DeGNN: Improving Graph Neural Networks with Graph Decomposition

Xupeng Miao*[1], Nezihe Merve Gürel*[2], Wentao Zhang*[3], Zhichao Han[4], Bo Li[5], Wei Min[6], Susie Xi Rao[7], Hansheng Ren[8], Yinan Shan[9], Yingxia Shao[10], Yujie Wang[11], Fan Wu[12], Hui Xue[13], Yaming Yang[14], Zitao Zhang[15], Yang Zhao[16], Shuai Zhang[17], Yujing Wang[18], Bin Cui[19], Ce Zhang[20]
[1,3,11,19]Department of Computer Science & Key Lab of High Confidence Software Technologies (MOE), Peking University,
[2,7,17,20]ETH Zurich, [4,6,9,15,16]eBay China, [5,12]UIUC, [8]UCAS China, [10]BUPT, [1,3]Tencent Inc., [13,14,18]MSRA
[1,3,11,18,19]{xupeng.miao, wentao_zhang, alfredwang, yujwang, bin.cui}@pku.edu.cn
[2,7,17,20]{nezihe.guerel, raox, shuazhang, ce.zhang}@inf.ethz.ch [4,6,9,15,16]{zhihan, wmin, yshan, zitzhang, yzhao5}@ebay.com
[8]renhansheng16@mails.ucas.edu.cn [5,12]{fanw6, lbo}@illinois.edu [10]shaoyx@bupt.edu.cn

ABSTRACT

Mining from graph-structured data is an integral component of graph data management. A recent trending technique, graph convolutional network (GCN), has gained momentum in the graph mining field, and plays an essential part in numerous graph-related tasks. Although the emerging GCN optimization techniques bring improvements to specific scenarios, they perform diversely in different applications and introduce many trial-and-error costs for practitioners. Moreover, existing GCN models often suffer from oversmoothing problem. Besides, the entanglement of various graph patterns could lead to non-robustness and harm the final performance of GCNs. In this work, we propose a simple yet efficient graph decomposition approach to improve the performance of general graph neural networks. We first empirically study existing graph decomposition methods and propose an automatic connectivity-ware graph decomposition algorithm, DeGNN. To provide a theoretical explanation, we then characterize GCN from the information-theoretic perspective and show that under certain conditions, the mutual information between the output after l layers and the input of GCN converges to 0 exponentially with respect to l. On the other hand, we show that graph decomposition can potentially weaken the condition of such convergence rate, alleviating the information loss when GCN becomes deeper. Extensive experiments on various academic benchmarks and real-world production datasets demonstrate that graph decomposition generally boosts the performance of GNN models. Moreover, our proposed solution DeGNN achieves state-of-the-art performances on almost all these tasks.

CCS CONCEPTS

• **Computing methodologies → Knowledge representation and reasoning**; • **Mathematics of computing → Information theory**; • **Information systems → Data mining**.

* Equal contribution.

KEYWORDS

Graph neural network, Graph decomposition, Information loss

ACM Reference Format:
Xupeng Miao, Nezihe Merve Gürel, Wentao Zhang, Zhichao Han, Bo Li, Wei Min, Susie Xi Rao, Hansheng Ren, Yinan Shan, Yingxia Shao, Yujie Wang, Fan Wu, Hui Xue, Yaming Yang, Zitao Zhang, Yang Zhao, Shuai Zhang, Yujing Wang, Bin Cui, Ce Zhang. 2021. DeGNN: Improving Graph Neural Networks with Graph Decomposition. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD ’21), August 14–18, 2021, Virtual Event, Singapore.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3447548.3467312

1 INTRODUCTION

Graph data is ubiquitous in various real-world applications, ranging from social networks to business transaction data [4] as well as bibliographic data [7]. Extensive efforts have been put into the filed of storage and access of graph data and numerous graph databases [13] and query techniques [8, 12] have been proposed. Yet, effectively utilizing graph data, i.e., node classification [37, 42, 44] or link predictions [3], is also an integral component of graph data management and remains to be a challenging topic. Given its emerging popularity, improving and understanding graph neural networks (GNNs) is important, for both future graph data management system and practitioners in general. However, despite its popularity, one pressing limitation is that the performances of GCNs on practical applications are usually not satisfactory enough “out of the box.” As a result, practitioners are often left with a range of GCN variants — to name a few JK-Net [40], ResGCN [15], DenseGCN [19], GPNN [20], NGCN [2], DGCN [46], DropEdge [29], LGCN [9], GMI [27], and GAT [34] — to try in practice. This vast diversity poses challenges on both the system and practitioners. In this paper, we ask: *Can we design a single family of GCN models that matches, or outperforms, the best of these state-of-the-art models?*

In this paper, we start with the standard textbook GCN model and try to identify its fundamental limitations. First, *overmoothing* is a notorious problem in deep graph convolutional networks [19, 25]. DropEdge [29] adopts the edge sampling technique to involve regularization for better performance. However, to the best of our knowledge, no work so far has systematically studied the problem that which technique significantly outperforms all the other methods. Second, a large graph usually contains entanglement of different graph patterns and semantics, which aggravate the oversmoothing

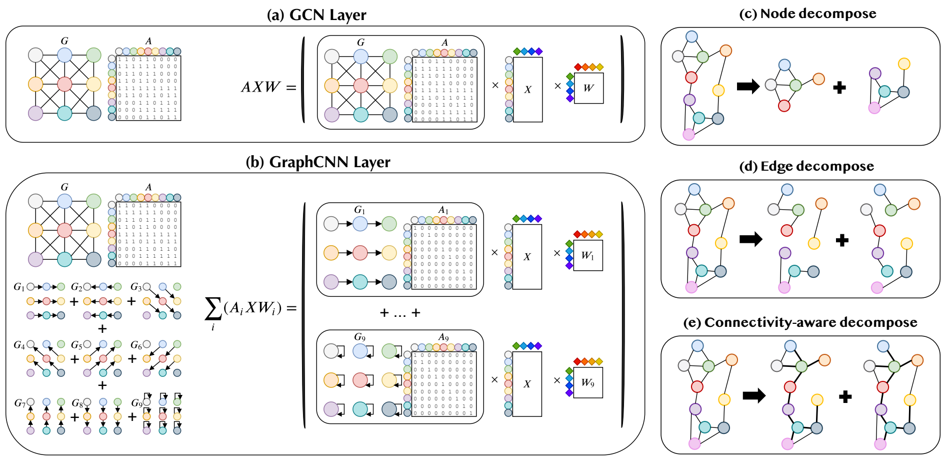

Figure 1: (a) and (b) are illustrations of one layer in GCN and one layer under one decomposition strategy in GraphCNN. A is the adjacency matrix, X is the input, and W (W_i) are learnable weights. In GraphCNN, $A = \sum_i A_i$ and $A_i \cap A_j = \emptyset$ for $i \neq j$. In our experiments and analysis, we follow the original normalized A in GCN [15]. (c) (d) (e) are illustrations of different decomposition methods introduced in Sec. 2.

issue and make the model sensitive to noises [23, 26]. Moreover, the choices and evaluations introduce additional complexity for practical applications. Large amounts of these improvements over GCN are sometimes confusing and tedious, especially from a industrial perspective.

In this paper, we aim to propose a systematic approach with theoretical guidance to address the aforementioned limitations. A recent empirical finding shows that a simple graph transformation by partitioning the graph data with a hand-picked structure can usually help boost the performance of GCNs. For example, thinking of an image as a graph, if we decompose it into multiple subgraphs (as illustrated in Figure 1), it is possible to design a GCN-variant to implement a standard CNN-like model, which obviously benefits from going deeper. GraphCNN [32] is one such example of taking advantage of graph decomposition. However, this requires us to know the "*right*" decomposition of a graph, which is often not available in practice.

Inspired by these observations and results, we ask two questions:

(1) From the *empirical* perspective, can we *automatically* decompose a graph and improve the quality of state-of-the-art Graph Neural Networks?
(2) From the *theoretical* perspective, can we explain the significant impact of graph decomposition on the performance of Graph Neural Networks?

Our first contribution is a novel graph connectivity aware decomposition algorithm to automatically decompose a graph into multiple subgraphs and use them to improve the quality of Graph Neural Networks. Unlike existing graph decomposition algorithms, which directly distribute the nodes or edges into different partitions, our propose DeGNN algorithm preserves the graph connectivity by a simple yet effective structure in the graph — the spanning trees. We first generate the spanning forest as the skeletons, and then decompose the residual graph into different subgraphs. With the replication of these skeletons, the decomposition will not bring any additional isolated structure so that the information propagation process is not blocked.

Our second contribution is to take the first step towards the theoretical analysis on the impact of graph decomposition. We take an information theoretical view and analyze the infinite-sample behaviour of Shannon's *mutual information* between the output after l layers and the input, $\mathcal{I}(\mathbf{x}; \mathbf{y}^{(l)})$. When $\mathcal{I}(\mathbf{x}; \mathbf{y}^{(l)}) = \mathcal{H}(\mathbf{x})$, it indicates that all information in the input are *fully preserved* after l layers; whereas when $\mathcal{I}(\mathbf{x}; \mathbf{y}^{(l)}) = 0$, it indicates that *all information are lost.* We show that:

(1) (Theorem 1, 2) Under certain conditions (on the singular value of the graph), mutual information $\mathcal{I}(\mathbf{x}; \mathbf{y}^{(l)})$ for GCN converges to 0 exponentially fast with respect to the depth l, corresponding to the oversmoothing problem of GCN in practice;
(2) (Theorem 3, 4) Only under a *much weaker* condition, the mutual information $\mathcal{I}(\mathbf{x}; \mathbf{y}^{(l)})$ of DeGNN with decomposition converges to 0.

The theoretical analysis is non-trivial — in a concurrent work [25], the authors conducted engaged analysis, from dynamic system perspective, and lead to a similar result for GCN (Theorem 1, 2). Our information theoretical perspective not only provides a much simpler, but equally tight analysis for GCN, but more importantly, our analysis makes it possible to analyze more complex cases for DeGNN with the presence of decomposition (Theorem 3, 4).

We systematically benchmark DeGNN, on 11 public benchmark datasets and three real-world datasets from our industry partners, in both transitive and inductive setting, and comparing its performance with more than twenty recent GNN models [2, 5, 6, 9, 11, 15, 16, 19, 20, 22, 27, 29, 34–36, 39–41, 46] (including GCN, JK-Net, ResGCN, DenseGCN, GPNN, NGCN, DGCN, STGCN, DGI, GMI, GAT, LGCN, APPNP, GIN, SGC, DropEdge, GraphSAGE, ClusterGCN, FastGCN, GraphSAINT). We show that with our graph decomposition method, simpler models such as DenseGCN can often outperform the best among these state-of-the-art models on 14 datasets. DeGNN achieves at least comparable, and often outperforms, the *best* of these models across all tasks and datasets, providing a strong method to use in future graph data management

Table 1: Test accuracy (in %) on citation datasets. We use bold font for methods with the highest average accuracy.

Models	K	Cora	Citeseer	Pubmed
GCN	1	**81.8±0.5**	**70.8±0.5**	**79.3±0.7**
Node decomposed GCN	2	81.4±0.3	70.3±0.4	78.8±0.4
	3	80.1±0.4	68.7±0.5	78.1±0.5
	4	79.4±0.6	67.1±0.9	77.2±0.7
Edge decomposed GCN	2	81.3±0.3	70.2±0.4	78.4±0.3
	3	80.7±0.5	69.7±0.4	78.3±0.3
	4	79.8±0.8	68.4±0.7	77.9±0.4

systems to serve diverse applications and users. In addition, graph decomposition largely moderates the oversmoothing problem in an empirical study, which confirms our motivation and theoretical conclusion in this paper.

