to address the over-smoothing problem: GCNII (Chen et al., 2020b), DropEdges(Rong et al., 2019), PairNorm(Zhao and Akoglu, 2019) and DGN(Zhou et al., 2020).

We reproduce the main tricks in these works and put them under a fair competition. Our contributions include testing the compatibility and effectiveness of different combinations of these approaches and providing a comprehensive comparison of them.

2. Related Works

Graph Neural Network is widely adopted in graph representation learning, which can be divided into two main categories: the spectral domain based methods and the spatial domain based methods. The spectral domain-based methods leverage the graph's spectral properties to learn node representations. A good example is GCN (Kipf and Welling, 2016), a localized first-order approximation of spectral graph convolutions. By stacking multiple layers of graph convolution, GCN can learn high-level representations of nodes in the graph. Wu et al. (2019) retrospectively designs SGC, which simply adopts linear transformation on the feature matrix. Other notable examples include the ChebNet (Defferrard et al., 2016) and Spectral Networks (Bruna et al., 2014). On the other hand, the spatial domain-based methods focus on aggregating information from the nodes' local neighborhood in the graph. A good example will be GAT (Veličković et al., 2018), which employs self-attention mechanisms to capture more meaningful contextual information. Other popular examples include GraphSAGE (Hamilton et al., 2017).

The over-smoothing problem usually arises when we naively increase the number of graph convolutional layers. A series of theoretical analysis has been conducted on the issue. In the scope of spectral analysis, the Laplacian of GCN is regarded as low-pass filter Nt and Maehara (2019), and it can be proved that deep models inevitably encounter over-smoothing for the symmetric Laplacian will converge to a stationary matrix and nodes with higher degrees suffer more from over-smoothing. Oono and Suzuki (2019) agrees that GNN lose expressive power exponentially for node classification under certain conditions. Xu et al. (2018); Zhao and Akoglu (2019); Liu et al. (2020); Chen et al. (2020a) also give theoretical analysis on over-smoothing problem concerning the topology of the graph and propose JKNet, and PairNorm, MADreg, and DAGNN respectively as solutions.

In addition to what we have mentioned, there has been a large collections of researches developed to tackle over-smoothing. And we categorize them as following:

1. **Skip connections**: Inspired by the success of ResNet (He et al., 2015) and DenseNet (Huang et al., 2017) in deep learning, the similar techniques have been deployed in GCNs to create shortcuts between layers, allowing the model to learn both local and global features. Notable examples include JKNet and ResGCN (Li et al., 2019). Chen et al. (2020b) produces GCNII, which amends the idea of APPNP (Gasteiger et al., 2018) by combining initial residual connection with identity mapping, successfully building up to 64 layers of Laplacian.

2. **Normalisation**: Classical tricks like BatchNorm can be applied in GNN (Ioffe and Szegedy, 2015; Dwivedi et al., 2020). In the context of GNN, PairNorm centers and rescales node embedding at the graph level; Zhou et al. (2020) proposes a novel approach Diiferentiable Group Normalisation (DGN) by softly assigning nodes into several groups and adjust embedding within groups. Other methods include NodeNorm (Zhou et al., 2021), MeanNorm (Yang et al., 2020), and a united graph normalisation (Chen et al., 2022). By controlling the propagation of information, these normalisation techniques enhance the model's ability to capture local and global structures effectively.

3. **Random Dropping**: Except for the traditional Dropout (Srivastava et al., 2014), DropEdge (Rong et al., 2019; Huang et al., 2020) and DropNode (Do et al., 2021) are introduced to randomly sample a proportion of edges or nodes from the input graph at each training epoch or each layer of Laplacian. They are essentially data augmentation methods that is complementary to other tricks, helping relieve both the over-fitting and over-smoothing issues in training very deep GNNs.

3. Problem Setting

3.1 Notations

Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of nodes and $\mathcal{E}$ is the set of edges, let $\mathbf{A}$ be the adjacency matrix of $\mathcal{G}$, and $\mathbf{X} \in \mathbb{R}^{|V| \times F}$ be the node feature matrix, where F is the number of features per node.

3.2 Preliminaries

3.2.1 Metrics

To the best of our knowledge, there has not been an universally accepted standard metric for smoothness. However, many researchers have come up with various heuristic yet innovative metrics for smoothness and over-smoothness. For example, the aforementioned MADreg is based on Mean Average Distance Gap (MADGap), which calculates the difference of the mean average distance among node representations of different clusters in the graph (Chen et al., 2020a), and PairNorm and JKNet also cement their theoretical basis on similar distance measures, namely row-diff, col-diff, and influence score in corresponding literature (Zhao and Akoglu, 2019; Xu et al., 2018). Another more rigorously defined metric of over-smoothness is ϵ-smoothing which seeks the infimum of number of layers that suffers from the smoothness at lease to some degree (Oono and Suzuki, 2019). In this project, however, we adopt Group Distance Ratio (R_{group}) and Instance Information Gain (G_{ins}) that are defined in Zhou et al. (2020) as two metrics of over-smoothness because they are from the work we reproduce, and more importantly, they sketch the phenomenon in two different perspectives.

