TOWARDS FEATURE OVERCORRELATION IN DEEPER GRAPH NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Graph neural networks (GNNs) have achieved great success in graph representation learning, which has tremendously facilitated various real-world applications. Nevertheless, the performance of GNNs significantly deteriorates when the depth increases. Recent researches have attributed this phenomenon to the oversmoothing issue, which indicates that the learned node representations are highly indistinguishable. In this paper, we observe a new issue in deeper GNNs, i.e., feature overcorrelation, and perform a thorough study to deepen our understanding on this issue. In particular, we demonstrate the existence of feature overcorrelation in deeper GNNs, reveal potential reasons leading to this issue, and validate that overcorrelation and oversmoothing present different patterns though they are related. Since feature overcorrelation indicates that GNNs encode less information and can harm the downstream tasks, it is of great significance to mitigate this issue. Therefore, we propose the DeCorr, a general framework to effectively reduce feature correlation for deeper GNNs. Experimental results on various datasets demonstrate that DeCorr can help train deeper GNNs effectively and is complementary to methods tackling oversmoothing.

1 INTRODUCTION

Graphs describe pairwise relations between entities for real-world data from various domains, which are playing an increasingly important role in many applications, including node classification (Kipf & Welling, 2016), recommender systems (Fan et al., 2019; Ying et al., 2018a) and drug discovery (Duvenaud et al., 2015). To facilitate these applications, it is particularly important to extract effective representations for graphs. In recent years, graph neural networks (GNNs) have achieved tremendous success in representation learning on graphs (Zhou et al., 2018; Wu et al., 2019). Most GNNs follow a message-passing mechanism to learn a node representation by propagating and transforming representations of its neighbors (Gilmer et al., 2017b), which significantly helps them in capturing the complex information of graph data.

Despite the promising results, it has been observed that deeply stacking GNN layers often results in significant performance deterioration (Li et al., 2018; Zhao & Akoglu, 2020). Hence, to enable larger receptive filed and larger model capacity, increasing efforts have been made on developing deeper GNNs (Zhao & Akoglu, 2020; Zhou et al., 2020; Liu et al., 2020; Rong et al., 2019; Chen et al., 2020b). Most of them attribute the performance deterioration to the oversmoothing issue. In other words, the learned node representations become highly indistinguishable when stacking many GNN layers. In this work, we observe a different issue, overcorrelation, which indicates that deeply stacking GNN layers renders the learned feature dimensions highly correlated. High correlation indicates high redundancy and less information encoded by the learned dimensions, which can harm downstream performance.

We first systematically study the overcorrelation issue in deeper GNNs by answering three questions: (1) does the overcorrelation issue exist? (2) what contributes to the overcorrelation issue? (3) what is the relationship and difference between overcorrelation and oversmoothing? Through exploring these questions, we find that when stacking more GNN layers, generally feature dimensions become more correlated and node representations become more smooth; but they present distinct patterns. Furthermore, through empirical study and theoretical analysis, we show that overcorrelation can be attributed to both propagation and transformation and we further demonstrate that in the extreme

case of oversmoothing, the feature dimensions are definitely overcorrelated but not vice versa. In other words, the overcorrelated feature dimensions does not necessarily indicate oversmoothed node representations. These observations suggest that overcorrelation and oversmoothing are related but not identical. Thus, handling overcorrelation has the potential to provide a new and complementary perspective to train deeper GNNs.

After validating the existence of the overcorrelation issue and understanding its relationship with oversmoothing, we aim to reduce the feature correlation and consequently enrich the encoded information for the representations, thus enabling deeper GNNs. In particular, we propose a general framework, DeCorr, to address the overcorrelation issue by introducing an explicit feature decorrelation component and a mutual information maximization component. The explicit feature decorrelation component directly regularizes the correlation on the learned dimensions while the mutual information maximization component encourages the learned representations to preserve a fraction of information from the input features.

Our contributions. Our contributions can be summarized as follows: (1) We introduce a new perspective in deeper GNNs, i.e., feature overcorrelation, and further deepen our understanding on this issue via empirical experiments and theoretical analysis. (2) We propose a general framework to effectively reduce the feature correlation and encourage deeper GNNs to encode less redundant information. (3) Extensive experiments have demonstrated the proposed framework can help enable deeper GNNs and is complementary to existing techniques tackling the oversmoothing issue.

2 Background and Related Work

We denote a graph as $\mathcal{G} = (\mathbf{A}, \mathbf{X})$, where $\mathbf{A} \in \{0,1\}^{N\times N}$ is the adjacency matrix, and $\mathbf{X} \in \mathbb{R}^{N\times d}$ indicates the node feature matrix with d as the number of features.

Preliminaries of Graph Neural Networks. A graph neural network model usually consists of several GNN layers, where each layer takes the output of the previous layer as the input. Each GNN layer updates the representations of all nodes by propagating and transforming representations of their neighbors. More specifically, the l-th GNN layer can be described as follows:

$$\mathbf{H}_{i,:}^{(l)} = \text{Transform}\left(\text{Propagate}\left(\mathbf{H}_{j,:}^{(l-1)} \mid v_j \in \mathcal{N}(v_i) \cup \{v_i\}\right)\right), \tag{1}$$

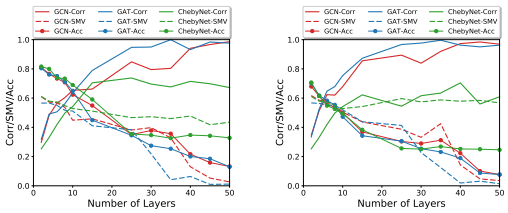
where $\mathbf{H}_{i,:}^{(l)}$ denotes the representation for node v_i after l-th GNN layer and $\mathcal{N}(v_i)$ is the set of neighboring nodes of node v_i. For an L layer graph neural network model, we adopt $\mathbf{H}^{(L)}$ as the final representation of all nodes, which can be utilized for downstream tasks. For example, for node classification task, we can calculate the discrete label probability distribution for node v_i as follows:

$$\hat{y}_{v_i} = \text{softmax}\left(\mathbf{H}_{i,:}^{(k)}\right), \tag{2}$$

where $\hat{y}_{v_i}[j]$ corresponds to the probability of predicting node v_i as the j-th class.

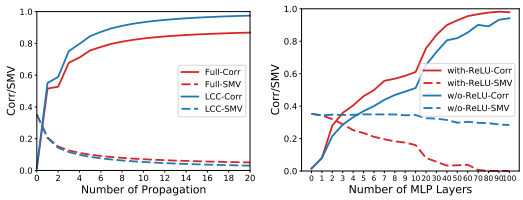
Related Work. Recent years have witnessed great success achieved by graph neural networks (GNNs) in graph representation learning, which has tremendously advanced various graph tasks (Ying et al., 2018b; Yan et al., 2018; Marcheggiani et al., 2018; Zitnik et al., 2018). In general, there are two main families of GNN models, i.e. spectral-based methods and spatial-based methods. The spectral-based GNNs utilize graph convolution based on graph spectral theory (Shuman et al., 2013) to learn node representations (Bruna et al., 2013; Henaff et al., 2015; Defferrard et al., 2016b; Kipf & Welling, 2016), while spatial-based GNNs update the node representation by aggregating and transforming information from its neighbors (Veličković et al., 2017; Hamilton et al., 2017; Gilmer et al., 2017a). For a thorough review, we please refer the reader to recent surveys (Zhou et al., 2018; Wu et al., 2019).