2 METHODOLOGY

2.1 Notations

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be an undirected graph with a vertex set $v_i \in \mathcal{V}$ and edge set $e_{i,j} \in \mathcal{E}$. We refer to v_i as a node, and $\mathbf{x}_i \in \mathbb{R}^d$ associated with v_i as its features. We denote the node feature attributes by $\mathbf{X} \in \mathbb{R}^{n\times d}$ whose rows are given by $\mathbf{x}_i$. The adjacency matrix $\mathbf{A}$ (weighted or binary) is derived as an $n \times n$ matrix with $(\mathbf{A})_{i,j} = e_{i,j}$ if $e_{i,j} \in \mathcal{E}$, and $(\mathbf{A})_{i,j} = 0$ elsewhere.

Common GCN. We define the following operator $f : \mathbb{R}^n \to \mathbb{R}^n$ that is composed of (1) a linear function parameterized by the adjacency matrix $\mathbf{A}$ and a weight matrix $\mathbf{W}^{(i+1)}$ at layer $i + 1$, and (2) an activation function. Given the input matrix $\mathbf{X}$, let $\mathbf{Y}^{(0)} = \mathbf{X}$. Each GNN layer maps it to an output vector of the same shape:

$$\mathbf{Y}^{(i+1)} = f_{\mathbf{A},\mathbf{W}^{(i+1)}}(\mathbf{Y}^{(i)}) = \sigma(\mathbf{A}\mathbf{Y}^{(i)}\mathbf{W}^{(i+1)}). \tag{1}$$

Graph Decomposed GCN. In GraphCNN [32], the adjacency matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ is decomposed into K additive $n\times n$ matrices such that $\mathbf{A} = \sum_{k=1}^{K} \mathbf{A}_k$. The layer-wise propagation rule becomes:

$$\mathbf{Y}^{(i+1)} = g_{\mathbf{A}_k,\mathbf{W}_k^{(i+1)}}(\mathbf{Y}^{(i)}) = \sigma\left(\sum_{k=1}^{K} \mathbf{A}_k\mathbf{Y}^{(i)}\mathbf{W}_k^{(i+1)}\right). \tag{2}$$

Unlike the image pixels graph decomposition in GraphCNN, it is non-trivial to perform decomposition on arbitrary graph-structured data. Clearly, there is no absolute geometric space and direction concept in most real-world graphs, and the *spatial anisotropy* [17] makes decomposing an image with the predefined coordinates and directions much easier than a graph. As such, we first explore existing graph decomposition methods in the following section.

2.2 Preliminary study

To verify the effectiveness of graph decomposed GCN, we select two representative graph decomposition methods, including node decomposition in Figure 1 (c) (i.e., METIS [14]) and edge decomposition in Figure 1 (d) (i.e., Greedy [10]), and empirically evaluate their performance. We compare the original GCN ($K = 1$) with the decomposed GCN (E.q. (2)) on the citation datasets [15] (experimental settings are consistent with Section 4.1). We build a two-layer GCN and decompose its matrix $\boldsymbol{A}$ into K pieces and rebuild every layer with K weight matrices. As shown in Table 1, the edge decomposed GCN performs better than node decomposed GCN under different K values, but none of them outperforms the original GCN. The empirical study implies that the typical decomposition methodology can not bring any performance improvement and even make it worse. It is also observed that increasing the number of decomposition components speeds up the convergence on the training set in our experiments but leads to lower testing accuracy.

Analysis and graph connectivity. Two possible explanations for the above phenomenon are: (1) More weight matrices brought by decomposition may cause overfitting; (2) Traditional decomposition may break the graph connectivity and impede the spread of information as GCN relies on the graph structures to propagate the node features and labels along the edges. For example, based on E.q. (2), the edge-cut algorithms decompose the adjacency matrix $\boldsymbol{A}$ into K matrices. Each $\boldsymbol{A}_k$ only has edges inside the partition, and those vertices outside the partition are isolated from each other. The vertex-cut algorithms can also lead to isolation for low-degree vertices due to skew power-law degree distributions of large practical graphs. The partitioned graph may lead to the result that the nodes can be trapped in a smaller region and cannot spread to distant reachable nodes in the original graph. These methods result in isolated vertices without any edges in graph partitions, and they have no chance to aggregate information from its neighborhoods. The hidden representation of these isolated vertices is only determined by its own input features, which lacks the graph structure information. Therefore, an inappropriate decomposition may reduce the graph connectivity and affect the information propagation among nodes, thus decreasing the model learning ability. How to maintain the graph connectivity remains a challenge.

Algorithm 1 Connectivity-aware graph decomposition

```
Input: The graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, $N = |\mathcal{V}|$.
Parameter: The number of partitions $p$ for METIS, the number of decomposed graphs $K$.
Output: The decomposed graph $(\mathcal{G}_1, \mathcal{G}_2, ..., \mathcal{G}_K)$.
 1: Partition the graph $\mathcal{G}$ into $p$ subgraphs with METIS.
 2: Merge the subgraphs into $\mathcal{G}_m$.
 3: $T \longleftarrow$ Generate a random spanning forest on $\mathcal{G}_m$.
 4: $\forall i \in [1, K], R_i \longleftarrow (\mathcal{V}, \varnothing)$.
 5: $k \longleftarrow 0$.
 6: for $i = 1$ to $N$ do
 7:     for $v_j \in$ Neighbor($v_i$) on the residual graph $\mathcal{G}/T$ do
 8:         Assign edge $(v_i, v_j)$ to $R_{k+1}$
 9:         $k \longleftarrow (k + 1)\%K$
10: return $(R_1 \cup T, R_2 \cup T, ..., R_K \cup T)$
```

2.3 DeGNN: Connectivity-aware graph decomposition

To overcome the defects of existing graph decomposition methods, we propose DeGNN to take the graph connectivity into account and automatically perform graph decomposition on general graph-structured data. We clarify that the graph connectivity requirement as follows: each pair of reachable vertices in the original graph are still reachable in the decomposed graph. The graph connectivity preserving property further leads to: $\boldsymbol{A}_k^l \approx \boldsymbol{A}^l$, when the network architecture is deep enough (i.e., $l \longrightarrow \infty$).

Considering the graph connectivity preserving principle, we propose to utilize a simple and effective structure in the graph — the spanning trees. As shown in Algorithm 1, unlike the previous decomposition methods, we first generate the spanning forest (line 3) of the graph, then the replicas of the generated graph skeleton T (i.e., bold edges in Figure 1 (e)) will be distributed to the decomposed graphs (line 10). In this way, the node connectivity is still completely preserved after decomposition with the presence of T in each sub-graph. Furthermore, we utilize METIS to eliminate some edge cuts before generating the skeletons to control the graph connectivity of the generated spanning tree structures (lines 1-2). The hyperparameter p in METIS controls the amounts of edge cuts and further leads to different node connectivity. When $p = 1$, the node connectivity can be completely preserved, but the over-smoothing issue is dominate. As supported by the theory (see Section 3), graph decomposition alleviates the over-smoothing problem but requires some edge cuts. Therefore, an appreciate value of p is desired to balance the impact of connectivity and over-smoothing. In our experiments, we find p generally works well around 100-300, and a hyper-parameter analysis will be provided later. Finally, we decompose the residual graph (lines 4-10). Specifically, for each node, the adjacent nodes and the associated edges are distributed to each sub-graph in a round-robin manner (lines 7-9). The advantage is that it will not generate any additional isolated vertices or independent sub-graphs, which facilitates the information propagation process.

3 THEORETICAL ANALYSIS

In this section, we analyze both GCN and DeGNN from the information theoretical perspective to provide a theoretical explanation on performing graph decomposition technique on GCN. We denote the jth singular value of a matrix by $\lambda_j(\cdot)$. We further denote the vectorized input $\mathbf{X}$ and output after the lth layer $\mathbf{Y}^{(l)}$ by $\mathbf{x}$ and $\mathbf{y}^{(l)}$, respectively. For n-dimensional real random vectors $\mathbf{x}$ and $\mathbf{y}$ defined over finite alphabets $\mathcal{X}^n$ and Ω^n, we denote entropy of $\mathbf{x}$ by $\mathcal{H}(\mathbf{x})$, and mutual information between $\mathbf{x}$ and $\mathbf{y}$ by $\mathcal{I}(\mathbf{x};\mathbf{y})$. In the following analysis, we focus on two measures to investigate the effect of decomposition, that is, *information preservation* $\mathcal{I}(\mathbf{x};\mathbf{y}^{(l)})$ and the *information loss* $\mathcal{L}(\mathbf{y}^{(l)}) = \mathcal{H}(\mathbf{x}|\mathbf{y}^{(l)})$ (relative entropy of $\mathbf{x}$ with respect to $\mathbf{y}^{(l)}$). We measure the information decay in GNNs at different output layers l: lower information loss or larger information preservation indicates more meaningful learned features for GNNs in the infinite-sample regime.

3.1 Information loss in GCN

In this section, our goal is to investigate the regimes where GCN (1) does not benefit from going deeper, or (2) is guaranteed to preserve all information at its output. We aim to understand this by analyzing the behavior of mutual information between the input and the output of certain network layers at different depths. Due to the space limitation, we relegate the proof details to the Appendix A and retains the necessary sketch.

First, we formulate the relationship between input and output layers incorporating the non-linear activation functions. In this paper, we focus on the most popular choices, i.e., ReLU, and leave the study of other functions to future work. The characteristics of the GCN layer-wise propagation rule leads to the following result:

Lemma 1. *Let $\otimes$ denote the Kronecker product. For GCNs with parametric ReLU activations $\sigma : x \to \max(x, ax)$ with $a \in (0, 1)$, we define $\mathbf{P}^{(i+1)}$ as a diagonal mask matrix whose nonzero entries are in $\{a, 1\}$ such that $(\mathbf{P}^{(i+1)})_{j,j} = 1$ if $\big((\mathbf{W}^{(i+1)} \otimes \mathbf{A})\mathbf{y}^{(i)}\big)_j \geq 0$, and $(\mathbf{P}^{(i+1)})_{j,j} = a$ elsewhere. $\mathbf{y}^{(l)}$ can be written as*

$$\mathbf{y}^{(l)} = \mathbf{P}^{(l)}(\mathbf{W}^{(l)} \otimes \mathbf{A}) \cdots \mathbf{P}^{(2)}(\mathbf{W}^{(2)} \otimes \mathbf{A})\mathbf{P}^{(1)}(\mathbf{W}^{(1)} \otimes \mathbf{A})\mathbf{x}.$$

Following our earlier discussion, we will now state our first result which characterizes the regime in which the information propagated across the GCN layers exponentially decays to 0.