1. **Group Distance Ratio**: Assume there are in total C classes of node labels. Let $E_i = \{\mathbf{h}_v^i\}$ be the set of representation of node v in class i, then we have a series of group representations $\{E_1, \cdots, E_C\}$. The Group Distance Ratio R_{group} is then defined by:

$$R_{group} = \frac{\frac{1}{(C-1)^2} \sum_{i \neq j} \left[\frac{1}{|E_i||E_j|} \sum_{\mathbf{h}_v^i \in E_i} \sum_{\mathbf{h}_{v'}^j \in E_j} \|\mathbf{h}_v^i - \mathbf{h}_{v'}^j\|_2 \right]}{\frac{1}{C} \sum_i \left[\frac{1}{|E_i|^2} \sum_{\mathbf{h}_v^i, \mathbf{h}_{v'}^i \in E_i} \|\mathbf{h}_v^i - \mathbf{h}_{v'}^i\|_2 \right]} \tag{1}$$

   The numerator (denominator) represents the average of pairwise distances between two different groups (within the same group), in other word, inter-group distance over intra-group distance. It is preferable that inter-group distance is larger and intra-group smaller, and a small R_{group} may indicate over-smoothness.

2. **Instance Information Gain**: Utilizing the tools in information theory, we treat each node instance independently and define Instance Information Gain (G_{ins}) to measure how much input feature information is contained in the final representation. Let $\mathcal{X}, \mathcal{H}$ be the random variables of input feature and final representation vectors respectively, and $\mathcal{P}_{\mathcal{X}}, \mathcal{P}_{\mathcal{H}}, \mathcal{P}_{\mathcal{XH}}$ be corresponding marginal distribution and joint distribution. G_{ins} then takes form of mutual information:

$$G_{ins} = \sum_{\mathbf{x}_v \in \mathcal{X}, \mathbf{h}_v \in \mathcal{H}} \mathcal{P}_{\mathcal{XH}}(\mathbf{x}_v, \mathbf{h}_v) \log \frac{\mathcal{P}_{\mathcal{XH}}(\mathbf{x}_v, \mathbf{h}_v)}{\mathcal{P}_{\mathcal{X}}(\mathbf{x}_v)\mathcal{P}_{\mathcal{H}}(\mathbf{h}_v)} \tag{2}$$

3.2.2 Backbone Models

1. **Graph Convolutional Network (GCN)**: The key idea behind GCNs is to learn representations of nodes by aggregating information from their neighbors in the graph. The graph convolution operation in a single GCN layer can be defined as:

$$\mathbf{H}^{(l+1)} = \sigma(\tilde{\mathbf{P}}\mathbf{H}^{(l)}\mathbf{W}^{(l)}) \tag{3}$$

where $\mathbf{H}^{(l)}$ is the node representation matrix at the l-th layer, $\tilde{\mathbf{P}} = \tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}$, and $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ is the adjacency matrix with self-connections, $\tilde{\mathbf{D}}$ is the diagonal degree matrix of $\tilde{\mathbf{A}}$, $\mathbf{W}^{(l)}$ is the learnable weight matrix at the l-th layer, and $\sigma(\cdot)$ is an activation function, such as the ReLU function. $\mathbf{H}^{(l)} \in \mathbb{R}^{n \times d_l}$; $\mathbf{H}^{(0)}$ is the input node featrue, and n is the number of nodes, d_l the dimension of l-th layer embedding.

2. **Graph Attention Network (GAT)**: GAT leverages attention mechanism to learn node representations by aggregating features from their neighbors in the graph, weighted by the attention coefficients. The attention mechanism in a single GAT layer can be defined as:

$$\alpha_{uv}^{(l)} = \frac{\exp(\delta(\mathbf{a}_{(l)}^{\top}\mathbf{W}^{(l)}(\mathbf{h}_u \oplus \mathbf{h}_v)))}{\sum_{v' \in \mathcal{N}(u)} \exp\left(\delta(\mathbf{a}_{(l)}^{\top}\mathbf{W}^{(l)}(\mathbf{h}_u \oplus \mathbf{h}_{v'}))\right)} \tag{4}$$

where the $\mathbf{h}_u$ is the feature vector of node u, $\mathbf{W}^{(l)}$ and $\mathbf{a}_{(l)}$ are the learnable weight matrix and the learnable attention vector at the l-th layer respectively. $\delta(\cdot)$ is the LeakyReLU activation function and we denote $\mathcal{N}(u)$ as the set of neighbors of node u. The attention coefficients α_{uv} determine the importance of node v's features to node u. $\oplus$ represents concatenation operation.

The output of a single GAT layer can be expressed as:

$$\mathbf{h}_u^{(l+1)} = \sigma\left(\sum_{v \in \mathcal{N}(u)} \alpha_{uv}^{(l)}\mathbf{W}^{(l)}\mathbf{h}_u^{(l)}\right) \tag{5}$$

where $\mathcal{N}(u)$ is the set of neighbours of node u

3.3 Techniques Alleviating Over-Smoothing

1. **GCNII**: GCNII circumvents the expression power decay (Oono and Suzuki, 2019) by assembling the initial residual connection component as in APPNP (Gasteiger

et al., 2018) and further reduces the performace degradation due to over-fitting by the identity mapping.

$$\mathbf{H}^{(l+1)} = \sigma\left(\left((1-\alpha_l)\tilde{\mathbf{P}}\mathbf{H}^{(l)} + \alpha_l\mathbf{H}^{(0)}\right)\left((1-\beta_l)\mathbf{I}_n + \beta_l\mathbf{W}^{(l)}\right)\right) \tag{6}$$

Here, α_l, β_l are tunable parameters. Note, by counting the most "unsmooth" $\mathbf{H}^{(0)}$ in, even when the number of layers goes to infinity, the output representation still involves the original feature information. Each Laplacian smoothing, the trick requires several extra matrix addition operation, hence the extra time complexity comparing to vanilla GCN is just $\mathcal{O}(n^2)$ at each layer.