However, recent studies have revealed that deeply stacking GNN layers can lead to significant performance deterioration, which is often attributed to the oversmoothing issue (Zhao & Akoglu, 2020; Chen et al., 2020a), i.e, the learned node representations become highly indistinguishable. To address the oversmoothing issue and enable deeper GNNs, various methods have been proposed (Zhao & Akoglu, 2020; Chen et al., 2020a; Zhou et al., 2020; Rong et al., 2019; Chen et al., 2020b). For example, PairNorm (Zhao & Akoglu, 2020) is proposed to keep the total pairwise distance of node



(a) Cora (b) Citeseer

Figure 1: $Corr$ and SMV of learned representations for three GNNs together with test accuracy.



(a) Propagation (b) Transformation

Figure 2: $Corr$ and SMV on Cora when stacking more propagation and transformation .

representations constant through a normalization layer. Similarly, DGN (Zhou et al., 2020) also normalizes the node representation by normalizing each group of similar nodes independently to maintain the group distance ratio and instance information gain. Different from the normalization methods, DropEdge (Rong et al., 2019) proposes to randomly drop some edges from the graph, which has been shown to alleviate both oversmoothing and overfitting. However, most of the works targets at solving oversmoothing while overlooking the feature overcorrelation. In this paper, we perform a systematical study on the overcorrelation issue and provide effective solution to tackle it. We also provide the connections between the previous methods and overcorrelation in Section 3.3.

3 Preliminary Study

In this section, we investigate the issues of overcorrelation in deep graph neural networks through both empirical study and theoretical analysis. We observe that overcorrelation and oversmoothing are different though they could be related.

3.1 Overcorrelation and Oversmoothing

In this subsection, we demonstrate that stacking multiple graph neural network layers can sharply increase the correlation among feature dimensions. We choose one popular correlation measure, pearson correlation coefficient (Benesty et al., 2009), to evaluate the correlation between the learned dimensions in deep GNNs. Specifically, given two vectors $\mathbf{x} \in \mathbb{R}^N$ and $\mathbf{y} \in \mathbb{R}^N$, the pearson correlation coefficient between them can be formulated as follows:

$$\rho(\mathbf{x}, \mathbf{y}) = \frac{\sum_{i=1}^{N} (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{N} (x_i - \bar{x})^2 \sum_{i=1}^{N} (y_i - \bar{y})^2}}, \tag{3}$$

where $\bar{x}$ and $\bar{y}$ denote the mean value of $\mathbf{x}$ and $\mathbf{y}$, respectively. Essentially, pearson correlation coefficient normalizes the covariance between the two variables and measures how much two variables are linearly related to each other. The value of $\rho(\mathbf{x}, \mathbf{y})$ ranges from -1 to 1 – high absolute values of pearson correlation coefficient indicate that the variables are highly correlated and vice versa. Furthermore, we propose the metric $Corr$ to measure the correlation among all learned dimension pairs in the representation $\mathbf{X} \in \mathbb{R}^{N \times d}$ as,

$$Corr(\mathbf{X}) = \frac{1}{d(d-1)} \sum_{i \neq j} |p(\mathbf{X}_{:,i}, \mathbf{X}_{:,j})| \quad i, j \in [1, 2, \ldots, d], \tag{4}$$

where $\mathbf{X}_{:,i}$ denotes the i-th column of $\mathbf{X}$. Since we are also interested in the oversmoothing issue, we use the metric SMV proposed in (Liu et al., 2020), which uses normalized node representations to compute their Euclidean distance:

$$SMV(\mathbf{X}) = \frac{1}{N(N-1)} \sum_{i \neq j} D(\mathbf{X}_{i,:}, \mathbf{X}_{j,:}), \tag{5}$$

where $D(\cdot, \cdot)$ is the normalized Euclidean distance between two vectors. The smaller SMV is, the smoother the node representations are. Furthermore, it is worth noting that, (1) both $Corr$ and SMV are in $[0, 1]$; and (2) they are different measures from two perspectives – $Corr$ measures *dimension-wise* correlation while SMV measures *node-wise* smoothness.

Based on the aforementioned metrics, we investigate the overcorrelation and oversmoothing issues in three representative GNN models, i.e., GCN (Kipf & Welling, 2016), GAT (Veličković et al., 2017) and ChebyNet (Defferrard et al., 2016a). Specifically, we vary their depth from 2 to 50 and calculate the values of correlation ($Corr$) and smoothness (SMV) of the final representations, as shown in Figure 1. From the figure, we make following observations:

(1) When the number of layers increases, the correlation among dimensions increases significantly. For example, for 50-layer GCN and GAT, the $Corr$ values on the two datasets are larger than 0.95, which shows extremely high redundancy in the learned dimensions of deep GNNs.
(2) For the first a few layers (i.e., the number of layers is smaller than 10), the $Corr$ values increase sharply (roughly from 0.3 to 0.6) together with the drop of test accuracy. However, the SMV values do not change much (roughly from 0.6 to 0.5). That can explain the observation in Section 5.2 why methods addressing oversmoothing perform worse than shallow GNNs while our methods tackling overcorrelation can outperform shallow GNNs.
(3) In general, with the increase of the number of layers, the learned representations becomes more correlated and more smoothing. However they show very different patterns. This observation suggests that overcorrelation and oversmoothing are different though related. Thus, addressing them can help deeper GNNs from different perspectives and can be complementary. This is validated by our empirical results in Section 5.2.

It is evident from this study that (i) deep GNNs suffer the overcorrelation issue, i.e., the learned dimensions become more correlated as the number of layers increases; and (ii) overcorrelation and oversmoothing present different patterns with the increase of the number of layers. Next, we will analyze possible factors causing overcorrelation and discuss the problems of overcorrelated features.

3.2 Analysis on Overcorrelation

Propagation and transformation are two major components in graph neural networks as discussed in Section 2. In this subsection, we first show that both propagation and transformation can increase feature correlation. Then we discuss potential problems caused by overcorrelated features.

3.2.1 Propagation Can Lead to Higher Correlation

In this subsection, we will show that propagation in GNNs can cause higher feature correlation from both theoretical analysis and empirical evidence. In essence, the propagation in GNNs will lead to smoother representation as shown in (Li et al., 2018; Liu et al., 2020; Oono & Suzuki, 2019); their analysis suggests that applying infinite propagation can make the node representations in a connected graph to be proportional to each other, which we call extreme oversmoothed features in this work. Next we show that the dimensions of extreme oversmoothed features are correlated.

Proposition 3.1. *Given an extreme oversmoothed matrix $\mathbf{X}$ where each row is proportional to each other, we have $Corr(\mathbf{X})$=1.*

Proof. The detailed proof of this proposition can be found in Appendix B.

The above proposition indicates that multiple propagation can lead to higher correlation. Though the analysis is only for connected graph, if the training nodes are in the same component, their representations would still be overcorrelated and harm the performance of downstream tasks. More importantly, we empirically find that propagation can increase the feature correlation in both connected and disconnected graph. Specifically, we use the full graph (Full) and largest connected component (LCC) of Cora dataset, which originally consists of 78 connected components, to compute the propagation matrix $\hat{\mathbf{A}} = \tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}$. Then we apply multiple propagation on randomly generated node features of dimension 100 whose correlation is very close to 0. We vary the number of propagation (K) and illustrate the average values of $Corr(\hat{\mathbf{A}}^K\mathbf{X})$ for 100 runs in Figure 2a. Note that we include results on other datasets in Appendix A.1 due to page limit. As we can see from the figure, there is a clear trend that performing multiple propagation can cause uncorrelated features to eventually overcorrelated, no matter whether the graph is connected or not.