Theorem 1. *Suppose $\sigma_{\mathbf{A}} = \max_j \lambda_j(\mathbf{A})$ and $\sigma_{\mathbf{W}} = \sup_{i\in\mathbb{N}^+} \max_j \lambda_j(\mathbf{W}^{(i)})$. If $\sigma_{\mathbf{A}}\sigma_{\mathbf{W}} < 1$, then $\mathcal{I}(\mathbf{x};\mathbf{y}^{(l)}) = O\big((\sigma_{\mathbf{A}}\sigma_{\mathbf{W}})^l\big)$, and hence $\lim_{l\to\infty} \mathcal{I}(\mathbf{x};\mathbf{y}^{(l)}) = 0$.*

This shows that under certain conditions the information after l GCN layers with (parametric) ReLUs asymptotically converges to 0 exponentially fast. Interestingly, there are also regimes in which GCN will perfectly preserve the information, stated as follows:

Theorem 2. *Following Theorem 1, let $\gamma_{\mathbf{A}} = \min_j \lambda_j(\mathbf{A})$ and $\gamma_{\mathbf{W}} = \inf_{i\in\mathbb{N}^+} \min_j \lambda_j(\mathbf{W}^{(i)})$. If $a\gamma_{\mathbf{A}}\gamma_{\mathbf{W}} \geq 1$, then $\forall l \in \mathbb{N}^+$ the information loss $\mathcal{L}(\mathbf{y}^{(l)}) = 0$.*

Effect of Normalized Laplacian: The results obtained above holds for any adjacency matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$. The unnormalized $\mathbf{A}$, however, comes with a major drawback as changing the scaling of feature vectors. To overcome this problem, $\mathbf{A}$ is often normalized such that its rows sum to one. We then adopt our results to GCN with normalized Laplacian whose largest singular value is one, and obtain the following results.

Corollary 1. *Let $\mathbf{D}$ denote the degree matrix such that $(\mathbf{D})_{j,j} = \sum_m (\mathbf{A})_{j,m}$, and $\mathbf{L}$ be the associated normalized Laplacian $\mathbf{L} = \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$. Suppose GCN uses the following mapping $\mathbf{Y}^{(i+1)} = \sigma(\mathbf{L}\mathbf{Y}^{(i)}\mathbf{W}^{(i)})$ and $\sigma_{\mathbf{W}} = \sup_i \max_j \lambda_j(\mathbf{W}^{(i+1)})$. If $\sigma_{\mathbf{W}} < 1$, then $\mathcal{I}(\mathbf{x};\mathbf{y}^{(l)}) = O(\sigma_{\mathbf{W}}^l)$, and hence $\lim_{l\to\infty} \mathcal{I}(\mathbf{x};\mathbf{y}^{(l)}) = 0$.*

This indicates that with the standard normalized adjacency matrix, the mutual information between the input and the output of lth layer of GCN will decay to 0 exponentially fast.

3.2 Information loss in DeGNN

Motivated by the graph decomposition strategy adopted by several work including GraphCNN, in this section we aim to analyze the information loss after graph decomposition, and understand whether the information can be preserved by aggregating local sub-graphs. In particular, we take the DeGNN(GCN) as as an example which sums the decomposed graphs together as the adjacency matrix to perform the analysis.

Similarly as in Lemma 1, $\mathbf{y}^{(l)}$ can be reduced to $\mathbf{y}^{(l)} = \mathbf{P}^{(l)} \sum_{k_l=1}^K (\mathbf{W}_{k_l}^{(l)} \otimes \mathbf{A}_{k_l}) \cdots (\mathbf{W}_{k_2}^{(2)} \otimes \mathbf{A}_{k_2})(\mathbf{W}_{k_1}^{(1)} \otimes \mathbf{A}_{k_1})\mathbf{x}$ for a diagonal mask matrix $\mathbf{P}^{(i+1)}$ such that $(\mathbf{P}^{(i+1)})_{j,j} = 1$ if $\sum_{k_{i+1}=1}^K (\mathbf{W}_{k_{i+1}}^{(i+1)} \otimes \mathbf{A}_{k_{i+1}})\mathbf{y}^{(i)} \geq 0$, and $(\mathbf{P}^{(i+1)})_{j,j} = a$ otherwise. Following a similar proof for GCN, we obtain the following result for DeGNN:

Theorem 3. *Let $\sigma^{(i)}$ denotes the maximum singular value of $\mathbf{P}^{(i)} \sum_{k_i=1}^K (\mathbf{W}_{k_i}^{(i)} \otimes \mathbf{A}_{k_i})$ such that $\sigma^{(i)} = \max_j \lambda_j\big(\mathbf{P}^{(i)} \sum_{k_i} (\mathbf{W}_{k_i}^{(i)} \otimes$*

$\mathbf{A}_{k_i})\big)$. *If* $\sup_{i\in\mathbb{N}^+}\sigma^{(i)} < 1$, *then* $\mathcal{I}(\mathbf{x};\mathbf{y}^{(l)}) = O\big((\sup_{i\in\mathbb{N}^+}\sigma^{(i)})^l\big)$, *and hence* $\lim_{l\to\infty}\mathcal{I}(\mathbf{x};\mathbf{y}^{(l)}) = 0$.

Theorem 3 describes the condition on the layer-wise weight matrices $\mathbf{W}_k$ where DeGNN fails in capturing the feature characteristics at its output in the asymptotic regime. We then state the second result for DeGNN which ensures the information loss $\mathcal{L}(\mathbf{y}^{(l)}) = 0$ as follows.

Theorem 4. *Consider the propagation rule of DeGNN. Let* $\gamma^{(i)}$ *denotes the minimum singular value of* $\mathbf{P}^{(i)}\sum_{k_i=1}^{K}(\mathbf{W}_{k_i}^{(i)}\otimes\mathbf{A}_{k_i})$ *such that* $\gamma^{(i)} = \min_j \lambda_j\big(\mathbf{P}^{(i)}\sum_{k_i=1}^{K}(\mathbf{W}_{k_i}^{(i)}\otimes\mathbf{A}_{k_i})\big)$. *If* $\inf_i \gamma^{(i)} \geq 1$, *then* $\forall l\in\mathbb{N}^+$ *we have* $\mathcal{L}(\mathbf{y}^{(l)}) = 0$.

Proof Sketch. Following Lemma 1, the key step in proving above theorems is as follows. Consider the singular value decomposition $\mathbf{U}\Lambda\mathbf{V}^T = \mathbf{P}^{(l)}(\mathbf{W}^{(l)}\otimes\mathbf{A})\ldots\mathbf{P}^{(2)}(\mathbf{W}^{(2)}\otimes\mathbf{A})\mathbf{P}^{(1)}(\mathbf{W}^{(1)}\otimes\mathbf{A})$ such that $(\Lambda)_{j,j} = \lambda_j(\mathbf{P}^{(l)}(\mathbf{W}^{(l)}\otimes\mathbf{A})\ldots\mathbf{P}^{(2)}(\mathbf{W}^{(2)}\otimes\mathbf{A})\mathbf{P}^{(1)}(\mathbf{W}^{(1)}\otimes\mathbf{A}))$, and let $\tilde{\mathbf{x}} = \mathbf{V}^T\mathbf{x}$. We have

$$\mathcal{I}(\mathbf{x};\mathbf{y}^{(l)}) \overset{(1)}{=} \mathcal{I}(\tilde{\mathbf{x}};\Lambda\tilde{\mathbf{x}}) \overset{(2)}{\leq} \mathcal{H}(\tilde{\mathbf{x}}) \overset{(3)}{=} \mathcal{H}(\mathbf{x}) \tag{3}$$

where (1, 3) results from that $\mathbf{U}$ and $\mathbf{V}$ are invertible, and equality holds in (2) *iff* Λ is invertible, i.e., singular values of $\mathbf{P}^{(l)}(\mathbf{W}^{(l)}\otimes\mathbf{A})\ldots\mathbf{P}^{(2)}(\mathbf{W}^{(2)}\otimes\mathbf{A})\mathbf{P}^{(1)}(\mathbf{W}^{(1)}\otimes\mathbf{A})$ are nonzero. Theorems 1, 2, 3 and 4 can be inferred from E.q. 3. That is, $\mathcal{I}(\mathbf{x};\mathbf{y}^{(l)}) = 0$ *iff* $\max_j(\Lambda^l)_{j,j} = 0$ in the asymptotic regime. Similarly, *iff* $\min_j(\Lambda^l)_{j,j} > 0$, $\mathcal{I}(\mathbf{x};\mathbf{y}^{(l)})$ is maximized and given by $\mathcal{H}(\mathbf{x})$, hence $\mathcal{L}(\mathbf{y}^{(l)}) = 0$.

In order to understand the role of decomposition in DeGNN, we revisit the conditions on full information loss ($\mathcal{I}(\mathbf{x};\mathbf{y}^{(l)}) = 0$) and full information preservation ($\mathcal{L}(\mathbf{y}^{(l)}) = 0$) for a specific choice of decomposition, which will be used to demonstrate the information processing capability.

Corollary 2. *Suppose the singular value decomposition of* $\mathbf{A}$ *is given by* $\mathbf{A} = \mathbf{U}_\mathbf{A}\mathbf{S}\mathbf{V}_\mathbf{A}^T$, *and each* $\mathbf{A}_k$ *is set to* $\mathbf{A}_k = \mathbf{U}_\mathbf{A}\mathbf{S}_k\mathbf{V}_\mathbf{A}^T$ *where* $(\mathbf{S}_k)_{m,m} = \lambda_m(\mathbf{A})$ *if* $k = m$ *and* $(\mathbf{S}_k)_{m,m} = 0$ *elsewhere. We then have the following results: For* $\sigma_{\mathbf{A}_k} = \lambda_k(\mathbf{A})$ *and* $\sigma_{\mathbf{W}_k} = \sup_{i\in\mathbb{N}^+}\max_j\lambda_j(\mathbf{W}_k^{(i)})$, *i.e., if* $\sigma_{\mathbf{A}_k}\sigma_{\mathbf{W}_k} < 1\ \underline{\forall k = \{1,2,\ldots,n\}}$, *then* $\lim_{l\to\infty}\mathcal{I}(\mathbf{x};\mathbf{y}^{(l)}) = 0$.

Corollary 3. *Let* $\gamma_{\mathbf{W}_k} = \inf_{i\in\mathbb{N}^+}\min_j\lambda_j(\mathbf{W}_k^{(i)})$. *If* $a\sigma_{\mathbf{A}_k}\gamma_{\mathbf{W}_k} \geq 1$, $\underline{\forall k \in \{1,2,\ldots,n\}}$, *then* $\mathcal{L}(\mathbf{y}^{(l)}) = 0\ \forall l\in\mathbb{N}^+$.

Discussion: Impact of Decomposition. Consider the setting where $\mathbf{A}$ is fixed for both GCN and DeGNN. The discussion below will revolve around the regime of singular values in layer-wise weight matrices, $\mathbf{W}_{\text{GCN}}^{(i)}$ and $\mathbf{W}_{\text{DeGNN}}^{(i)}$ where the information loss $\mathcal{L}(\mathbf{y}^{(l)}) = 0$ for specific decomposition strategy used in Corollary 3.

Recall from Theorem 2 and Corollary 3 that while GCN requires singular values of all weight matrices $\mathbf{W}_{\text{GCN}}^{(i)}$ to compensate for the minimum singular value of $\mathbf{A}$ such that $\min_j\lambda_j(\mathbf{W}_{\text{GCN}}^{(i)}) \geq \frac{1}{a\min_k\lambda_k(\mathbf{A})}$ to ensure $\mathcal{L}(\mathbf{y}^{(l)}) = 0$, DeGNN relaxes this condition by introducing a milder constraint. That is, the singular values of its weight matrices $\mathbf{W}_{\text{k, DeGNN}}^{(i)}$ need to compensate only for the singular value of their respective component $\mathbf{A}_k$, meaning $\min_j\lambda_j(\mathbf{W}_{\text{k, DeGNN}}^{(i)}) \geq \frac{1}{a\lambda_k(\mathbf{A})}$ implies $\mathcal{L}(\mathbf{y}^{(l)}) = 0$.