2. **DropEdge**: DropEdge slows down the convergence rate of over-smoothing by partly blocking message passage each training (or between layers). It can be formulated as:

$$\mathbf{H}^{(l+1)} = \sigma\left(\mathfrak{N}\left(\mathbf{A}\odot\mathbf{Z}^{(l)}\right)\mathbf{H}^{(l)}\mathbf{W}^{(l)}\right) \tag{7}$$

where $\mathfrak{N}$ is the selected normalisation operator, for instance we can choose the form $\mathfrak{N}(\mathbf{A}) = (\mathbf{D}+\mathbf{I}_n)^{-\frac{1}{2}}(\mathbf{A}+\mathbf{I}_n)(\mathbf{D}+\mathbf{I}_n)^{-\frac{1}{2}}$. $\mathbf{Z}$ is the random binary mask and $\odot$ the element-wise multiplication. In this project, we only employ DropEdge at the begining each training, i.e. we set $\mathbf{Z}^{(l)} = \mathbf{Z}^{(0)}$ for $l > 0$, then in practice the extra time complexity of this one-shot operation is actually $\mathcal{O}(n^2)$.

3. **PairNorm**: PairNorm can be split into two step, centering and rescaling:

$$\mathbf{h}_u^{(l)} \leftarrow \mathbf{h}_u^{(l)} - \bar{\mathbf{h}}^{(l)}, \quad \bar{\mathbf{h}}^{(l)} = \frac{1}{n}\sum_{u\in[n]}\mathbf{h}_u^{(l)} \tag{8}$$

$$\mathbf{h}_u^{(l)} \leftarrow \frac{s\mathbf{h}_u^{(l)}}{\sqrt{\frac{1}{n}\sum_{v\in[n]}\|\mathbf{h}_v^{(l)}\|_2^2}} \tag{9}$$

Here $\mathbf{h}_u^{(l)} \in \mathbb{R}^{d_l}$ is the i-th node embedding of the l-th layer, and s is a tunable scaler. PairNorm looks ordinary, yet it produces an embedding matrix that keeps total pairwise squared distance a constant:

$$\sum_{i,j\in[n]}\|\tilde{\mathbf{h}}_u^{(l)} - \tilde{\mathbf{h}}_v^{(l)}\|_2^2 \equiv 2n^2s^2 \tag{10}$$

The centering step takes $\mathcal{O}(n^2d_l)$ and rescaling takes $\mathcal{O}(nd_l)$, so summing up each PairNorm layer consumes $\mathcal{O}(n^2d_l + nd_l)$.

4. **DGN**: Differentiable Group Normalisation softly assigns nodes into G groups (G is a tunable hyperparameter) by a cluster assignment matrix $\mathbf{S}^{(l)} \in \mathbb{R}^{n\times G}$. First suppose $\mathbf{S}^{(l)}$ is known, then we computed the features $\mathbf{H}_i^{(l)} \in \mathbb{R}^{n\times d_l}, i \in [G]$ of each cluster by:

$$\mathbf{H}_i^{(l)} = \mathbf{S}^{(l)}[\,:,\ i\,] \circ \mathbf{H}^{(l)} \tag{11}$$

where $\mathbf{S}^{(l)}[\ :\ ,\ i\ ]$ is the i-th colomn of the matrix, and $\circ$ the row-wise multiplication (done by broadcast mechanism). And we update $\mathbf{H}^{(l)}$ by

$$\mathbf{H}^{(l)} \leftarrow \mathbf{H}^{(l)} + \lambda \sum_{i \in [G]} \left(\gamma_i \frac{\mathbf{H}_i^{(l)} - \mu_i}{\sigma_i} + \beta_i \right) \tag{12}$$

here, μ_i, σ_i denotes the running mean and standard deviation of group i, γ_i, β_i are trainable scale and bias vectors, λ is a balancing coefficient. In short, the l-th layer representation is modified by a sum-pooling group features. And then we reveal how $\mathbf{S}^{(l)}$ is calulated:

$$\mathbf{S}^{(l)} = \text{softmax}(\mathbf{H}^{(l)}\mathbf{U}^{(l)}) \tag{13}$$

$\mathbf{U}^{(l)} \in \mathbb{R}^{d_l \times G}$ is the trainable weights for one DGN module. Softmax is applied row-wisely to generate the normalisd probability with regard to all the G groups for each node. This step is literally why the trick is named "differentiable". To avoid redundant derivation of time complexity, we shortcut the time complexity of embedding normalisation at each group (getting one block of $\gamma_i(\mathbf{H}_i^{(l)} - \mu_i)/\sigma_i + \beta_i$ is $\mathcal{O}(T)$, which is dependent on n and d_l then for G groups it will be $\mathcal{O}(GT)$ in total. Other extra operations are of $\mathcal{O}(nd_lG)$, so the total time complexity for one DGN module is $\mathcal{O}(nd_lG + GT)$ (ignoring the extra auto-gradient cost).