Despite that in the extreme case oversmoothing indicates overcorrelation, we show that the opposite of this statement could not hold. Specifically, we have the following proposition:

Proposition 3.2. *Given an extreme overcorrelated representation $\mathbf{X}$, i.e., $Corr(\mathbf{X}) = 1$, the rows of $\mathbf{X}$ are not necessarily proportional to each other.*

Proof. The detailed proof of this proposition can be found in Appendix B.

To demonstrate it more clearly, consider a 2×2 matrix with row vectors $v_1 = [1, 0]$ and $v_2 = [-0.1, 1.1]$. The pearson correlation coefficient between the column vectors is 1. But the normalized euclidean distance between the row vectors is 0.738 (the cosine similarity is -0.09).

3.2.2 TRANSFORMATION CAN LEAD TO HIGHER CORRELATION

In addition to propagation, we also find that transformation can make transformed features more correlated through empirical study. Specifically, we randomly generate uncorrelated node features of dimension 100 and apply K-layer multi layer perceptron (MLP) with hidden units of 16. We vary the value of K and plot the $Corr$ values for final representations in Figure 2b. Due to page limit, we include results on other datasets in Appendix A.1. Note we do not train the MLP but only focus on the forward pass of the neural network, i.e., the weights of MLP are randomly initialized. From the figure we can see that repeatedly applying (linear or non-linear) transformation will increase the correlation of feature dimensions. Intuitively, this is because transformation linearly combines the feature dimensions and increases the interaction between feature dimensions. As a result, repeated transformation could make each dimension of final representation contain similar amount of information from previous layers and become overcorrelated. On the other hand, backpropagation can to some extent alleviate the overcorrelation issue as the objective downstream task will guide the training process of parameters. However, training very deep neural networks can face other challenges like gradient vanishing/exploding and overfitting, thus failing to effectively reduce the representation correlation. That could also be the reason why trained deep GNNs still exhibit the overcorrelation issue as shown in Figure 1.

3.3 FURTHER DISCUSSIONS

Overcorrelation vs. Oversmoothing. The previous study suggests that overcorrelation and oversmoothing are neither identical nor independent. Their difference can be summarized as: oversmoothing is measured by node-wise smoothness while overcorrelation is measured by dimension-wise correlation. These two measures are essentially different. Correlated feature dimensions do not indicate similar node-wise features. As shown in Figure 2b, we can observe that SMV does not change much while $Corr$ goes up very quickly in the MLP without ReLU. On the other hand, their relations can be summarized as follows: (1) both overcorrelation and oversmoothing can make the learned representation encode less information and harm the downstream performance; and (2) both of them can be caused by multiple propagation since the extreme case of oversmoothing also suffers overcorrelation as demonstrated in Section 3.2.1 and Figure 2a.

Revisiting Previous Methods Tackling Oversmoothing. Since we have introduced the overcorrelation issue, next we revisit the previous methods tackling oversmoothing that have the potential to help alleviate the overcorrelation issue:

(1) *DropEdge* (Rong et al., 2019). DropEdge tackles the oversmoothing problem by randomly dropping edges in the graph. This can be beneficial to correlation reduction: (1) it can weaken the propagation process, thus alleviating overcorrelation; and (2) dropping edges can make the graph more disconnected and further reduce the feature correlation as we showed in Figure 2a.
(2) *Residual based methods* (Kipf & Welling, 2016; Chen et al., 2020b; Li et al., 2019). ResGCN (Kipf & Welling, 2016) equips GCN with residual connection and GCNII (Chen et al., 2020b) employs initial residual and identity mapping. Such residual connection bring in additional information from previous layers that can help the final representation encode more useful information and thus alleviate the overcorrelation issue.
(3) *Normalization and other methods* (Zhao & Akoglu, 2020; Zhou et al., 2020). These methods target at pushing GNNs to learn distinct representations. They also can implicitly reduce the feature correlation.

4 THE PROPOSED FRAMEWORK

In this section, we introduce the proposed framework, DeCorr, to tackle the overcorrelation issue. Specifically, the framework consists of two components: (1) explicit feature decorrelation which directly reduces the correlation among feature dimensions; and (2) mutual information maximization which maximizes the mutual information between the input and the representations to enrich the information, thus implicitly making features more independent.

4.1 EXPLICIT FEATURE DIMENSION DECORRELATION

In order to decorrelate the learned feature dimensions, we propose to minimize the correlation among the dimensions of the learned representations. There are many metrics to measure the correlation for variables including linear metrics and non-linear metrics. As discussed in Section 3.1, though measured by linear correlation metric, deep GNNs are shown to have high correlation among dimensions of the learned representations. For simplicity, we propose to use the covariance, as a proxy for pearson correlation coefficient, to minimize the correlation among representation dimensions. Specifically, given a set of feature dimensions $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_d\}$ with $\mathbf{x}_i \in \mathbb{R}^{N\times 1}$, we aim to minimize the following loss function,

$$\frac{1}{N-1}\left(\sum_{i,j,i\neq j}\left((\mathbf{x}_i-\bar{\mathbf{x}}_i)^\top(\mathbf{x}_j-\bar{\mathbf{x}}_j)\right)^2+\sum_i\left((\mathbf{x}_i-\bar{\mathbf{x}}_i)^\top(\mathbf{x}_i-\bar{\mathbf{x}}_i)-1\right)^2\right), \quad (6)$$

where $\bar{\mathbf{x}}_i$ is the vector with all elements as the mean value of $\mathbf{x}_i$. In Eq. (6), minimizing the first term reduces the covariance among different feature dimensions and when the first term is zero, the dimensions will become uncorrelated. By minimizing the second term, we are pushing the norm of each dimension (after subtracting the mean) to be 1. We then rewrite Eq. (6) as the matrix form,

$$\frac{1}{N-1}\left\|(\mathbf{X}-\bar{\mathbf{X}})^\top(\mathbf{X}-\bar{\mathbf{X}})-\mathbf{I}_d\right\|_F^2, \quad (7)$$

where $\bar{\mathbf{X}}=[\bar{\mathbf{x}}_1,\bar{\mathbf{x}}_2,\ldots,\bar{\mathbf{x}}_d]\in\mathbb{R}^{N\times d}$ and $\|\cdot\|_F$ indicates the Frobenius norm. It is worth noting that the gradient of $\left\|\mathbf{X}^\top\mathbf{X}-\mathbf{I}\right\|_F^2$ is calculated as $4\mathbf{X}(\mathbf{X}^\top\mathbf{X}-\mathbf{I})$. Thus, the gradient calculation has a complexity of $O(N^2d^2)$, which is not scalable when the graph size is extremely large in the real world applications (e.g., millions of nodes). To deal with this issue, instead of using all nodes to calculate the covariance, we propose to apply Monte Carlo sampling to sample $\sqrt{N}$ nodes with equal probability to estimate the covariance in Eq. (6). Then the complexity for calculating the gradient reduces to $O(Nd^2)$, which linearly increase with the graph size. In addition, the sampling strategy injects randomness in the model training process and thus can help achieve better generalization.