Table 2: Dataset Statistics

Dataset	#Nodes	#Features	#Edges	#Classes	#Train/Val/Test
Cora	2,708	1,433	5,429	7	140/500/1,000
Citeseer	3,327	3,703	4,732	6	120/500/1,000
Pubmed	19,717	500	44,338	3	60/500/1,000
Amazon Computer	13,381	767	245,778	10	200/300/12,881
Amazon Photo	7,487	745	119,043	8	160/240/7,087
Coauthor CS	18,333	6,805	81,894	15	300/450/17,583
Coauthor Physics	34,493	8,415	247,962	5	100/150/34,243
Actor	7,600	931	33,544	5	3,648/608/760
Chameleon	2,277	2,325	36,101	4	1,093/182/228
eBay Small	96,532	114	1,013,936	2	709,755/101,393/202,788
eBay Large	126,327	480	5,001,222	2	3,500,855/500,122/1,000,245
Tencent	100,000	64	841,341	253	5,000/10,000/30,000
Cora (Inductive)	2,708	1,433	5,429	7	1,208/500/1,000
Flickr	89,250	500	899,756	7	44,625/22,312/22,312
Reddit	232,965	602	11,606,919	41	155,310/23,297/54,358

The decomposition makes deep GCN training easier by permitting a much larger regime of model weights where the information is still preserved. In other words, under the same weight characteristics (singular values of layer-wise weight matrices), the decomposed GCN will be able to preserve more information of the node features than the vanilla GCN when going deeper. So far, we theoretically justify the potential of graph decomposition in the infinite-sample regime. For the analysis in the finite-sample regime, one could possibly utilize the theory of information bottleneck [30, 31], we leave this as future work.

4 EXPERIMENTS

In this section, we evaluate the effectiveness of our proposed automatic graph decomposition algorithm — DeGNN, on the semi-supervised node classification task. In our experiments, we apply the proposed graph decomposition method to a family of GCN models (e.g., GCN, JK-Net, ResGCN, DenseGCN) for comparison, and it is also viable to apply it to other graph neural networks.

4.1 Experiments setup

4.1.1 Datasets. We conduct experiments on widely used benchmark datasets to validate the effectiveness of our method in both transductive and inductive settings. An overview summary of statistics of the datasets is given in Table 2.

Transductive. Cora, Citeseer, and Pubmed are three well-known citation network datasets, and we follows the same training / validation / test split as [15]. We also evaluate DeGNN on five benchmark graph datasets, including Amazon Computers, Amazon Photo, Coauthor CS, Coauthor Physics [43], Actor and Chameleon [41].

Inductive. Reddit is a social network dataset modeling the community structure of Reddit posts. This dataset is often used as an inductive training setting and the training/validation/test split is coherent with that of GraphSAGE [11]. Flickr originates from NUS-wide and contains different types of images based on the descriptions and common properties of online images. We use a public version of Reddit and Flickr provided by GraphSAINT [41].

Production. The eBay dataset is a real-world transaction graph which we used for fraud transactions detection. Historical transaction records spanning a given period of time were extracted for graph construction. We treat each transaction as a node and assume there is an edge between two nodes if they have the same hard linkage, such as purchasing by the same buyer, shipping to the

Table 3: Test accuracy (in %) on the transductive benchmark datasets. * indicates that we ran our own implementation. We use bold font for methods with the highest average accuracy.

Models	Cora	Citeseer	Pubmed
GPNN	81.8	69.7	79.3
NGCN	83.0	72.2	79.5
DGCN	83.5	72.6	80
DropEdge	82.8	72.3	79.6
STGCN	83.6	72.6	79.5
DGI	82.3±0.6	71.8±0.7	76.8±0.6
GMI	82.7±0.2	73.0±0.3	**80.1±0.2**
GAT	83.0±0.7	72.5±0.7	79.0±0.3
LGCN	83.3±0.5	73.0±0.6	79.5±0.2
APPNP	83.3±0.5	71.8±0.5	**80.1±0.2**
GIN	77.6±1.1	66.1±0.9	77.0±1.2
SGC	81.0±0.0	71.9±0.1	78.9±0.0
GCN*	81.8±0.5	70.8±0.5	79.3±0.7
JK-Net*	81.8±0.5	70.7±0.7	78.8±0.7
ResGCN*	82.2±0.6	70.8±0.7	78.3±0.6
DenseGCN*	82.1±0.5	70.9±0.8	79.1±0.9
DeGNN(GCN)*	83.7±0.4	72.5±0.3	79.8±0.6
DeGNN(JK)*	84.1±0.3	**73.1±0.5**	80.0±0.4
DeGNN(Res)*	83.9±0.5	72.6±0.4	79.9±0.5
DeGNN(Dense)*	**84.3±0.3**	72.7±0.5	**80.1±0.7**

same address or using the same financial instruments etc. Node features are constructed from individual risk factors. To reduce graph size and meanwhile preserve graph connectivity, we adopt a graph sampling strategy: firstly, all fraudulent transactions and random sampled normal transactions are selected as seeds; secondly, each seed is expanded to its 3-hop neighbors, at each hop, no more than 32 neighbors are picked. Thirdly, those groups with transaction numbers less than 5 are filtered out. There are two different sizes of transaction graph (eBay small and eBay large) vary in transaction spanning periods and number of individual features.

The Tencent dataset is an short-video recommendation graph, collected from a real-world mobile application from Tencent Inc.. We sampled 100,000 nodes and the correspond history watching records. The generated graph is a bipartite graph including 57,022 short-videos with labels and 42,978 users. The edge between each pair of short-video item and user represents that the user have watched this short-video. Each user has 64 features. We category these short-videos into 253 different pre-defined classes.

4.1.2 Settings. We use PyTorch to implement the models and we train them using Adam optimizer. Besides, we train each model 400 epochs and terminate the training process if the validation accuracy does not improve for 20 consecutive steps. Note that JK-Net has three aggregators, and we choose the concatenation as the final aggregation layer since it performs best in most cases. Every experiment is ran ten times and the mean accuracy is reported. For inductive tasks, the training procedure is on the training set. The validation set and testing set are added into the graph only for the prediction. Therefore, we need not only perform decomposition on the training graph, but also continue to decompose the whole graph for new nodes and edges. The second decomposition reuses the same K decomposition and the trained DeGNN model to make predictions. The hyperparameters (e.g., learning rate, number of hidden units) are selected from grid search. The grid search was performed over the following search space: hidden size ([8, 16, 32, 64, 128, 256, 512]), learning rate ([1e-3, 3e-3, 5e-3, 8e-3, 1e-2]), partition numbers K ([2,3,4,5,6,7,8]), parameter p in METIS ([40,80,100,150,180,200, 250,500,1000]), dropout rate ([0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.85, 0.9]), L_2 regularization strength ([1e-4, 5e-4, 1e-3, 5e-3, 1e-2, 5e-2, 1e-1]).

Table 4: Test accuracy (in %) on the inductive benchmark datasets.

Models	Cora	Reddit	Flickr
GraphSAGE	73.1±1.4	95.4±0.0	50.1±1.3
ClusterGCN	80.7±0.6	96.6±0.0	48.1±0.5
FastGCN	82.7±0.1	93.7±0.0	50.4±0.1
GraphSAINT	81.5±1.2	96.6±0.1	51.1±0.1
GCN*	83.3±1.1	95.7±0.0	49.2±0.3
JK-Net*	84.2±0.8	96.4±0.1	51.9±0.1
ResGCN*	83.7±1.3	96.3±0.1	51.5±0.1
DenseGCN*	83.9±0.9	96.4±0.0	52.1±0.0
DeGNN(GCN)*	85.4±0.7	96.4±0.0	51.5±0.2
DeGNN(JK)*	**86.1±0.6**	96.6±0.0	**52.5±0.0**
DeGNN(Res)*	85.6±0.8	**96.7±0.1**	51.9±0.1
DeGNN(Dense)*	85.7±0.8	96.6±0.0	**52.5±0.0**

4.1.3 Baselines. We compare our method with the representative methods in recent years, including shallow models such as GCN [15], GPNN [20], NGCN [2], DGCN [46], STGCN [22], DGI [35], GMI [27], GAT [34], LGCN [9], APPNP [16], GIN [39], and SGC [36]; and deeper models such as JK-Net [40], ResGCN [15], DenseGCN [19], DropEdge [29]. We also compare our method with the inductive methods such as GraphSAGE [11], ClusterGCN [6], FastGCN [5] and GraphSAINT [41] on two larger graph dataset including Flickr and Reddit. Since we can apply the decomposition techniques to a range of base models, we use DeGNN(GCN), DeGNN(JK), DeGNN(Res), and DeGNN(Dense) to denote the method that applies our decomposition algorithm to vanilla GCN, JK-Net, ResGCN, and DenseGCN. These DeGNN models in our implementation are built on top of standard GCN on both transductive and inductive settings. Most of the other testing accuracy results are directly collected from the corresponding original paper, except for 1) DropEdge on citation datasets reused from [1], 2) GraphSAGE and ClusterGCN on Flickr reused from GraphSAINT [41], 3) GraphSAGE, ClusterGCN, FastGCN and GraphSAINT on Cora (inductive) reused from [21].

4.2 Comparison with state-of-the-art

We compare DeGNN with large amounts of state-of-the-art baselines on 11 public graph-structured datasets for both transductive and inductive settings, and 3 real-world production datasets.

Transductive. Table 3 summarizes the test accuracy of the baselines and our approaches on Cora, Citeseer, and Pubmed. Deep architecture models relying on node embeddings from previous layers, including JK-Net, ResGCN, and DenseGCN, are indeed better than vanilla GCN but are still weaker than some baseline models, such as GMI, GAT and APPNP. Surprisingly, simply adding the decomposition step on GCN can lead to better performance even better than more recent state-of-the-art models. DeGNN(GCN) achieves significantly better performance over the original GCN by a margin of 2.5%, 1.9%, and 0.8% on the three citation datasets respectively. Moreover, with the help of deeper architectures, DeGNN

Table 5: Test accuracy (in %) on other transductive datasets. * indicates that we ran our own implementation. We use AUC on the eBay Small/Large because of class imbalance. We use bold font for methods with the highest average accuracy.