We aim to explore how these techniques and their combinations may alleviate over-smooth on the two backbones, GCN and GAT in the following section (Note GCNII cannot be treated as a plug-in module like the others; it works more like a new backbone). In paticular, we conduct 3 types of normalisation methods (BatchNorm, PairNorm, DGN) and vanilla backbone either with or without DropEdge. Additionally, we deploy solely GCNII and GCNII with 3 types of normalisations as another resort. After training the models, we will apply the model on the test dataset, document the corresponding accuracy, as well as compute the metrics of over-smoothness.

4. Experiment

4.1 Datasets

In our project, we conduct experiments on two widely used citation networks, Cora and CiteSeer. We employ the Torch Geometric library, specifically the Planetoid collection, to load the CORA and CiteSeer datasets. These datasets contain sparse bag-of-words feature vectors for each document and a list of citation links between documents, as described in Yang et al. (2016). In these networks, nodes represent documents and edges represent citation links. We adhere to the default settings provided by the Torch Geometric Planetoid package for the train-test split. For the Cora dataset, there are 2,708 nodes, 10,556 edges, 1,433 features, and 7 classes. In the case of the CiteSeer dataset, it consists of 3,327 nodes, 9,104 edges, 3,703 features, and 6 classes.

4.2 Parameter Setting and Details of Deployments

We employed the GCN, GAT and GCNII model with a set of hyperparameters optimized for the specific datasets Cora and Citeseer.

All models were run for 1000 epochs to ensure convergence, and tested configurations of 2, 15, and 30 layers for each model. Each configuration was run for 5 times with different random seeds and the average test accuracy of Cora are summarised in table 1 with corresponding comparable numbers in papers (results of Citeseer is attached in the appendix A).

We choose the Adam optimizer with the learning rate 0.005, set the weight decay as 5e-4 for Cora and 5e-5 for Citeseer. The hidden dimensions d_l were set to 16 for all ls.

For the DropEdge, the random dropping rate was set to 0.05.

For the DGN, the group number G was set to 10. The balancing coefficient λ, was determined based on the number of layers in the network. For networks with 2 layers, λ was set to 0.001. However, for networks with 15 or 30 layers, λ was set to 0.03 for GCN and 0.01 for GAT.

For the GCNII architecture, α_l is set to 0.1 and β_l is set to 0.5 for all ls, and the hidden dimension size is changed to 64. A learning rate of 0.01 is used for training the model. And when group normalization is employed with GCNII, the balancing parameter λ is modified into smaller magnititude: for 2 layers, λ is set to 0.005, and for 15 or 30 layers, λ is set to 0.001.

4.3 Main Results and Discussion

As shown in table 1, astonishingly, not only does DropEdge not slow down the smoothing, but it even worsens the performance in almost all the experimentation (and the results tremendously differ from the numbers in original papers). We rechecked our implementations against the official codes, and one major difference is that we apply randomly dropping on $\tilde{\mathbf{A}}$ rather than the original adjacency matrix $\mathbf{A}$. In addition, we ignore the normalisation after dropping. Such naive version of DropEdge can be formulated as:

$$\mathbf{H}^{(l+1)} = \sigma\left((\tilde{\mathbf{A}} \odot \mathbf{Z})\mathbf{H}^{(l)}\mathbf{W}^{(l)}\right) \tag{14}$$

Statistically, the lack of normalisation may lead to biased data augmentation which can further dampen the subsequent message propagation. And with the application of our naive version of DropEdge, we empirically provide favourable evidence to the suspicion. The over-smoothing metrics are also in-line with the results that we get on the test accuracy, as summarised in table 2 (G_{ins} and other results are attached in appendix B).

Then, we focus on 3 types of normalisation. Clearly, DGN is the best out of them with regard to the application in GCN and GAT. Group Normalisation alone improves the accuracy and mitigates over-smoothing by increasing R_{group} and G_{ins}, especially when the number of layers gets larger (see Layer = 15, 30). In contrast, PairNorm and BatchNorm enhance the performance only when the number of layers are large. Specifically, PairNorm seems to be more compatible with GAT while BatchNorm suits GCN more if we juxtapose these two tricks.

Finally, we inspect on the effect of GCNII design. It is not hard-edged at first when we stack only 2 layers, but as the number of layers increases, GCNII exhibits sensational