Minimizing the loss function in Eq. (7) is analogous to minimizing the following decorrelation loss ℓ_D where we normalize the two terms by diving their Frobenius norm in Eq. (7) to make $0\leq\ell_D\leq 2$:

$$\ell_D(\mathbf{X})=\left\|\frac{(\mathbf{X}-\bar{\mathbf{X}})^\top(\mathbf{X}-\bar{\mathbf{X}})}{\left\|(\mathbf{X}-\bar{\mathbf{X}})^{\mathbf{T}}(\mathbf{X}-\bar{\mathbf{X}})\right\|_F}-\frac{\mathbf{I}_d}{\sqrt{d}}\right\|_F, \quad (8)$$

where $\mathbf{X}$ can be the output representation matrix for the current GNN layer. Since we hope the representation after each layer can be less correlated, the final decorrelation loss for the explicit representation decorrelation component is formulated as:

$$\mathcal{L}_D=\sum_{i=1}^{K-1}\ell_D\left(\mathbf{H}^{(i)}\right), \quad (9)$$

where $\mathbf{H}^{(i)}$ denotes the hidden representation at the i-th layer and K stands for the total number of layers. By minimizing $\mathcal{L}_D$, we explicitly force the representation after each layer to be less correlated, thus mitigating the overcorrelation issue when developing deep GNNs.

4.2 MUTUAL INFORMATION MAXIMIZATION

In Section 3.1, we have demonstrated that the final learned features can be of high redundancy and encode little useful information. To address this issue, in addition to directly constraining on the correlation of features, we propose to further enrich the encoded information by maximizing the mutual information (MI) between the input and the learned features. The motivation comes from independent component analysis (ICA) (Bell & Sejnowski, 1995). The ICA principle aims to learn representations with low correlation among dimensions while maximizing the MI between the input and the representation. Since deeper GNNs encode less information in the representations, the MI maximization process can ensure that the learned representations retain a fraction of information from the input even if we stack many layers. Specifically, given two random variables A and B, we formulate the MI maximization process as follows:

$$\max \mathbf{MI}(A,B)=H(A)-H(A|B)=H(B)-H(B|A) \quad (10)$$

where $H(\cdot)$ denotes the entropy function and $\text{MI}(A, B)$ measures dependencies between A and B. However, maximizing mutual information directly is generally intractable when A or B is obtained through neural networks (Paninski, 2003), thus we resort to maximizing a lower bound on $\text{MI}(A, B)$. Specifically, we follow MINE (Belghazi et al., 2018) to estimate a lower bound of the mutual information by training a classifier to distinguish between sample pairs from the joint distribution $P(A, B)$ and those from $P(A)P(B)$. Formally, this lower bound of mutual information can be described as follows:

$$\text{MI}(A, B) \geq \mathbb{E}_{P(A,B)} \left[\mathcal{D}(A, B)\right] - \log \mathbb{E}_{P(A)P(B)} \left[e^{\mathcal{D}(A,B)}\right], \tag{11}$$

where $\mathcal{D}(A, B)$ is a binary discriminator. Hence, to maximize the mutual information between k-th layer hidden representation $\mathbf{H}^{(k)}$ and input feature $\mathbf{X}$, denoted as $\text{MI}(\mathbf{H}^{(k)}, \mathbf{X})$, we minimize the following objective:

$$\ell_M(\mathbf{H}^{(k)}, \mathbf{X}) = -\mathbb{E}_{P(\mathbf{h}_i^{(k)}, \mathbf{x}_i)} \left[\mathcal{D}(\mathbf{h}_i^{(k)}, \mathbf{x}_i)\right] + \log \mathbb{E}_{P(\mathbf{h}^{(k)})P(\mathbf{x})} \left[e^{\mathcal{D}(\mathbf{h}_i^{(k)}, \mathbf{x}_i)}\right], \tag{12}$$

where $\mathbf{h}_i^{(k)}$ and $\mathbf{x}_i$ are the hidden representation and input feature for node v_i, respectively; the discriminator $\mathcal{D}(\cdot, \cdot)$ is modeled as a bilinear layer. In practice, in each batch, we sample a set of $\{(\mathbf{h}_i^{(k)}, \mathbf{x}_i)\}_{i=1}^{B}$ from the joint distribution $P(\mathbf{h}_i^{(k)}, \mathbf{x}_i)$ to estimate the first term in Eq. (12) and then shuffle $\mathbf{x}_i$ in the batch to generate "negative pairs" for estimating the second term.

Although we can apply the above loss function for each layer of deep GNNs as we did in Section 4.1, we only apply ℓ_M every t layers to accelerate the training process as:

$$\mathcal{L}_M = \sum\nolimits_{i \in [t, 2t, 3t, \ldots, \frac{K-1}{t}t]} \ell_M \left(\mathbf{H}^{(i)}\right). \tag{13}$$

We empirically observe that a small value of 5 for t is sufficient to achieve satisfying performance.

Overall Optimization Objective. We jointly optimize the classification loss along with decorrelation loss and MI maximization loss. The overall objective function can be stated as:

$$\mathcal{L} = \mathcal{L}_{\text{class}} + \alpha \mathcal{L}_D + \beta \mathcal{L}_M \tag{14}$$

where $\mathcal{L}_{\text{class}}$ is classification loss; α and β are the hyper-parameters that control the contribution of $\mathcal{L}_D$ and $\mathcal{L}_M$, respectively. We provide complexity analysis of the proposed algorithm in Appendix C.

5 Experiment

In this section, we evaluate the effectiveness of the proposed framework under various settings and aim to answer the following research questions: **RQ1.** Can DeCorr help train deeper GNNs? **RQ2.** By enabling deeper GNNs, is DeCorr able to help GNNs achieve better performance? **RQ3.** Can DeCorr be equipped with methods that tackle oversmoothing and serve as a complementary technique? **RQ4.** How do different components affect the performance of the proposed DeCorr?

5.1 Experimental Settings

To validate the proposed framework, we conduct experiments on 9 benchmark datasets, including Cora, Citeseer, Pubmed (Sen et al., 2008), CoauthorCS (Shchur et al., 2018), Chameleon, Texas, Cornell, Wisconsin and Actor (Pei et al., 2020). Following (Zhao & Akoglu, 2020; Zhou et al., 2020), we also create graphs by removing features in validation and test sets for Cora, Citeseer, Pubmed and CoauthorCS. The statistics of these datasets and data splits can be found in Appendix D. Furthermore, we consider three basic GNN models, GCN (Kipf & Welling, 2016), GAT (Veličković et al., 2017) and ChebyNet (Defferrard et al., 2016a), and equip them with the following methods tackling the oversmoothing issue: PairNorm (Zhao & Akoglu, 2020), BatchNorm (Ioffe & Szegedy, 2015), DGN (Zhou et al., 2020), and DropEdge (Rong et al., 2019). We implemented our proposed framework based on the code provided by DGN (Zhou et al., 2020), which uses Pytorch (Paszke et al., 2017) and PyTorch Geometric (Fey & Lenssen, 2019). Without specific mention, we follow previous settings in (Zhao & Akoglu, 2020; Zhou et al., 2020): train with a maximum of 1000 epochs using the Adam optimizer (Kingma & Ba, 2014), run each experiment 5 times and report the average. Detailed parameter settings can be found in Appendix E.