Models	eBay Small	eBay Large	Tencent	Amazon Computer	Amazon Photo	Coauthor CS	Coauthor Physics	Actor	Chemeleon
GAT*	70.4±0.5	80.6±0.5	55.0±0.3	80.1±0.6	85.7±1.0	87.4±0.2	90.2±1.4	27.7±0.5	26.6±0.9
GCN*	73.1±0.6	80.5±0.4	55.7±0.2	82.4±0.4	85.9±0.6	**90.7±0.2**	92.7±1.1	27.0±0.8	25.4±1.1
JK-Net*	71.9±0.3	80.7±0.4	56.5±0.4	82.0±0.6	85.9±0.7	89.5±0.6	92.5±0.4	24.7±0.9	27.1±1.2
ResGCN*	73.0±0.5	80.2±0.4	56.2±0.3	81.1±0.7	85.3±0.9	87.9±0.6	92.2±1.5	27.0±1.2	26.1±1.3
DenseGCN*	73.5±0.3	80.9±0.2	56.9±0.4	81.3±0.9	84.9±1.1	88.4±0.8	91.9±1.4	26.2±0.8	22.1±1.7
DeGNN(GCN)*	71.6±0.3	81.1±0.2	**58.6±0.2**	82.8±0.6	**86.3±0.4**	89.5±0.5	92.4±0.5	**29.2±0.5**	26.1±0.8
DeGNN(JK)*	72.4±0.4	80.7±0.3	57.8±0.4	82.5±0.7	86.1±0.7	90.5±0.4	92.2±0.5	28.5±0.5	**27.3±0.9**
DeGNN(Res)*	**74.0±0.4**	81.1±0.4	57.2±0.5	82.5±0.5	85.8±0.9	90.1±0.2	**92.9±0.6**	28.6±0.8	27.1±0.9
DeGNN(Dense)*	73.7±0.6	**81.2±0.3**	58.4±0.4	**83.1±0.5**	86.2±0.8	90.2±0.1	92.1±1.7	28.8±0.9	26.3±0.8

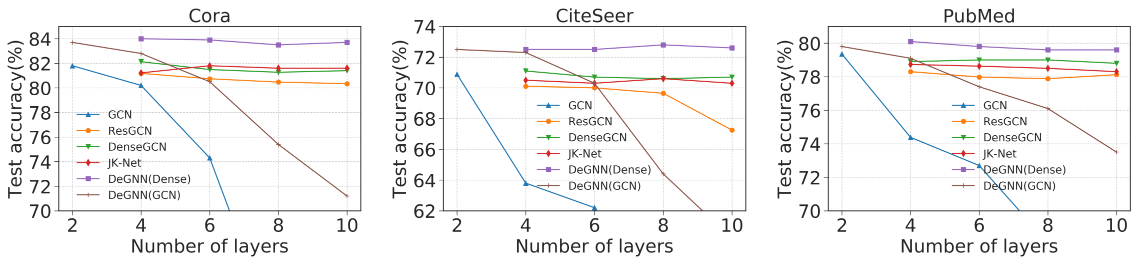

Figure 2: Influence of model depth (number of layers) on classification performance. More details are in Table 6.

with more advanced base model can outperform current state-of-the-art methods. Specifically, DeGNN(Dense) achieves a remarkable 84.3% testing accuracy with 5 layers on Cora.

To demonstrate the generality of DeGNN, we also have a systematical evaluation on other datasets in a variety of domains, such as Coauthor CS, Coauthor Physics, Amazon Computers and Amazon Photo. Table 5 demonstrates that, in general, DeGNN outperforms GCN, JK-Net, ResGCN and DenseGCN and GAT, and the base models can benefit from DeGNN. We make a further evaluation on industry production datasets: eBay Small, eBay Large and Tencent. DeGNNs easily achieves great performance improvements over existing GCN base models even on these real-world applications.

Inductive. Besides the transductive tasks, we also evaluate on the inductive ones. For inductive settings, only the nodes in the training set can be used during the training procedure, and the validation set and testing set are only involved for evaluation procedure. As it is not consistent with the standard GCN setting, we replace the baselines with some recent inductive models, including GraphSAGE, ClusterGCN, FastGCN and GraphSAINT. These models are naturally suitable for inductive tasks due to the node sampling techniques. For DeGNN, we keep the full batch training scheme as GCN and add an additional decomposition step when involving the validation set and the testing set. It is excited to see that DeGNN can still achieve competitive results in Table 4. We think it is an interesting future work to design an end-to-end framework that can automatically combine DeGNN with graph sampling based methods for inductive scenarios.

4.3 Analysis

4.3.1 Analysis of model architecture depth. Here, we investigate the influence of model depth (number of layers) on classification

Table 6: Testing accuracy (%) comparisons on different models

		4 layers		6 layers		8 layers	
Dataset	Model	Original	DeGNN	Original	DeGNN	Original	DeGNN
Cora	GCN	80.2	**82.8**	74.3	**80.5**	59.4	**75.4**
	ResGCN	81.2	**83.7**	80.7	**83.4**	80.5	**82.4**
	JK-Net	81.2	**83.6**	81.8	**83.9**	81.6	**83.7**
	DenseGCN	82.1	**84.0**	81.5	**83.5**	81.3	**83.3**
Citeseer	GCN	63.8	**72.3**	62.2	**70.3**	47.4	**64.4**
	ResGCN	70.1	**72.4**	70.0	**71.8**	69.6	**71.8**
	JK-Net	70.5	**73.1**	70.3	**72.8**	70.6	**72.7**
	DenseGCN	71.1	**72.5**	70.7	**72.5**	70.6	**72.8**
Pubmed	GCN	74.4	**79.1**	72.7	**77.4**	68.1	**76.1**
	ResGCN	78.3	**79.5**	78.0	**79.6**	77.9	**79.4**
	JK-Net	78.8	**80.0**	78.6	**79.8**	78.5	**79.6**
	DenseGCN	78.9	**80.1**	79.0	**79.4**	79.0	**79.2**

performance on the three citation datasets. We compare DeGNN (GCN) and DeGNN (Dense) with ResGCN, JK-Net, and DenseGCN. When the model depth is two, all baselines degenerate to the original 2-layer GCN model. As shown in Figure 2, for the original GCN, it gets the best results with a 2-layer model and its performance decreases rapidly with the increase of layers. For ResGCN, DenseGCN, and JK-Net, they can keep more information on the original features compared with GCN and get a relatively good performance, but perform much worse than DeGNN (Dense). Even with 10 layers, DeGNN does not decrease in performance as the other baselines do, and outperforms the best SOTA on all datasets. Table 6 shows a detailed version of the influence of model depth for different models on the three citation datasets.

4.3.2 Analysis of decomposition parameter K. The number of decomposed subgraphs K is an important parameter in our framework. To analyze its influence, we conduct an experiment on three citation networks, and the results are illustrated in Figure 3. Here we set the

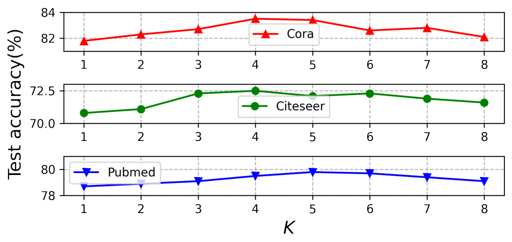

Figure 3: Test accuracy for different parameter K.

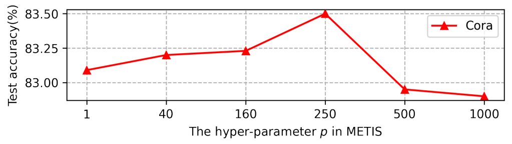

Figure 4: Test accuracy for parameter p in METIS.

hyper-parameter in METIS $p = 100$. As we can see, the best number of decomposed subgraphs K for Cora and Citeseer is 4 and it is 5 for Pubmed. As K grows from 1, the test accuracy increases until it reaches the maximum point and it decreases when K is larger. These results imply that with the proposed spanning-tree-based sampling framework, the optimal graph decomposition parameter K is around 4 or 5 to achieve the best performance.

4.3.3 Analysis of hyper-parameter p. The proposed spanning forest based graph decomposition can control the graph connectivity with the METIS partition step p. For simplicity, we evaluate on a standard two-layer GCN on Cora, and only the first layer is replace by a decomposed GCN with $K = 4$. Other hyperparameters are selected by grid search on the validation set. We tune the hyper-parameter p in METIS, which generates different connected components and results in different sizes of spanning tree. Then we test its influence on the final testing accuracy. As shown in Figure 4, as p increases, the testing accuracy improves at first, but drops down quickly at last. This is because the METIS eliminate too many edges cuts and result in a loss of graph connectivity. In the experiments above, we show that graph decomposition does contribute to better performance, while proper connectivity is also crucial for achieving good performance. To conclude, there exists a trade-off between graph decomposition and graph connectivity.

5 RELATED WORK

GCN and its variants. GCNs have achieved promising results on various graph applications, while one limitation of GCN is that its performance would not improve with the increase of network depths. For instance, [15] show that a two-layer GCN would achieve the best performance on a classic graph dataset while stacking more layers cannot help to improve the performance. Several studies have been conducted [38] trying to figure out the reasons behind the depth limitation and provide workarounds. DropEdge [29] aims to address the oversmoothing problem by randomly removing some edges from the graph. There is also a rising interest in deepening GCN by utilizing some techniques that are used to build deeper CNN architectures (e.g., ResGCN [15], DenseGCN [18], JK-Net [40]). However, these lacks of evidence showing whether these techniques are helpful to improve the performance of general GNNs. [19] shows that GCN is a special form of Laplacian smoothing, and they prove that, under certain conditions, by repeatedly applying Laplacian smoothing many times, the features of vertices within each connected component of the graph will converge to the same value. Therefore, the oversmoothing property of GCN will make the features indistinguishable and thus hurt the classification accuracy. [25] conducts more engaged theoretical analysis. Compared with these work, we aim to go beyond the analysis of oversmoothing. Instead, we theoretically show that the decomposition in DeGNN can help to slow down such information loss, which in turn inspires practical graph decomposition algorithm for general graph-structured data.

Graph decomposition methods. Existing graph decomposition approaches can be mainly divided into two categories, namely edge-oriented algorithms (i.e., edge-cut or 1D partition) or vertex-oriented algorithms (i.e., vertex-cut or 2D block partition). The first category algorithms are proposed to partition a graph by dividing it by vertices and minimize the number of crossing edges whose endpoints are in different partitions [28]. The most representative algorithm is METIS [14], which has been applied extensively in many areas so far. It approximates the graph partition process with a novel multilevel scheme and significantly reduces the partitioning time cost. Recently, some GNN algorithms (e.g., ClusterGCN [6]) and distributed GNN training systems (e.g., AliGraph [45]) have been proposed to adapt METIS to obtain memory/communication reduction due to the neighborhood explosion. Another category studies are based on vertex-cut algorithms that directly assign the edges to different partitions, where part of graph vertices are replicated between other partitions. For "edge-centric" graph processing models (e.g., PowerGraph [10], cuWide [24]), vertex-cut algorithms are easy to keep the workload balanced. They use a sequential greedy heuristic, which distributes the next edge e to a partition that minimizes the conditional expected replication factor.

6 CONCLUSION

In this paper, we investigated the importance of graph decomposition in graph neural networks. We theoretically verified that graph decomposition can help avoid the information loss problem caused by increasing networks depth. To utilize the information preserving ability of the decomposition in general graph-structured data , we introduce a novel connectivity-aware graph decomposition to balance the trade-off between information loss and model performance of GNNs. We conducted extensive experiments on ten datasets and analyzed the property of our model. Our model achieves state-of-the-art performances and could better preserve information with deeper architectures.

7 ACKNOWLEDGEMENT

This work is supported by the National Key Research and Development Program of China (No. 2018YFB1004403), the National Natural Science Foundation of China (No. 61832001, U1936104), PKU-Tencent joint research Lab, Beijing Academy of Artificial Intelligence (BAAI), CAAI Huawei MindSpore Open Fund, and The Fundamental Research Funds for the Central Universities 2020RC25. Ce Zhang and the DS3Lab gratefully acknowledge the support from the Swiss National Science Foundation (Project Number 200021_184628), Innosuisse/SNF BRIDGE Discovery (Project Number 40B2-0_187132), European Union Horizon 2020 Research and Innovation Programme (DAPHNE, 957407), Botnar Research Centre for Child Health, Swiss Data Science Center, Alibaba, Cisco, eBay, Google Focused Research Awards, Oracle Labs, Swisscom, Zurich Insurance, Chinese Scholarship Council, and the Department of Computer Science at ETH Zurich.