Table 1: Best Test Accuracy (%) on the Cora dataset

Method	Layers = 2			Layers = 15			Layers = 30		
	GCN	GAT	GCNII	GCN	GAT	GCNII	GCN	GAT	GCNII
NN	79.96 (86.1)	78.76 (80.9)	81.56 (82.2)	18.42 (18.1)	18.00 (16.8)	84.12 (84.6)	13.00 (13.1)	13.10 (13.0)	82.28 (85.4)
NN + DE	47.26 (85.5)	73.42		17.12 (75.7)	18.00		13.14 (62.5)	13.10	
BN	72.50 (73.9)	77.14 (77.8)	77.02	69.76 (70.3)	28.80 (33.1)	67.98	70.92 (67.2)	25.58 (25.0)	68.82
BN + DE	37.36	68.60		25.98	26.46		21.72	20.98	
PN	70.66 (71.0)	74.16 (74.4)	72.24	66.48 (67.2)	65.02 (49.6)	56.26	62.42 (64.3)	34.96 (30.2)	58.10
PN + DE	36.94	68.82		20.18	23.84		28.12	26.76	
DGN	80.10 (82.0)	80.00 (81.1)	79.80	73.90 (75.2)	66.02 (71.8)	82.56	73.86 (73.2)	57.44 (51.3)	81.72
DGN + DE	47.12	73.12		29.88	32.20		26.00	30.88	

[1] NN stands for No Normalisation, BN for BatchNorm, PN for PairNorm, DGN for Differentiable Group Normalisation, and DE for DropEdge(naive).
[2] All the unparenthesized numbers are the average of 5 random seeds.
[3] the numbers in parentheses are comparable results in original papers (Rong et al., 2019; Zhou et al., 2020; Chen et al., 2020b). If there were no corresponding implementation, then the location is left blankly

performance. This is aligned with the theoretical prediction that even when the number of layers goes to infinity, GCNII can still skirt the plague of over-smoothing. Moreover, adding BatchNorm or PairNorm only hinders GCNII's pursuance. Only the accompany of DGN has a minor chance to further distinguish different groups as indicated by R_{group} , yet eventually impairs the test accuracy more or less.

For more details like the test accuracy curves along the training, please refer to appendix D

5. Conclusion

In summary, our project delves into the over-smoothing issue in graph representation learning, which affects the performance of GCN and GAT. By conducting a series of experiments, we discovered that GCNII and the combination of DGN to GCNII yielded the optimal results in comparison to other tested configurations.

Interestingly, our findings reveal that DropEdge should be paired with normalisation when being used as a technique to mitigate over-smoothing. While DropEdge has shown great potential in previous literature, our fair competition results demonstrate that GCNII

Table 2: Group Distance Ratio on the Cora dataset

Method	Layers = 2			Layers = 15			Layers = 30		
	GCN	GAT	GCNII	GCN	GAT	GCNII	GCN	GAT	GCNII
NN	3.55021	3.80466	3.46125	1.27579	1.11024	3.77802	1.14437	1.34700	4.26436
NN + DE	1.39346	2.77381		1.00901	1.11316		1.00633	1.34701	
BN	2.63955	3.07512	2.87636	2.47480	1.54464	2.25546	2.38970	1.06963	2.39061
BN + DE	1.22541	2.20472		1.09254	0.73168		1.00239	1.04676	
PN	2.61367	3.00433	2.76699	2.14872	2.40118	2.92002	1.89231	2.27899	2.48638
PN + DE	1.21404	2.39282		1.04964	1.16397		1.07220	1.21725	
DGN	3.52845	3.84101	3.34332	2.74739	3.09741	4.20578	2.80990	3.15689	4.41184
DGN + DE	1.38838	2.69239		1.14232	1.21972		1.03601	1.06173	

[1] NN stands for No Normalisation, BN for BatchNorm, PN for PairNorm, DGN for Differentiable Group Normalisation, and DE for DropEdge(naive). [2] All the numbers are the average of 5 random seeds.

desgin and DGN are more effective in preserving node-specific information and enhancing the discriminative power of the learned embedding. DGN, in particular, dominates the category among other normalisation tricks, namely BatchNorm and PairNorm. Even though we have implemented various settings of trick combinations, we cover only some representative ones in the literature. Besides, we only set numbers of stacked model layers to 2, 15, 30 due to the restriction of computational resource.

Based on our project, we propose several remaining issues: 1) the reason for the incompatibility of GCNII the normalisation tricks like PairNorm and BatchNorm. 2) the compatibility of authentic DropEdge and layer-wise DropEdge with GCNII. 3) the upgradability of the group assignment matrix in DGN. Moreover, theoretical formalisation of the over-smoothing problem is yet to be solved and we are in dearth of a unified framework to explain the rationale behind these approaches and interventions. We hope future researches will address these problems.

Appendix A. Test Accuracy in Citeseer

Table 3: Best Test Accuracy (%) on the CiteSeer dataset

Method	Layers = 2			Layers = 15			Layers = 30		
	GCN	GAT	GCNII	GCN	GAT	GCNII	GCN	GAT	GCNII
NN	66.98 (70.6)	67.72 (70.2)	64.68 (68.2)	22.3 (15.2)	18.46 (22.6)	70.32 (72.9)	8.6 (9.4)	7.5 (7.7)	71.58 (73.4)
NN + DE	47.18 (78.7)	61.62		19.46 (77.2)	18.58		11.88 (61.4)	0748	
BN	49.04 (51.3)	61.66 (61.5)	49.92	46.38 (46.9)	22.44 (28.0)	48.28	43.02 (47.9)	20.94 (21.4)	54.7
BN + DE	35.18	49.9		20.02	21.18		18.38	19.34	
PN	57.06 (60.5)	60.48 (62.0)	49.94	42.0 (46.7)	37.78 (41.4)	46.58	43.06 (47.1)	29.54 (33.3)	37.38
PN + DE	36.56	54.84		21.82	18.16		21.26	22.24	
DGN	66.16 (69.5)	67.08 (69.3)	67.26	50.06 (53.1)	47.36 (52.6)	69.16	49.48 (52.6)	21.3 (45.6)	70.88
DGN + DE	43.72	61.12		29.72	22.32		21.3	23.8	