Dataset	Method	GCN			GAT			ChebyNet		
		L2	L15	L30	L2	L15	L30	L2	L15	L30
Cora	None	82.2	18.1	13.1	80.9	16.8	13.0	81.7	43.0	33.4
	BatchNorm	73.9	70.3	67.2	77.8	33.1	25.0	70.3	66.0	61.7
	PairNorm	71.0	67.2	64.3	74.4	49.6	30.2	67.6	58.2	49.7
	DropEdge	**82.8**	70.5	45.4	81.5	66.3	51.0	81.5	67.1	55.7
	DGN	82.0	75.2	73.2	81.1	71.8	51.3	81.5	**76.3**	59.4
	DeCorr (Ours)	82.2	**77.0**	**73.4**	**81.6**	**76.0**	**54.3**	**81.8**	73.9	**65.4**
Citeseer	None	70.6	15.2	9.4	70.2	22.6	7.7	67.3	38.0	28.3
	BatchNorm	51.3	46.9	47.9	61.5	28.0	21.4	51.3	38.2	37.4
	PairNorm	60.5	46.7	47.1	62.0	41.4	33.3	53.2	37.6	34.6
	DropEdge	71.7	43.3	31.6	69.8	52.6	36.1	69.8	45.8	40.7
	DGN	69.5	53.1	52.6	69.3	52.6	45.6	67.3	49.3	47.0
	DeCorr (Ours)	**72.1**	**67.7**	**67.3**	**70.6**	**63.2**	**46.9**	**72.6**	**56.0**	**53.2**
Pubmed	None	79.3	22.5	18.0	77.8	37.5	18.0	78.4	49.5	43.5
	BatchNorm	74.9	73.7	70.4	76.2	56.2	46.6	73.6	68.0	69.1
	PairNorm	71.1	70.6	70.4	72.4	68.8	58.2	73.4	67.6	62.3
	DropEdge	78.8	74.0	62.1	77.4	72.3	64.7	78.7	73.3	68.4
	DGN	79.5	76.1	76.9	77.5	75.9	73.3	78.6	71.0	70.5
	DeCorr (Ours)	**79.6**	**78.1**	**77.3**	**78.1**	**77.5**	**74.1**	**78.7**	**77.0**	**72.9**
CoauthorCS	None	92.3	72.2	3.3	91.5	6.0	3.3	92.9	71.7	35.2
	BatchNorm	86.0	78.5	**84.7**	89.4	77.7	16.7	84.1	77.2	80.7
	PairNorm	77.8	69.5	64.5	85.9	53.1	48.1	79.1	51.5	57.9
	DropEdge	92.2	76.7	31.9	91.2	75.0	52.1	92.9	76.5	68.1
	DGN	92.3	83.7	84.4	**91.8**	**84.5**	75.5	92.7	84.0	80.4
	DeCorr (Ours)	**92.4**	**86.4**	84.5	91.3	83.5	**77.3**	**93.0**	**86.1**	**81.3**

Table 1: Node classification accuracy (%) on different number of layers. (Bold: best)

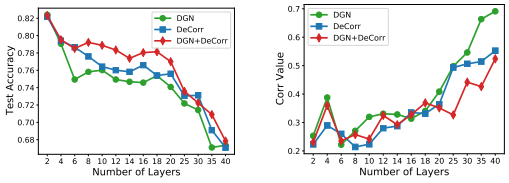
5.2 Experimental Results

Alleviating the Performance Drop in Deeper GNNs. We aim to study the performance of deeper GNNs when equipped with DeCorr and answer **RQ 1**. Following the previous settings in (Zhou et al., 2020), we perform experiments on Cora, Citeseer, Pubmed and CoauthorCS datasets. Note that "None" indicates vanilla GNNs without equipping any methods. We report the performance of GNNs with 2/15/30 layers in Table 1 due to space limit while similar patterns are observed in other numbers of layers. Table 1 show that the proposed DeCorr can greatly improve deeper GNNs. Furthermore, given the same layers, DeCorr consistently achieves the best performance for most cases and significantly slows down the performance drop. For example, on Cora dataset, DeCorr improves 15-layer GCN and 30-layer GCN by a margin of 58.9% and 60.3%, respectively. Given the improvement of the performance, the node representations become much more distinguishable than that in vanilla GNNs, thus alleviating the oversmoothing issue just as what the baseline methods do. The above observations indicate that dealing with feature overcorrelation can allow deeper GNNs and even achieves better performance than these only focusing on tackling oversmoothing.

It is worth noting that in most cases the propose framework can also boost the performance of 2-layer GNNs while the strongest baseline, DGN, fail to achieve that and sometimes deteriorate the performance. For instance, on Citeseer dataset, DeCorr improves GCN, GAT and ChebyNet by a margin of 1.5%, 0.4% and 5.3%, respectively. This suggests that decorrelating the learned features is generally helpful for improving the generalization of various models instead of only deeper models. In addition, we also provide result on missing feature setting in Appendix A.3 to answer **RQ 2**.

Combining DeCorr with DGN. To verify if DeCorr can serve as a complementary technique for methods tackling oversmoothing, we choose the strongest baseline, DGN, to be combined with DeCorr. Specifically, we vary the values of K in $\{2, 4, \ldots, 20, 25, \ldots, 40\}$ and report the test accuracy and $Corr$ values of DGN+DeCorr, DGN and DeCorr on the Cora dataset in Figure 3. From the figure, we make the following observations:

(1) When combining DGN with DeCorr, we can achieve even better performance than each individual method, which indicates that overcorrelation and oversmoothing are not identical. It provides new insights for developing deeper GNNs as we can combine the strategies tackling overcorrleation with ones solving oversmoothing to enable stronger deeper models.
(2) In Figure 3b, we can observe that DGN is not as effective as DeCorr in reducing feature correlation $Corr$. However, combining DGN with DeCorr can achieve even lower $Corr$ than DeCorr with the larger number of layers. It could be the reason that the training process of DeCorr be-

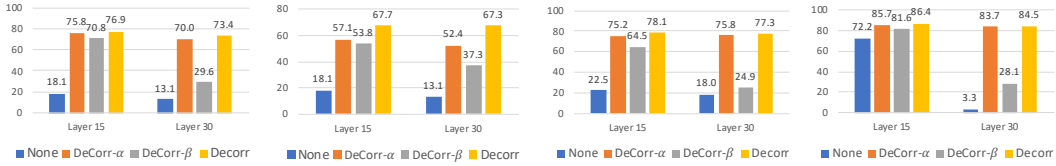


(a) Test Accuracy (b) Correlation

Figure 3: Test accuracy and $Corr$ on Cora.

	Cite.	Pubm.	Corn.	Wisc.	Cham.	Actor
GCN	71.1	79.0	28.2	52.2	45.9	26.9
GAT	70.8	78.5	42.9	58.4	49.4	28.5
APPNP	71.8	80.1	54.3	65.4	54.3	34.5
GCNII	73.5	79.9	70.8	76.7	52.8	34.5
+DeCorr	**73.8**	**80.3**	**75.4**	**80.8**	**54.1**	**35.3**
GCNII*	73.1	80.0	73.8	79.2	54.3	35.1
+DeCorr	**73.7**	**80.4**	**79.2**	**82.5**	**59.0**	**35.3**

Table 2: Node classification accuracy (%).



(a) Cora (b) Citeseer (c) Pubmed (d) CoauthorCS

Figure 4: Test accuracy and $Corr$ values on Cora dataset.

comes more stable when combined with DGN, which leads to a better minimization on feature correlation.

Combining DGN with Other Deep Models. In addition to those general frameworks which alleviate the oversmoothing issue, we further combine deep models GCNII and GCNII* (Chen et al., 2020b) (the depth is in $\{16, 32, 64\}$) with DeCorr. We perform experiments on eight benchmark datasets and report the average accuracy for 10 random seeds in Table 4. It shows that DeCorr can further improve both GCNII and GCNII* in most datasets. For example, DeCorr improves GCNII* by 0.6%, 0.4%, 5.4%, 1.9%, 3.3% and 4.7% on Citeseer, Pubmed, Cornell, Texas, Wisconsin and Chameleon, respectively. Such observation further supports that decorrelating features can help boost the model performance. Due to page limit, we include more results in Appendix A.