REFERENCES

[1] 2020. DropEdge openreview. https://openreview.net/forum?id=Hkx1qkrKPr.
[2] Sami Abu-El-Haija, Amol Kapoor, Bryan Perozzi, and Joonseok Lee. 2019. N-GCN: Multi-scale Graph Convolution for Semi-supervised Node Classification. In *UAI*.
[3] Rianne van den Berg, Thomas N Kipf, and Max Welling. 2017. Graph convolutional matrix completion. *arXiv preprint arXiv:1706.02263* (2017).
[4] Shaosheng Cao, Xinxing Yang, Cen Chen, Jun Zhou, Xiaolong Li, and Yuan Qi. 2019. TitAnt: Online Real-time Transaction Fraud Detection in Ant Financial. *PVLDB* 12, 12 (2019), 2082–2093.
[5] Jie Chen, Tengfei Ma, and Cao Xiao. 2018. FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling. In *ICLR*.
[6] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. 2019. Cluster-GCN: An Efficient Algorithm for Training Deep and Large Graph Convolutional Networks. In *SIGKDD*. 257–266.
[7] Yodsawalai Chodpathumwan, Amirhossein Aleyasen, Arash Termehchy, and Yizhou Sun. 2015. Universal-DB: Towards Representation Independent Graph Analytics. *VLDB* 8, 12 (2015), 2016–2019.
[8] Wenfei Fan, Kun He, Qian Li, and Yue Wang. 2020. Graph algorithms: parallelization and scalability. *Sci. China Inf. Sci.* 63, 10 (2020), 1–21.
[9] Hongyang Gao, Zhengyang Wang, and Shuiwang Ji. 2018. Large-scale learnable graph convolutional networks. In *SIGKDD*. 1416–1424.
[10] Joseph E. Gonzalez, Yucheng Low, Haijie Gu, Danny Bickson, and Carlos Guestrin. 2012. PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs. In *OSDI*. 17–30.
[11] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *NeurIPS*. 1024–1034.
[12] Louis Jachiet, Pierre Genevès, Nils Gesbert, and Nabil Layaïda. 2020. On the Optimization of Recursive Relational Queries: Application to Graph Queries. In *SIGMOD*. 681–697.
[13] Chathura Kankanamge, Siddhartha Sahu, Amine Mhedbhi, Jeremy Chen, and Semih Salihoglu. 2017. Graphflow: An Active Graph Database. In *SIGMOD*. 1695–1698.
[14] George Karypis and Vipin Kumar. 1998. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing* 20, 1 (1998), 359–392.
[15] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*.
[16] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. 2019. Predict then Propagate: Graph Neural Networks meet Personalized PageRank. In *ICLR*.
[17] Boris Knyazev, Xiao Lin, Mohamed R Amer, and Graham W Taylor. 2018. Spectral Multigraph Networks for Discovering and Fusing Relationships in Molecules. *arXiv preprint arXiv:1811.09595* (2018).
[18] Guohao Li, Matthias Müller, Ali Thabet, and Bernard Ghanem. 2019. Can GCNs Go as Deep as CNNs? *arXiv preprint arXiv:1904.03751* (2019).
[19] Qimai Li, Zhichao Han, and Xiao-Ming Wu. 2018. Deeper insights into graph convolutional networks for semi-supervised learning. In *AAAI*.
[20] Renjie Liao, Marc Brockschmidt, Daniel Tarlow, Alexander L. Gaunt, Raquel Urtasun, and Richard S. Zemel. 2018. Graph Partition Neural Networks for Semi-Supervised Classification. In *ICLR*.
[21] Ziqi Liu, Zhengwei Wu, Zhiqiang Zhang, Jun Zhou, Shuang Yang, Le Song, and Yuan Qi. 2020. Bandit Samplers for Training Graph Neural Networks. In *NeurIPS*.
[22] Sitao Luan, Mingde Zhao, Xiao-Wen Chang, and Doina Precup. 2019. Break the Ceiling: Stronger Multi-scale Deep Graph Convolutional Networks. In *NeurIPS*.
[23] Jianxin Ma, Peng Cui, Kun Kuang, Xin Wang, and Wenwu Zhu. 2019. Disentangled Graph Convolutional Networks. In *ICML*, Vol. 97. 4212–4221.
[24] X. Miao, L. Ma, Z. Yang, Y. Shao, B. Cui, L. Yu, and J. Jiang. 2020. CuWide: Towards Efficient Flow-based Training for Sparse Wide Models on GPUs. *TKDE* (2020), 1–1. https://doi.org/10.1109/TKDE.2020.3038109
[25] Kenta Oono and Taiji Suzuki. 2019. On Asymptotic Behaviors of Graph CNNs from Dynamical Systems Perspective. *arXiv preprint arXiv:1905.10947* (2019).
[26] Chanyoung Park, Carl Yang, Qi Zhu, Donghyun Kim, Hwanjo Yu, and Jiawei Han. 2020. Unsupervised Differentiable Multi-aspect Network Embedding. In *SIGKDD*. 1435–1445.
[27] Zhen Peng, Wenbing Huang, Minnan Luo, Qinghua Zheng, Yu Rong, Tingyang Xu, and Junzhou Huang. 2020. Graph Representation Learning via Graphical Mutual Information Maximization. In *WWW*. 259–270.
[28] Hannu Reittu, Ilkka Norros, Tomi Räty, Marianna Bolla, and Fülöp Bazsó. 2019. Regular Decomposition of Large Graphs: Foundation of a Sampling Approach to Stochastic Block Model Fitting. *Data Sci. Eng.* 4, 1 (2019), 44–60.
[29] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. 2019. DropEdge: Towards Deep Graph Convolutional Networks on Node Classification. In *ICLR*.
[30] Andrew M Saxe, Yamini Bansal, Joel Dapello, Madhu Advani, Artemy Kolchinsky, Brendan D Tracey, and David D Cox. 2019. On the information bottleneck theory of deep learning. *Journal of Statistical Mechanics: Theory and Experiment* 2019, 12 (2019), 124020.
[31] Ohad Shamir, Sivan Sabato, and Naftali Tishby. 2010. Learning and generalization with the information bottleneck. *Theoretical Computer Science* 411, 29-30 (2010).
[32] Felipe Petroski Such, Shagan Sah, Miguel Alexander Dominguez, Suhas Pillai, Chao Zhang, Andrew Michael, Nathan D Cahill, and Raymond Ptucha. 2017. Robust spatial filtering with graph convolutional neural networks. *IEEE Journal of Selected Topics in Signal Processing* 11, 6 (2017), 884–896.
[33] Emre Telatar. 1999. Capacity of multi-antenna Gaussian channels. *European transactions on telecommunications* 10 (1999), 585–595. Issue 6.
[34] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2018. Graph attention networks. *ICLR* (2018).
[35] Petar Velickovic, William Fedus, William L. Hamilton, Pietro Liò, Yoshua Bengio, and R. Devon Hjelm. 2019. Deep Graph Infomax. In *ICLR*.
[36] Felix Wu, Amauri H. Souza Jr., Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Q. Weinberger. 2019. Simplifying Graph Convolutional Networks. In *ICML*.
[37] Shiwen Wu, Yuanxing Zhang, Chengliang Gao, Kaigui Bian, and Bin Cui. 2020. GARG: Anonymous Recommendation of Point-of-Interest in Mobile Networks by Graph Convolution Network. *Data Sci. Eng.* 5, 4 (2020), 433–447.
[38] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S Yu. 2019. A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596* (2019).
[39] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How Powerful are Graph Neural Networks?. In *ICLR*.
[40] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. 2018. Representation Learning on Graphs with Jumping Knowledge Networks. In *ICML*. 5449–5458.
[41] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor K. Prasanna. 2020. GraphSAINT: Graph Sampling Based Inductive Learning Method. In *ICLR*.
[42] Wentao Zhang, Xupeng Miao, Yingxia Shao, Jiawei Jiang, Lei Chen, Olivier Ruas, and Bin Cui. 2020. Reliable Data Distillation on Graph Convolutional Network. In *SIGMOD*. 1399–1414.
[43] Da Zheng, Minjie Wang, Quan Gan, Zheng Zhang, and George Karypis. 2020. Scalable Graph Neural Networks with Deep Graph Library. In *SIGKDD*. 3521–3522.
[44] Hao Zhong and Hong Mei. 2020. Learning a graph-based classifier for fault localization. *Sci. China Inf. Sci.* 63, 6 (2020).
[45] Rong Zhu, Kun Zhao, Hongxia Yang, Wei Lin, Chang Zhou, Baole Ai, Yong Li, and Jingren Zhou. 2019. AliGraph: A Comprehensive Graph Neural Network Platform. *VLDB* 12, 12 (2019), 2094–2105.
[46] Chenyi Zhuang and Qiang Ma. 2018. Dual graph convolutional networks for graph-based semi-supervised classification. In *WWW*. 499–508.

A PROOFS

We begin by introducing our notation. Hereafter, scalars will be written in italics, vectors in bold lower-case and matrices in bold upper-case letters. For an $m \times n$ real matrix $\mathbf{A}$, the matrix element in the ith row and jth column is denoted as $(\mathbf{A})_{ij}$, and ith entry of a vector $\mathbf{a} \in \mathbb{R}^m$ by $(\mathbf{a})_i$. Also, jth column of $\mathbf{A}$ is denoted by $(\mathbf{A})_j$, or $(\mathbf{A})_{[i=1,2,\ldots,m],j}$. Similarly, we denote ith row by $(\mathbf{A})_{i,[j=1,2,\ldots,n]}$. The inner product between two vectors $(\mathbf{A})_i$ and $(\mathbf{A})_{i'}$ is denoted by $\langle(\mathbf{A})_i, (\mathbf{A})_{i'}\rangle$.

We vectorize a matrix $\mathbf{A}$ by concatenating its columns such that

$$\mathrm{vec}(\mathbf{A}) = \begin{bmatrix} (\mathbf{A})_1 \\ (\mathbf{A})_2 \\ \vdots \\ (\mathbf{A})_n \end{bmatrix}$$

and denote it by vec($\mathbf{A}$). For matrices $\mathbf{A} \in \mathbb{R}^{m\times n}$ and $\mathbf{B} \in \mathbb{R}^{k\times l}$, we denote the kronecker product of $\mathbf{A}$ and $\mathbf{B}$ by $\mathbf{A} \otimes \mathbf{B}$ such that

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} (\mathbf{A})_{11}\mathbf{B} & \ldots & (\mathbf{A})_{1n}\mathbf{B} \\ \vdots & \ddots & \vdots \\ (\mathbf{A})_{m1}\mathbf{B} & \ldots & (\mathbf{A})_{mn}\mathbf{B} \end{bmatrix}.$$

Note that $\mathbf{A} \otimes \mathbf{B}$ is of size $mk \times nl$.

Next, we list some existing results which we require repeatedly throughout this section.

Preliminaries.

(1) Suppose $\mathbf{A} \in \mathbb{R}^{m\times n}$, $\mathbf{B} \in \mathbb{R}^{n\times k}$ and $\mathbf{C} \in \mathbb{R}^{k\times p}$. We have
$$\mathrm{vec}(\mathbf{ABC}) = (\mathbf{C}^T \otimes \mathbf{A})\,\mathrm{vec}(\mathbf{B}). \tag{4}$$
(2) Let $\mathbf{A} \in \mathbb{R}^{m\times n}$, $\mathbf{B} \in \mathbb{R}^{n\times k}$ and $\mathbf{C} \in \mathbb{R}^{m'\times n'}$, $\mathbf{D} \in \mathbb{R}^{n'\times k'}$
$$(\mathbf{AB} \otimes \mathbf{CD}) = (\mathbf{A} \otimes \mathbf{C})(\mathbf{B} \otimes \mathbf{D}). \tag{5}$$
(3) For $\mathbf{A} \in \mathbb{R}^{m\times m}$ and $\mathbf{B} \in \mathbb{R}^{n\times n}$, singular values of $\mathbf{A} \otimes \mathbf{B}$ is given by $\lambda_i(\mathbf{A})\lambda_j(\mathbf{B})$, $i = 1, 2, \dots, m$ and $j = 1, 2, \dots, n$.