[1] NN stands for No Normalisation, BN for BatchNorm, PN for PairNorm, DGN for Differentiable Group Normalisation, and DE for DropEdge(naive).
[2] All the unparenthesized numbers are the average of 5 random seeds.
[3] the numbers in parentheses are comparable results in original papers (Rong et al., 2019; Zhou et al., 2020; Chen et al., 2020b). If there were no corresponding implementation, then the location is left blankly

Appendix B. Over-Smoothing Metrics in Cora

Table 4: Instance Information Gain on the Cora dataset

Method	Layers = 2			Layers = 15			Layers = 30		
	GCN	GAT	GCNII	GCN	GAT	GCNII	GCN	GAT	GCNII
NN	0.12071	0.11381	0.11926	0.00432	0.00750	0.12845	0.00182	0.00253	0.12704
NN + DE	0.08715	0.11209		0.00546	0.00709		0.00313	0.00253	
BN	0.10415	0.10761	0.10684	0.09319	0.01781	0.10615	0.09215	0.00542	0.10672
BN + DE	0.06772	0.09937		0.03862	0.00000		0.02197	0.00000	
PN	0.10648	0.11028	0.09675	0.09126	0.07864	0.10199	0.08047	0.04212	0.09471
PN + DE	0.06809	0.10385		0.03750	0.02752		0.01777	0.02404	
DGN	0.12040	0.11691	0.11962	0.09905	0.08885	0.12525	0.09672	0.06895	0.12262
DGN + DE	0.08580	0.11201		0.05068	0.00377		0.01929	0.00000	

Appendix C. Over-Smoothing metrics in CiteSeer

Table 5: Group Distance on the CiteSeer dataset

Method	Layers = 2			Layers = 15			Layers = 30		
	GCN	GAT	GCNII	GCN	GAT	GCNII	GCN	GAT	GCNII
NN	2.0007	2.0677	1.9858	1.1278	1.1086	2.4045	1.1269	1.0447	2.4126
NN + DE	1.3037	1.7822		1.0363	1.0500		1.0343	1.0460	
BN	1.3660	1.6469	1.6230	1.27749	1.1097	1.4699	1.2914	1.0802	1.4068
BN + DE	1.1234	1.3625		1.0230	1.0652		1.0087	1.0375	
PN	1.5213	1.6353	1.1648	1.2493	1.3896	1.4083	1.2180	1.2524	1.2790
PN + DE	1.1306	1.4555		1.0200	1.0752		1.0158	1.0316	
DGN	1.9075	2.0100	1.8145	1.3573	1.5256	2.4118	1.3485	1.4522	2.1894
DGN + DE	1.2388	1.7135		1.0763	-33.1659		1.0360	1.0644	

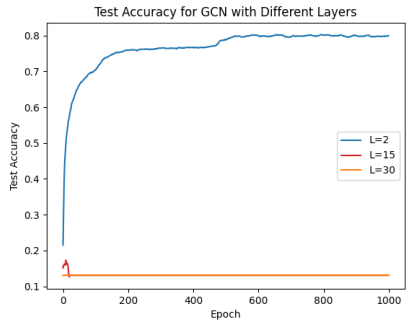
[1] NN stands for No Normalisation, BN for BatchNorm, PN for PairNorm, DGN for Differentiable Group Normalisation, and DE for DropEdge(naive). [2] All the numbers are the average of 5 random seeds.

Table 6: Instance Information Gain on the CiteSeer dataset

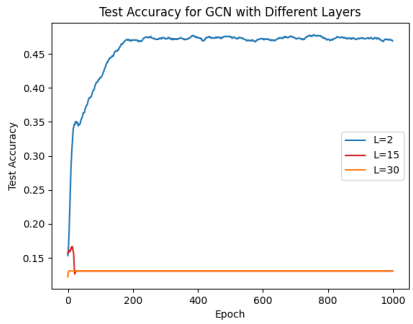
Method	Layers = 2			Layers = 15			Layers = 30		
	GCN	GAT	GCNII	GCN	GAT	GCNII	GCN	GAT	GCNII
NN	0.0801	0.0789	0.7589	0.0047	0.0113	0.0842	0.0038	0.0044	0.0848
NN + DE	0.0572	0.0736		-0.0006	0.0084		-0.0012	0.0045	
BN	0.0556	0.0682	0.0675	0.0485	0.0067	0.0710	0.0462	0.0020	0.0683
BN + DE	0.0426	0.0544		0.0196	0.0019		0.0143	0.0000	
PN	0.0678	0.0711	0.0375	0.0471	0.0341	0.0626	0.0465	0.0274	0.0486
PN + DE	0.0449	0.0639		0.0237	0.0233		0.0140	0.0248	
DGN	0.0782	0.0775	0.0771	0.0546	0.0460	0.0854	0.0514	0.0371	0.0834
DGN + DE	0.0523	0.0715		0.0316	0.0062		0.0137	0.0087	

[1] NN stands for No Normalisation, BN for BatchNorm, PN for PairNorm, DGN for Differentiable Group Normalisation, and DE for DropEdge(naive). [2] All the numbers are the average of 5 random seeds.