Ablation Study. We take a deeper look at the proposed framework to understand how each component affects its performance and answer the fourth question. Due to space limit, we focus on studying GCN while similar observations can be made for GAT and ChebyNet. We choose 15-layer and 30-layer GCN to perform the ablation study on four datasets. Specifically, we create the following ablations: **None**, vanilla GCN without other components; **DeCorr-**α, which removes the $\mathcal{L}_M$ loss but keeps $\mathcal{L}_D$; **DeCorr-**β, which removes the $\mathcal{L}_D$ loss but keeps $\mathcal{L}_M$. The results shown in Figure 4 domonstrate that both DeCorr-α and DeCorr-β can greatly improve the performance of 15-layer GCN, indicating that optimizing $\mathcal{L}_D$ and $\mathcal{L}_M$ can both help reduce feature correlation and boost the performance. Note that DeCorr-α and DeCorr-β can achieve comparable performance on 15-layer GCN. However, on 30-layer GCN, DeCorr-β does not bring as much improvement as DeCorr-α does on the four datasets. This observation suggests that when GNNs are very deep, explicit feature decorrelation ($\mathcal{L}_D$) is of more significance than the implicit method ($\mathcal{L}_M$).

6 Conclusion

Graph neural networks suffer severe performance deterioration when deeply stacking layers. Recent studies have shown that the oversmoothing issue is the major cause of this phenomenon. In this paper, we introduce a new perspective in deeper GNNs, i.e., feature overcorrelation, and perform both theoretical and empirical studies to deepen our understanding on this issue. We find that overcorrelation and oversmothing present different patterns while they are also related to each other. To address the overcorrelation issue, we propose a general framework, DeCorr, which aims to directly reduce the correlation among feature dimensions while maximizing the mutual in-formation between input and the representations. Extensive experiments have demonstrated that the proposed DeCorr can help deeper GNNs encode more useful information and achieve better performance under the settings of normal graphs and graphs with missing features. As one future work, we plan to explore the potential of applying DeCorr on various real-world applications such as recommender systems.

REFERENCES

Mohamed Ishmael Belghazi, Aristide Baratin, Sai Rajeswar, Sherjil Ozair, Yoshua Bengio, Aaron Courville, and R Devon Hjelm. Mine: mutual information neural estimation. *arXiv preprint arXiv:1801.04062*, 2018.

Anthony J Bell and Terrence J Sejnowski. An information-maximization approach to blind separation and blind deconvolution. *Neural computation*, 1995.

Jacob Benesty, Jingdong Chen, Yiteng Huang, and Israel Cohen. Pearson correlation coefficient. In *Noise reduction in speech processing*, pp. 1–4. Springer, 2009.

Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.

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*, 2020a.

Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks. *arXiv preprint arXiv:2007.02133*, 2020b.

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems*, pp. 3844–3852, 2016a.

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems*, pp. 3844–3852, 2016b.

David K Duvenaud, Dougal Maclaurin, Jorge Aguilera-Iparraguirre, Rafael Gómez-Bombarelli, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In *NIPS*, 2015.

Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. Graph neural networks for social recommendation. In *The World Wide Web Conference*, pp. 417–426, 2019.

Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. *arXiv preprint arXiv:1704.01212*, 2017a.

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017b.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, 2017.

Mikael Henaff, Joan Bruna, and Yann LeCun. Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163*, 2015.

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

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Guohao Li, Matthias Muller, Ali Thabet, and Bernard Ghanem. Deepgcns: Can gcns go as deep as cnns? In *CVPR*, 2019.

Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. *arXiv preprint arXiv:1801.07606*, 2018.

Meng Liu, Hongyang Gao, and Shuiwang Ji. Towards deeper graph neural networks. *arXiv preprint arXiv:2007.09296*, 2020.

Diego Marcheggiani, Joost Bastings, and Ivan Titov. Exploiting semantics in neural machine translation with graph convolutional networks. *arXiv preprint arXiv:1804.08313*, 2018.

Kenta Oono and Taiji Suzuki. Graph neural networks exponentially lose expressive power for node classification. In *ICLR*, 2019.

Liam Paninski. Estimation of entropy and mutual information. *Neural computation*, 15(6):1191–1253, 2003.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. Geom-gcn: Geometric graph convolutional networks. *arXiv preprint arXiv:2002.05287*, 2020.

Al Mamunur Rashid, George Karypis, and John Riedl. Learning preferences of new users in recommender systems: an information theoretic approach. *Acm Sigkdd Explorations Newsletter*, 10(2): 90–100, 2008.

Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. Dropedge: Towards deep graph convolutional networks on node classification. In *International Conference on Learning Representations*, 2019.

Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.

Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*, 2018.

David I Shuman, Sunil K Narang, Pascal Frossard, Antonio Ortega, and Pierre Vandergheynst. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE signal processing magazine*, 30(3):83–98, 2013.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S Yu. A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596*, 2019.

Sijie Yan, Yuanjun Xiong, and Dahua Lin. Spatial temporal graph convolutional networks for skeleton-based action recognition. *arXiv preprint arXiv:1801.07455*, 2018.

Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *KDD*. ACM, 2018a.

Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 974–983, 2018b.

Lingxiao Zhao and Leman Akoglu. Pairnorm: Tackling oversmoothing in {gnn}s. In *International Conference on Learning Representations*, 2020.

Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, and Maosong Sun. Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434*, 2018.

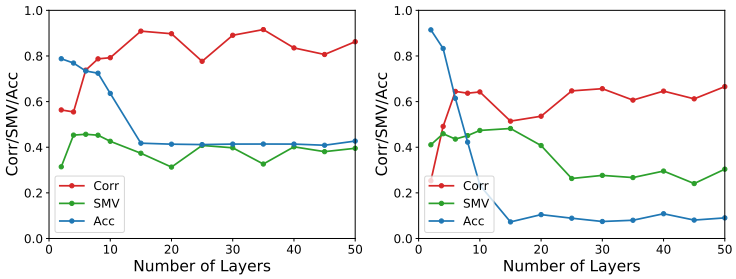
Kaixiong Zhou, Xiao Huang, Yuening Li, Daochen Zha, Rui Chen, and Xia Hu. Towards deeper graph neural networks with differentiable group normalization. In *Advances in neural information processing systems*, 2020.

Marinka Zitnik, Monica Agrawal, and Jure Leskovec. Modeling polypharmacy side effects with graph convolutional networks. *Bioinformatics*, 34(13):i457–i466, 2018.

A ADDITIONAL EXPERIMENTAL RESULTS

A.1 PRELIMINARY STUDY

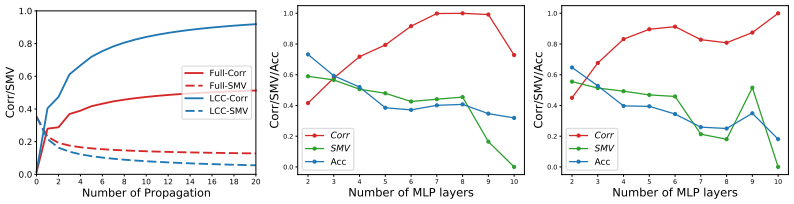
Correlation and Smoothness w.r.t. GCN Layers. We further plot the changes of $Corr$ and SMV w.r.t. number of GCN layers on Pubmed and CoauthorCS in Figure 5. From the figure, we make two observations. (1) With the increase of number of layers, the $Corr$ value tends to increase while the SMV does not change much on these two datasets. (2) The test accuracy drops with the increase of number of layers. Based on the two observations, we can conjecture that it is not oversmoothing but overcorrelation that causes the performance degradation in Pubmed and CoauthorCS.