(4) Let $\mathbf{x}$ and $\mathbf{y}$ be an n-dimensional random vector defined over finite alphabets $\mathcal{X}^n$ and Ω^n, respectively. We denote entropy of $\mathbf{x}$ by $\mathcal{H}(\mathbf{x})$ and mutual information between $\mathbf{x}$ and $\mathbf{y}$ by $\mathcal{I}(\mathbf{x};\mathbf{y})$. We list the followings:
$$\mathcal{H}(f(\mathbf{x})) \overset{(a)}{\leq} \mathcal{H}(\mathbf{x}),\ \mathcal{I}(\mathbf{x}; f(\mathbf{y})) \overset{(b)}{\leq} \mathcal{I}(\mathbf{x};\mathbf{y}) \tag{6}$$
such that $f : \mathbb{R} \longrightarrow \mathbb{R}$ is some deterministic function, and equality holds for both inequalities *iff* f is bijective.

Proof of Lemma 1. Applying vectorization to the layer-wise propagation rule introduced in (1), we have
$$\begin{aligned}
\mathbf{y}^{(i+1)} &= \mathrm{vec}\left(\sigma(\mathbf{AY}^{(i)}\mathbf{W}^{(i+1)})\right)\\
\mathbf{y}^{(i+1)} &\overset{(a)}{=} \sigma\left(\mathrm{vec}(\mathbf{AY}^{(i)}\mathbf{W}^{(i+1)})\right)\\
\mathbf{y}^{(i+1)} &\overset{(b)}{=} \sigma\left(((\mathbf{W}^{(i+1)})^T \otimes \mathbf{A})\mathbf{y}^{(i)}\right)\\
\mathbf{y}^{(i+1)} &\overset{(c)}{=} \mathbf{P}^{(i+1)}((\mathbf{W}^{(i+1)})^T \otimes \mathbf{A})\mathbf{y}^{(i)}
\end{aligned} \tag{7}$$
where (a) follows from the element-wise application of σ, (b) follows from (4), and (c) results from introducing a diagonal matrix $\mathbf{P}^{(i+1)}$ with diagonal entries in $\{a, 1\}$ such that $(\mathbf{P}^{(i+1)})_{j,j} = 1$ if $\left((\mathbf{W}^{(i+1)} \otimes \mathbf{A})\mathbf{y}^{(i)}\right)_j \geq 0$, and $(\mathbf{P}^{(i+1)})_{j,j} = a$ elsewhere.

By a recursive application of (7c), we have
$$\mathbf{y}^{(l)} = \mathbf{P}^{(l)}(\mathbf{W}^{(l)} \otimes \mathbf{A}) \dots \mathbf{P}^{(2)}(\mathbf{W}^{(2)} \otimes \mathbf{A})\mathbf{P}^{(1)}(\mathbf{W}^{(1)} \otimes \mathbf{A})\mathbf{x}.$$
□

We drop the transpose from $\mathbf{W}^{(i+1)}$ in order to avoid cumbersome notation. The singular values of $\mathbf{W}^{(i+1)}$ are our primary interest thereof our results still hold.

Following Lemma 1, the next key step in our proving is as follows.

Lemma 2. *Consider the singular value decomposition* $\mathbf{U\Lambda V}^T = \mathbf{P}^{(l)}(\mathbf{W}^{(l)} \otimes \mathbf{A})...\mathbf{P}^{(2)}(\mathbf{W}^{(2)} \otimes \mathbf{A})\mathbf{P}^{(1)}(\mathbf{W}^{(1)} \otimes \mathbf{A})$ *such that* $(\mathbf{\Lambda})_{j,j} = \lambda_j(\mathbf{P}^{(l)}(\mathbf{W}^{(l)} \otimes \mathbf{A})...\mathbf{P}^{(2)}(\mathbf{W}^{(2)} \otimes \mathbf{A})\mathbf{P}^{(1)}(\mathbf{W}^{(1)} \otimes \mathbf{A}))$, *and let* $\tilde{\mathbf{x}} = \mathbf{V}^T\mathbf{x}$. *We have*
$$\mathcal{I}(\mathbf{x};\mathbf{y}^{(l)}) \overset{(1)}{=} \mathcal{I}(\tilde{\mathbf{x}};\mathbf{\Lambda}\tilde{\mathbf{x}}) \overset{(2)}{\leq} \mathcal{H}(\tilde{\mathbf{x}}) \overset{(3)}{=} \mathcal{H}(\mathbf{x}) \tag{8}$$
where (1, 3) results from that $\mathbf{U}$ and $\mathbf{V}$ are invertible, and equality holds in (2) *iff* $\mathbf{\Lambda}$ is invertible, i.e., singular values of $\mathbf{P}^{(l)}(\mathbf{W}^{(l)} \otimes \mathbf{A})...\mathbf{P}^{(2)}(\mathbf{W}^{(2)} \otimes \mathbf{A})\mathbf{P}^{(1)}(\mathbf{W}^{(1)} \otimes \mathbf{A})$ are nonzero.

Theorem 1, 2, 3 and 4 can easily be inferred from Lemma 2. That is, $\mathcal{I}(\mathbf{x};\mathbf{y}^{(l)}) = 0$ *iff* $\max_j(\mathbf{\Lambda}^l)_{j,j} = 0$ in the asymptotic regime. Similarly, *iff* $\min_j(\mathbf{\Lambda}^l)_{j,j} > 0$, $\mathcal{I}(\mathbf{x};\mathbf{y}^{(l)})$ is maximized and given by $\mathcal{H}(\mathbf{x})$, hence $\mathcal{L}(\mathbf{y}^{(l)}) = 0$.

In particular Theorem 1, 3 and Corollary 2, i.e., exponential decay to zero, also hold for traditional ReLU with $f : x \longrightarrow x^+ = \max(0, x)$.

Proof of Lemma 2. Let Σ be a $n\times n$ matrix with singular value decomposition $\Sigma = \mathbf{U\Lambda V}^T$. Inspired by the derivation for the capacity of deterministic channels introduced by [33], we derive the following
$$\begin{aligned}
\mathcal{I}(\mathbf{x};\Sigma\mathbf{x}) &= \mathcal{I}(\mathbf{x};\mathbf{U\Lambda V}^T\mathbf{x}) \overset{(a)}{=} \mathcal{I}(\mathbf{x};\mathbf{\Lambda V}^T\mathbf{x})\\
\mathcal{I}(\mathbf{x};\Sigma\mathbf{x}) &\overset{(b)}{=} \mathcal{I}(\mathbf{V}^T\mathbf{x};\mathbf{\Lambda V}^T\mathbf{x}) \overset{(c)}{=} \mathcal{I}(\tilde{\mathbf{x}};\mathbf{\Lambda}\tilde{\mathbf{x}}).
\end{aligned} \tag{9}$$
(a) and (b) are a result of (6b) and that $\mathbf{U}$ and $\mathbf{V}$ are unitary hence invertible (bijective) transformations. (c) follows from the change of variables $\tilde{\mathbf{x}} = \mathbf{V}^T\mathbf{x}$.

Note that $\mathcal{I}(\tilde{\mathbf{x}};\mathbf{\Lambda}\tilde{\mathbf{x}}) \leq \mathcal{H}(\mathbf{\Lambda}\tilde{\mathbf{x}})$. Using (6a), we further have $\mathcal{H}(\mathbf{\Lambda}\tilde{\mathbf{y}}) \leq \mathcal{H}(\tilde{\mathbf{x}}) = \mathcal{H}(\mathbf{x})$ which completes the proof. □

We recall that we are interested in regimes where $\mathcal{I}(\mathbf{x};\mathbf{y}^{(l)}) = 0$ and $\mathcal{L}(\mathbf{y}^{(l)}) = 0$. Lemma 2 shows that $\mathcal{I}(\mathbf{x};\mathbf{y}^{(l)}) = 0$ if $\max_j \lambda_j(\mathbf{P}^{(l)}(\mathbf{W}^{(l)} \otimes \mathbf{A}) \cdots \mathbf{P}^{(2)}(\mathbf{W}^{(2)} \otimes \mathbf{A})\mathbf{P}^{(1)}(\mathbf{W}^{(1)} \otimes \mathbf{A})) = 0$, and maximized (and given by $\mathcal{H}(\mathbf{x})$) when $\mathbf{P}^{(l)}(\mathbf{W}^{(l)} \otimes \mathbf{A}) \cdots \mathbf{P}^{(2)}(\mathbf{W}^{(2)} \otimes \mathbf{A})\mathbf{P}^{(1)}(\mathbf{W}^{(1)} \otimes \mathbf{A})$ is invertible. Therefore, maximum and minimum singular values of $\mathbf{P}^{(l)}(\mathbf{W}^{(l)} \otimes \mathbf{A}) \cdots \mathbf{P}^{(2)}(\mathbf{W}^{(2)} \otimes \mathbf{A})\mathbf{P}^{(1)}(\mathbf{W}^{(1)} \otimes \mathbf{A})$ are of our interest.

Proof of Theorem 1. Let $\sigma_{\mathbf{A}} = \max_j \lambda_j(\mathbf{A})$ and $\sigma_{\mathbf{W}} = \sup_i \max_j \lambda_j(\mathbf{W}^{(i)})$. Given singular values of $\mathbf{P}^{(i)}$ is in $\{a, 1\}$, $\sup_i \max_j \lambda_j(\mathbf{P}^{(i)}(\mathbf{W}^{(i)} \otimes \mathbf{A})) = \sigma_{\mathbf{A}}\sigma_{\mathbf{W}}$. We, moreover, have $\max_j \lambda_j(\mathbf{P}^{(l)}(\mathbf{W}^{(l)} \otimes \mathbf{A}) \cdots \mathbf{P}^{(2)}(\mathbf{W}^{(2)} \otimes \mathbf{A})\mathbf{P}^{(1)}(\mathbf{W}^{(1)} \otimes \mathbf{A})) \leq (\sigma_{\mathbf{A}}\sigma_{\mathbf{W}})^l$. Therefore, if $\sigma_{\mathbf{A}}\sigma_{\mathbf{W}} < 1$, by Lemma 2 we have $\mathcal{I}(\mathbf{x};\mathbf{y}^{(l)}) = O((\sigma_{\mathbf{A}}\sigma_{\mathbf{W}})^l)$, and $\lim_{l\to\infty} \mathcal{I}(\mathbf{x};\mathbf{y}^{(l)}) = 0$. □