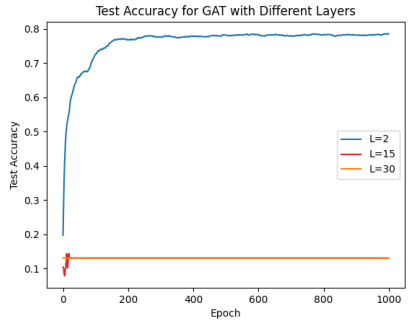
Appendix D. Test Accuracy Curves In Cora



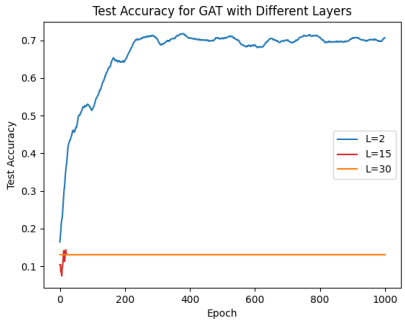
(a) GCN



(b) GCN + DropEdge

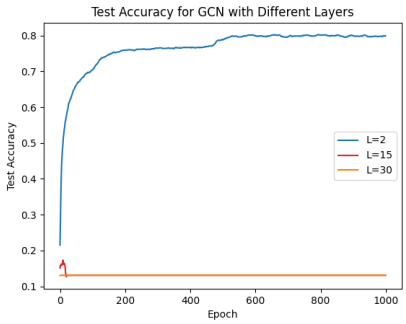


(c) GAT



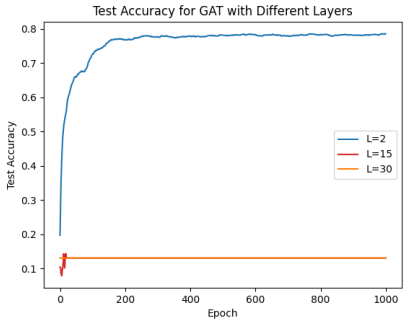
(d) GAT + DropEdge

Figure 1: DropEdge vs Baseline

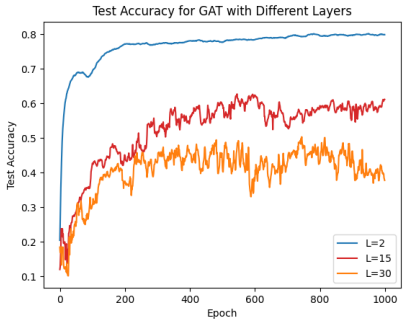


(a) GCN

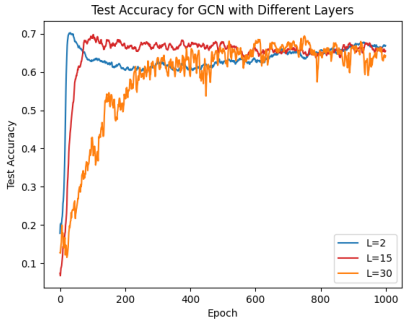
(b) GCN + DGN



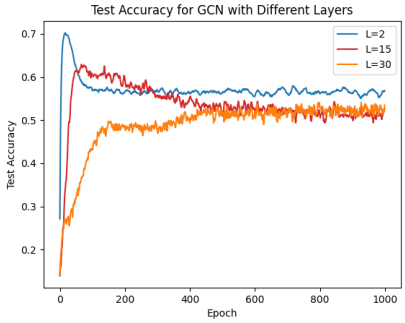
(c) GAT



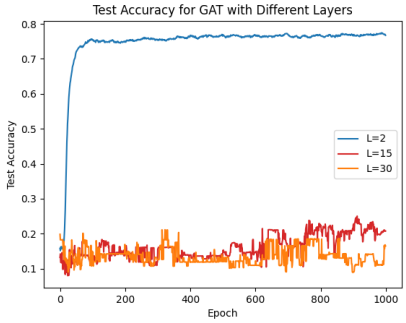
(d) GAT + DGN



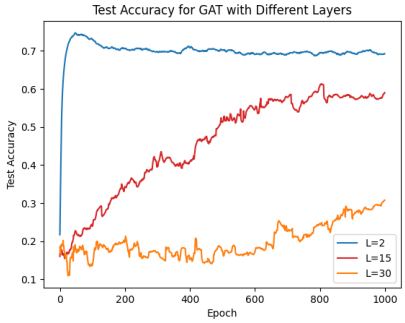
(e) GCN + BN



(f) GCN + PN

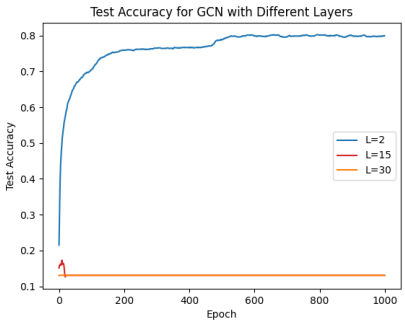


(g) GAT + BN

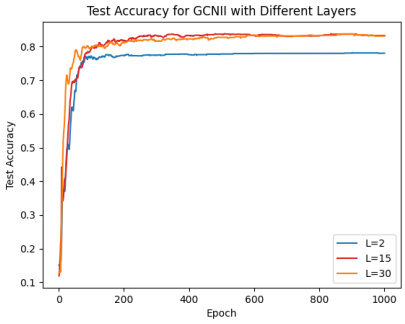


(h) GAT + PN

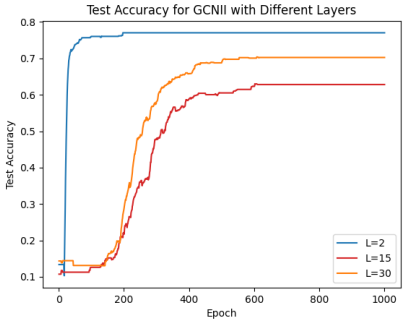
Figure 2: Normalisations vs Baseline



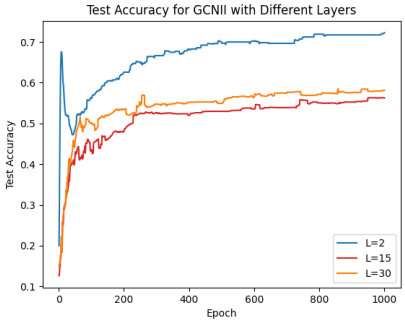
(a) GCN



(b) GCNII



(c) GCNII+BN



(d) GCNII+PN

Figure 3: GCNII vs GCN baseline

References

Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann Lecun. Spectral networks and locally connected networks on graphs. In *International Conference on Learning Representations (ICLR2014), CBLS, April 2014*, 2014.

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. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 3438–3445, 2020a.

Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks. In *International conference on machine learning*, pages 1725–1735. PMLR, 2020b.

Yihao Chen, Xin Tang, Xianbiao Qi, Chun-Guang Li, and Rong Xiao. Learning graph normalization for graph neural networks. *Neurocomputing*, 493:613–625, 2022. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2022.01.003. URL `https://www.sciencedirect.com/science/article/pii/S0925231222000030`.

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *CoRR*, abs/1606.09375, 2016. URL `http://arxiv.org/abs/1606.09375`.

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 Systems with Applications*, 174:114711, 2021.

Vijay Prakash Dwivedi, Chaitanya K Joshi, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks. 2020.

Johannes Gasteiger, Aleksandar Bojchevski, and Stephan Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. *arXiv preprint arXiv:1810.05997*, 2018.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL `http://arxiv.org/abs/1512.03385`.

Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

Wenbing Huang, Yu Rong, Tingyang Xu, Fuchun Sun, and Junzhou Huang. Tackling over-smoothing for general graph convolutional networks. *arXiv preprint arXiv:2008.09864*, 2020.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr, 2015.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *CoRR*, abs/1609.02907, 2016. URL `http://arxiv.org/abs/1609.02907`.