(a) Pubmed (b) CoauthorCS

Figure 5: $Corr$, SMV and test accuracy w.r.t number of GCN layers.

Correlation and Smoothness w.r.t. Propagation/Transformation. We provide $Corr$, SMV w.r.t. number of propagation on Citeseer dataset in Figure 7a. We can make similar observations as we made in Section 3.2.1: propagation can lead to higher correlation. We further investigate the relationship between correlation and transformation. Different from Section 3.2.2, here we use $\mathbf{AX}$ as the input feature instead of random feature and we also train the MLP model for 200 epochs. The results are summarized in Figure 6. It is very clear that stacking more transformation layers tends to cause higher correlation and worse test accuracy. But SMV values are not always small, which indicates that oversmoothing is not a major concern in this case.

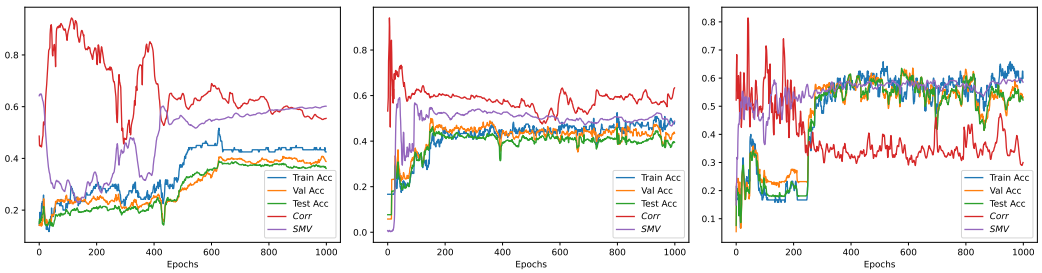


(a) Propagation: Citeseer (b) Transformation: Cora (c) Transformation:Citeseer

Figure 6: **(a):** $Corr$, SMV w.r.t. number of propagation on Citeseer dataset. **(b)(c):** $Corr$, SMV and test accuracy w.r.t number of MLP layers (trained model) when the input feature is $\mathbf{AX}$.

A.2 CORRELATION AND SMOOTHNESS OVER TRAINING EPOCHS

As shown in Table 1, DeCorr achieves significant improvement over other baselines in GAT on Citeseer dataset. Hence, we further investigate the reason behind it. Concretely, we plot the $Corr$, SMV, train/val/test accuracy for PairNorm, DGN and DeCorr when they are equipped to 15-layer GAT. The results are shown in Figure 7. From the figure, we can see that although PairNorm and DGN can maintain a high SMV value (around 0.6), their performance is still not very satisfying. Moreover, their $Corr$ values are much higher than DeCorr: around 0.6 in PairNorm and DGN, and around 0.35 in DeCorr. Based on this observation, we believe that overcorrelation is an important issue when enabling deeper GNN that researchers should pay attention to.



(a) PairNorm (b) DGN (c) DeCorr

Figure 7: The change of $Corr$, SMV and accuracy over training epochs for PairNorm, DGN and DeCorr.

A.3 Enabling Deeper and Better GNNs Under the Missing Feature Setting.

In Section 5.2, we have demonstrated the superiority of reducing feature overcorrelation in helping train deeper GNNs. However, the performance achieved by deeper GNNs are not as good as shallow ones (2-layer GNNs) as shown in Table 1. Then a natural question is: when deeper GNNs are beneficial? To answer this question, we remove the node features in validation and test set following the idea in Zhao & Akoglu (2020); Zhou et al. (2020). This scenario often happens in real world. For instance, new users in social networks are often lack of profile information while connecting with a few other users Rashid et al. (2008). To learn effective representations for those new users, we would need a larger number of propagation steps to propagate the attribute information from existing users. Hence, under this circumstance, deeper GNNs can behave much better than shallow ones. We vary the number of layers K in $\{1, 2, \ldots, 10, 15, \ldots, 30\}$ and report the performance in Table 3. Note that "None" indicates vanilla GNNs without equipping any methods and $\#K$ indicates the number of layers when the model achieves the best performance. Since we empirically find that BatchNorm is necessary in ChebyNet under this setting, we also include the variant of our model which equips with BatchNorm (DGN is also based on BatchNorm). From the table, we make the following two observations:

(1) Under the missing feature setting, the best performance is always achieved by the deeper models, i.e., the values of $\#K$ are always relatively larger. This suggests that more propagation steps is necessary to learn the good representation for nodes with missing features.
(2) DeCorr achieves the best performance in 8 out of the 12 cases and significantly outperforms shallow GNNs. For example, on Pubmed dataset, DeCorr achieves improvements of 36.9%, 33.9% and 19.5% on GCN, GAT and ChebyNet, respectively. It further demonstrates the importance of alleviating overcorrelation in developing deeper GNNs.

A.4 Combining with GCNII

In this subsection, we provide the full version of Table 4. Specifically, we report the mean values and variances of test accuracy over 10 random seeds. As shown in the table, we find that DeCorr can further improve both GCNII and GCNII* in most datasets. For example, DeCorr improves GCNII* by 0.6%, 0.4%, 5.4%, 1.9%, 3.3% and 4.7% on Citeseer, Pubmed, Cornell, Texas, Wisconsin and Chameleon, respectively. It supports that decorrelating features can improve model performance.

A.5 Comparison with DeepGCNs

B Proof

Proposition 1. *Given an extreme over-smoothed matrix* $\mathbf{X}$ *where each row is proportional to each other, we have* $Corr(\mathbf{X})$*=1.*

Model	Method	Cora		Citeseer		Pubmed		Coauth.	
		Acc	#K	Acc	#K	Acc	#K	Acc	#K
GCN	None	57.3	3	44.0	6	36.4	4	67.3	3
	BatchNorm	71.8	20	45.1	25	70.4	30	82.7	30
	PairNorm	65.6	20	43.6	25	63.1	30	63.5	4
	DropEdge	67.0	6	44.2	8	69.3	6	68.6	4
	DGN	**76.3**	20	**50.2**	30	72.0	30	83.7	25
	DeCorr	73.8	20	49.1	30	**73.3**	15	**84.3**	20
GAT	None	50.1	2	40.8	4	38.5	4	63.7	3
	BatchNorm	72.7	5	48.7	5	60.7	4	80.5	6
	PairNorm	68.8	8	50.3	6	63.2	20	66.6	3
	DropEdge	67.2	6	48.2	6	67.2	6	75.1	4
	DGN	**75.8**	8	**54.5**	5	72.3	20	83.6	15
	DeCorr	72.8	15	46.5	6	**72.4**	15	**83.7**	15
Cheby.	None	50.3	8	31.7	4	43.7	6	34.4	10
	BatchNorm	61.3	30	35.5	6	61.6	30	74.8	30
	PairNorm	53.7	15	35.8	30	53.7	25	41.5	20
	DropEdge	60.6	8	35.1	4	49.3	15	38.2	8
	DGN	61.0	30	35.0	30	56.3	25	75.1	30
	DeCorr	56.0	8	**35.9**	6	49.1	30	48.3	10
	DeCorr+BN*	**62.5**	30	35.4	6	**63.2**	30	**76.5**	30

* DeCorr+BN is the variant when BatchNorm (BN) is equipped with ours.

Table 3: Test accuracy (%) on missing feature setting.