Guohao Li, Matthias Müller, Ali K. Thabet, and Bernard Ghanem. Can gcns go as deep as cnns? *CoRR*, abs/1904.03751, 2019. URL `http://arxiv.org/abs/1904.03751`.

Qimai Li, Zhichao Han, and Xiao-ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), Apr. 2018. doi: 10.1609/aaai.v32i1.11604. URL `https://ojs.aaai.org/index.php/AAAI/article/view/11604`.

Meng Liu, Hongyang Gao, and Shuiwang Ji. Towards deeper graph neural networks. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 338–348, 2020.

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

Kenta Oono and Taiji Suzuki. Graph neural networks exponentially lose expressive power for node classification. *arXiv preprint arXiv:1905.10947*, 2019.

Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. Dropedge: Towards deep graph convolutional networks on node classification. *arXiv preprint arXiv:1907.10903*, 2019.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

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

Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In *International conference on machine learning*, pages 6861–6871. PMLR, 2019.

Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. *CoRR*, abs/1806.03536, 2018. URL `http://arxiv.org/abs/1806.03536`.

Chaoqi Yang, Ruijie Wang, Shuochao Yao, Shengzhong Liu, and Tarek Abdelzaher. Revisiting over-smoothing in deep gcns. *arXiv preprint arXiv:2003.13663*, 2020.

Zhilin Yang, William W. Cohen, and Ruslan Salakhutdinov. Revisiting semi-supervised learning with graph embeddings. *CoRR*, abs/1603.08861, 2016. URL `http://arxiv.org/abs/1603.08861`.

Lingxiao Zhao and Leman Akoglu. Pairnorm: Tackling oversmoothing in gnns. *arXiv preprint arXiv:1909.12223*, 2019.

Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, and Maosong Sun. Graph neural networks: A review of methods and applications. *CoRR*, abs/1812.08434, 2018. URL `http://arxiv.org/abs/1812.08434`.

Kaixiong Zhou, Xiao Huang, Yuening Li, Daochen Zha, Rui Chen, and Xia Hu. Towards deeper graph neural networks with differentiable group normalization. *Advances in neural information processing systems*, 33:4917–4928, 2020.

Kuangqi Zhou, Yanfei Dong, Kaixin Wang, Wee Sun Lee, Bryan Hooi, Huan Xu, and Jiashi Feng. Understanding and resolving performance degradation in deep graph convolutional networks. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pages 2728–2737, 2021.