	Cora	**Cite.**	**Pubm.**	**Corn.**	**Texas**	**Wisc.**	**Cham.**	**Actor**
GCN	81.5	71.1	79.0	28.2	52.7	52.2	45.9	26.9
GAT	83.1	70.8	78.5	42.9	54.3	58.4	49.4	28.5
APPNP	83.3	71.8	80.1	54.3	73.5	65.4	54.3	34.5
GCNII	85.5±0.5	73.5±0.7	79.9 ±0.5	70.8±7.8	75.7±5.0	76.7±5.0	52.8±5.4	34.5± 0.9
GCNII+DeCorr	**85.6±0.5**	**73.8±0.4**	**80.3±0.6**	**75.4±6.1**	**76.5±5.8**	**80.8±4.7**	**54.1±6.4**	**35.3±0.7**
GCNII*	**85.4±0.3**	73.1±0.5	80.0±0.5	73.8±4.5	78.4 ±6.5	79.2 ±4.5	54.3±4.1	35.1±0.8
GCNII*+DeCorr	85.3±0.3	**73.7±0.7**	**80.4±0.5**	**79.2±5.3**	**80.3±7.7**	**82.5±4.8**	**59.0±4.1**	**35.3±0.7**

Table 4: Node classification accuracy (%) for eight datasets.

Proof. Since each row is proportional to each other, each column will also be proportional to each other. We take two arbitrary dimensions (columns) of $\mathbf{X}$, and denote them as $[\mathbf{x}, w\mathbf{x}]$. The pearson coefficient $\rho(\mathbf{x}, w\mathbf{x}) = \frac{w}{|w|} \cdot \frac{(\mathbf{x}-\bar{\mathbf{x}})^\top(\mathbf{x}-\bar{\mathbf{x}})}{\|\mathbf{x}-\bar{\mathbf{x}}\|\|\mathbf{x}-\bar{\mathbf{x}}\|}$ is either 1 or -1. Since the correlation between any two arbitrary dimensions is 1 or -1, we have $Corr(\mathbf{X}) = 1$.[1] □

Proposition 2. *Given an extreme overcorrelated representation* $\mathbf{X}$*, the rows of* $\mathbf{X}$ *are not necessarily proportional to each other.*

Proof. We take two arbitrary dimensions of $\mathbf{X}$ and denote them as $[\mathbf{x}, w\mathbf{x}+b]$ since they are linearly dependent on each other. We further take two rows from the two dimensions, denoted as $[[x_1, wx_1 + b], [x_2, wx_2 + b]]$. If the two rows are proportional to each other, they need to satisfy $x_1(wx_2 + b) = x_2(wx_1 + b)$, which can be written as $bx_1 = bx_2$. When $bx_1 = bx_2$ does not hold, $\mathbf{X}$ will not be an extreme over-smoothed matrix. □

[1]Note that we exclude the case where $\|\mathbf{x} - \bar{\mathbf{x}}\| = 0$, i.e., $\mathbf{x}$ is a constant vector, the pearson correlation is undefined.

C COMPLEXITY ANALYSIS

We compare the proposed method with vanilla GNNs by analyzing the additional complexity in terms of model parameters and time. For simplicity, we assume that all hidden dimension is d and the input dimension is d_0.

Model Complexity. In comparison to vanilla GNNs, the only additional parameters we introduce are the weight matrix $\mathbf{W}$ in the bilinear layer of the discriminator $D(\cdot, \cdot)$ in Eq. (12) when scoring the agreement for positive/negative samples. Its complexity is $O(d_0 d)$, which does not depend on the graph size. Since the hidden dimension is usually much smaller than the number of nodes in the graph, the additional model complexity is negligible.

Time Complexity. As shown in Section 4.1 and 4.2, the additional computational cost comes from the calculation and backpropagation of the $\mathcal{L}_D$ and $\mathcal{L}_M$ losses. Since we perform Monte Carlo sampling to sample $\sqrt{N}$ nodes, the complexity of calculating $\mathcal{L}_D$ becomes $O(K\sqrt{N}d^2)$ and the complexity of backpropagation for it becomes $O(KNd^2)$; the complexity of calculating $\mathcal{L}_M$ and its gradient is $O(KNd_0 d)$. Considering d and K are usually much smaller than the number of nodes N, the total additional time complexity becomes $O(KNd^2 + KNd_0 d)$, which increases linearly with the number of nodes.

D DATASET STATISTICS

The dataset statics is shown in Table 5. For the experiments on Cora/Citeer/Pubmed, we follow the widely used semi-supervised setting in Kipf & Welling (2016); Zhou et al. (2020) with 20 nodes per class for training, 500 nodes for validation and 1000 nodes for test. For CoauthorCS, we follow Zhou et al. (2020) and use 40 nodes per class for training, 150 nodes per class for validation and the rest for test. For other datasets, we follow Pei et al. (2020); Chen et al. (2020b) to randomly split nodes of each class into 60%, 20%, and 20% for training, validation and test.

Datasets	#Nodes	#Edges	#Features	#Classes
Cora	2,708	5,429	1,433	7
Citeseer	3,327	4,732	3,703	6
Pubmed	19,717	44,338	500	3
CoauthorCS	18,333	81894	6805	15
Chameleon	2,277	36,101	2,325	5
Actor	7,600	33,544	931	5
Cornell	183	295	1,703	5
Texas	183	309	1,703	5
Wisconsin	251	499	1,703	5

Table 5: Dataset Statistics.

The datasets are publicly available at:

Cora/Citeseer/Pubmed: https://github.com/tkipf/gcn/tree/master/gcn/data

CoauthorCS: https://github.com/shchur/gnn-benchmark/tree/master/data/npz

Others: https://github.com/graphdml-uiuc-jlu/geom-gcn/tree/master/new_data

E PARAMETER SETTINGS

Experiments for Table 1. For BN, PairNorm and DGN, we reuse the performance reported in Zhou et al. (2020) for GCN and GAT. For ChebyNet, we use their best configuration to run the experiments. For DropEdge, we tune the sampling percent from $\{0.1, 0.3, 0.5, 0.7\}$, weight decay from $\{0, 5e\text{-}4\}$, dropout rate from $\{0, 0.6\}$ and fix the learning rate to be 0.005. For the proposed DeCorr, following Zhou et al. (2020) we use $5e$-4 weight decay on Cora, $5e$-5 weight decay on Citeseer and

CoauthorCS, $1e$-3 weight decay on Pubmed. We further search α from $\{0.1, 1\}$, β from $\{1, 10\}$, learning rate from $\{0.005, 0.01, 0.02\}$ and dropout rate from $\{0, 0.6\}$.

Experiments in Table 3. For all methods, we tune the learning rate from $\{0.005, 0.01, 0.02\}$, dropout rate from $\{0, 0.6\}$. For DeCorr, we tune α from $\{0.1, 1\}$, β from $\{1, 10\}$ while for DGN+ DeCorr we tune α from $\{0.05, 0.1, 0.5\}$ and β from $\{0.1, 1\}$.

Experiments in Table 4. For GCNII and GCNII*, we use their best configuration as reported in Chen et al. (2020b). Based on these configurations, we further equip them with DeCorr and tune α from $\{0.01, 0.05, 0.1\}$ and β from $\{0.1, 1, 10\}$.

Implementation of Baselines. We use the following publicly available implementations of baseline methods and deep models:

(a) DGN: https://github.com/Kaixiong-Zhou/DGN/
(b) PairNorm: https://github.com/Kaixiong-Zhou/DGN/
(c) BatchNorm: https://github.com/Kaixiong-Zhou/DGN/
(d) DropEdge: https://github.com/DropEdge/DropEdge
(e) GCNII: https://github.com/chennnM/GCNII/tree/master/PyG
(f) APPNP: https://github.com/rusty1s/pytorch_geometric