Proof of Theorem 2. We now denote $\gamma_{\mathbf{A}} = \min_j \lambda_j(\mathbf{A})$ and $\gamma_{\mathbf{W}} = \inf_i \min_j \lambda_j(\mathbf{W}^{(i)})$. Hence $\inf_i \min_j \lambda_j(\mathbf{P}^{(i)}(\mathbf{W}^{(i)} \otimes \mathbf{A})) = a\gamma_{\mathbf{A}}\gamma_{\mathbf{W}}$. Moreover, $\min_j \lambda_j(\mathbf{P}^{(l)}(\mathbf{W}^{(l)} \otimes \mathbf{A}) \cdots \mathbf{P}^{(2)}(\mathbf{W}^{(2)} \otimes \mathbf{A})\mathbf{P}^{(1)}(\mathbf{W}^{(1)} \otimes \mathbf{A})) \geq (a\gamma_{\mathbf{A}}\gamma_{\mathbf{W}})^l$. If $a\gamma_{\mathbf{A}}\gamma_{\mathbf{W}} \geq 1$, $\min_j \lambda_j(\mathbf{P}_l(\mathbf{W}^{(l)} \otimes \mathbf{A}) \cdots \mathbf{P}_2(\mathbf{W}^{(2)} \otimes \mathbf{A})\mathbf{P}_1(\mathbf{W}^{(1)} \otimes \mathbf{A})) \geq 1\ \forall l \in \mathbb{N}^+$, hence $\mathcal{I}(\mathbf{x};\mathbf{y}^{(l)}) = \mathcal{H}(\mathbf{x})$ and $\mathcal{L}(\mathbf{y}^{(l)}) = 0$ results by Lemma 2. □

Proof of Corollary 1. Let $\mathbf{D}$ denote the degree matrix such that $(\mathbf{D})_{j,j} = \sum_m(\mathbf{A})_{j,m}$, and $\mathbf{L}$ be the associated normalized Laplacian $\mathbf{L} = \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$. Due to the property of normalized Laplacian such that $\max_j \lambda_j(\mathbf{L}) = 1$, we have $\sigma_{\mathbf{A}} = 1$. Inserting this into Theorem 1, the corollary results. □

Similarly as in (7), $\mathbf{y}^{(i+1)}$ can be derived from (2) as follows:
$$\begin{aligned}
&\mathbf{y}^{(i+1)} = \mathrm{vec}\left(\sigma(\sum_k \mathbf{A}_k\mathbf{Y}^{(i)}\mathbf{W}_k^{(i+1)})\right) \overset{(a)}{=} \sigma(\sum_k \mathrm{vec}(\mathbf{A}_k\mathbf{Y}^{(i)}\mathbf{W}_k^{(i+1)}))\\
&\mathbf{y}^{(i+1)} \overset{(b)}{=} \sigma(\sum_k(\mathbf{W}_k^{(i+1)} \otimes \mathbf{A}_k)\mathbf{y}^{(i)}\sigma) \overset{(c)}{=} \mathbf{P}^{(i+1)} \sum_k(\mathbf{W}_k^{(i+1)} \otimes \mathbf{A}_k)\mathbf{y}^{(i)}
\end{aligned} \tag{10}$$
where $\mathbf{P}^{(i+1)}$ is a diagonal matrix with diagonal entries in $\{a, 1\}$ with $a \in (0, 1)$ such that $(\mathbf{P}^{(i)})_{j,j} = 1$ if $\left(\sum_k(\mathbf{W}_k^{(i+1)} \otimes \mathbf{A})\mathbf{y}^{(i)}\right)_j \geq 0$, and $(\mathbf{P}^{(i)})_{j,j} = a$ otherwise.

Therefore, $\mathbf{y}^{(l)}$ is given by $\mathbf{y}^{(l)} =$
$$\mathbf{P}^{(l)} \sum_{k_l}(\mathbf{W}_{k_l}^{(l)} \otimes \mathbf{A}_{k_l}) \cdots \mathbf{P}^{(2)} \sum_{k_2}(\mathbf{W}_{k_2}^{(2)} \otimes \mathbf{A}_{k_2})\mathbf{P}^{(1)} \sum_{k_1}(\mathbf{W}_{k_1}^{(1)} \otimes \mathbf{A}_{k_1})\mathbf{x}.$$

Consider (9) where Σ is replaced with $\mathbf{P}^{(l)}\sum_{k_l}(\mathbf{W}^{(l)}_{k_l}\otimes\mathbf{A}_{k_l})\cdots\mathbf{P}^{(2)}\sum_{k_2}(\mathbf{W}^{(2)}_{k_2}\otimes\mathbf{A}_{k_2})\mathbf{P}^{(1)}\sum_{k_1}(\mathbf{W}^{(1)}_{k_1}\otimes\mathbf{A}_{k_1})$.

We deduce the followings:

Proof of Theorem 3. Suppose $\sigma^{(i)}$ denotes the largest singular value of $\mathbf{P}^{(i)}\sum_{k_i=1}^{K}(\mathbf{W}^{(i)}_{k_i}\otimes\mathbf{A}_{k_i})$ such that $\sigma^{(i)}=\max_j\lambda_j\big(\mathbf{P}^{(i)}\sum_{k_i}(\mathbf{W}^{(i)}_{k_i}\otimes\mathbf{A}_{k_i})\big)$. Following the same argument as in the proofs of Theorem 1 and 2, Lemma 2 implies that if $\sup_i\sigma^{(i)}<1$, then $\mathcal{I}(\mathbf{x};\mathbf{y}^{(l)})=O\big((\sup_i\sigma^{(i)})^l\big)$, and hence $\lim_{l\to\infty}\mathcal{I}(\mathbf{x};\mathbf{y}^{(l)})=0$ results. □

Proof of Theorem 4. We now $\gamma^{(i)}$ denote the minimum singular value of $\mathbf{P}^{(i)}\sum_{k_i=1}^{K}(\mathbf{W}^{(i)}_{k_i}\otimes\mathbf{A}_{k_i})$. By Lemma 2, it immediately follows that if $\inf_i\sigma^{(i)}\geq 1$, then $\forall l\in\mathbb{N}^+$ we have $\mathcal{L}(\mathbf{y}^{(l)})=0$. □

Before we move on to the proofs of Corollary 2 and 3, we state the following lemma.

Lemma 3. *Let the singular value decomposition of* $\mathbf{A}\in\mathbb{R}^{n\times n}$ *is given by* $\mathbf{A}=\mathbf{U_A}\mathbf{S}\mathbf{V}_\mathbf{A}^T$ *and we set each* $\mathbf{A}_k$ *to* $\mathbf{A}_k=\mathbf{U_A}\mathbf{S}_k\mathbf{V}_\mathbf{A}^T$ *with* $(\mathbf{S}_k)_{m,m}=\lambda_m(\mathbf{A})$ *if* $k=m$ *and* $(\mathbf{S}_k)_{m,m}=0$ *elsewhere. For such specific composition, we argue that singular values of* $\sum_k\mathbf{W}_k\otimes\mathbf{A}_k$ *for* $\mathbf{W}_k\in\mathbb{R}^{d\times d}$ *is given by* $\lambda_k(\mathbf{A})\lambda_j(\mathbf{W}_k)$ *for* $k=1,2,\ldots,n$ *and* $j=1,2,\ldots,d$.

Proof of Lemma 3. Let the singular value decomposition of $\mathbf{W}_k$ be $\mathbf{W}_k=\mathbf{U}_{\mathbf{W}_k}\mathbf{S}_{\mathbf{W}_k}\mathbf{V}^T_{\mathbf{W}_k}$. By the property of kronecker product, we have

$$\sum_k\mathbf{W}_k\otimes\mathbf{A}_k=\sum_k(\mathbf{U}_{\mathbf{W}_k}\otimes\mathbf{U_A})(\mathbf{S}_{\mathbf{W}_k}\otimes\mathbf{S}_k)(\mathbf{V}^T_{\mathbf{W}_k}\otimes\mathbf{V}^T_\mathbf{A}).$$

Next, we define a set of $nd\times nd$ mask matrices $\mathbf{M}_k$ such that $(\mathbf{M}_k)_{i,i'}=1$ if $i=i'$ and i (hence i') is of the form $i=k+(j-1)n$ for $j=1,2,\ldots,d$, and $(\mathbf{M}_k)_{i,i'}=0$ otherwise. Reminding that $(\mathbf{S}_k)_{m,m}=\lambda_m(\mathbf{A})$ if $k=m$ and $(\mathbf{S}_k)_{m,m}=0$ elsewhere, above equation can be rewritten as

$$\sum_k\mathbf{W}_k\otimes\mathbf{A}_k=\sum_k(\mathbf{U}_{\mathbf{W}_k}\otimes\mathbf{U_A})\mathbf{M}_k(\mathbf{S}_{\mathbf{W}_k}\otimes\mathbf{S}_k)\mathbf{M}_k(\mathbf{V}^T_{\mathbf{W}_k}\otimes\mathbf{V}^T_\mathbf{A}).$$

In other words, the mask matrix $\mathbf{M}_k$ applies on the columns (rows) of $\mathbf{U}_{\mathbf{W}_k}\otimes\mathbf{U_A}$ ($\mathbf{V}^T_{\mathbf{W}_k}\otimes\mathbf{V}^T_\mathbf{A}$) where the respective diagonal entries of $(\mathbf{S}_{\mathbf{W}_k}\otimes\mathbf{S}_k)$ are nonzero.

Next, we note that if $k=k'$, $\mathbf{M}_k\mathbf{M}_{k'}=\mathbf{M}_k$, and $\mathbf{M}_k$ and $\mathbf{M}_{k'}$ are orthogonal for $k\neq k'$. This leads us to

$$\begin{aligned}&(\mathbf{U}_{\mathbf{W}_k}\otimes\mathbf{U_A})\mathbf{M}_k(\mathbf{S}_{\mathbf{W}_k}\otimes\mathbf{S}_k)\mathbf{M}_k(\mathbf{V}^T_{\mathbf{W}_k}\otimes\mathbf{V}^T_\mathbf{A})\\&=\sum_{k'}(\mathbf{U}_{\mathbf{W}_{k'}}\otimes\mathbf{U_A})\mathbf{M}_{k'}(\mathbf{S}_{\mathbf{W}_k}\otimes\mathbf{S}_k)\sum_{k''}(\mathbf{V}^T_{\mathbf{W}_{k''}}\otimes\mathbf{V}^T_\mathbf{A})\mathbf{M}_{k''}.\end{aligned}$$

By defining $\tilde{\mathbf{U}}=\sum_k(\mathbf{U}_{\mathbf{W}_k}\otimes\mathbf{U_A})\mathbf{M}_k$ and $\tilde{\mathbf{V}}=\sum_k\mathbf{M}_k(\mathbf{V}^T_{\mathbf{W}_k}\otimes\mathbf{V}^T_\mathbf{A})$ and using the above equation, we resume $\sum_k\mathbf{W}_k\otimes\mathbf{A}_k$ as

$$\sum_k\mathbf{W}_k\otimes\mathbf{A}_k=\tilde{\mathbf{U}}\sum_k(\mathbf{S}_{\mathbf{W}_k}\otimes\mathbf{S}_k)\tilde{\mathbf{V}}^T. \tag{11}$$

Next, we will show that $\tilde{\mathbf{U}}$ and $\tilde{\mathbf{V}}$ are unitary matrices through proving that $\tilde{\mathbf{U}}\tilde{\mathbf{U}}^T=\tilde{\mathbf{U}}^T\tilde{\mathbf{U}}=\mathbf{I}$ and $\tilde{\mathbf{V}}^T\tilde{\mathbf{V}}=\tilde{\mathbf{V}}\tilde{\mathbf{V}}^T=\mathbf{I}$. To avoid repeating the same procedure, we will only show it for $\tilde{\mathbf{U}}$, but the same result also holds for $\tilde{\mathbf{V}}$.

First, we show that **(A.1)** $\tilde{\mathbf{U}}\tilde{\mathbf{U}}^T=\mathbf{I}$, and then **(A.2)** $\tilde{\mathbf{U}}^T\tilde{\mathbf{U}}=\mathbf{I}$ to argue that $\tilde{\mathbf{U}}$ (and $\tilde{\mathbf{V}}$) is unitary.

(A.1) We can simplify $\tilde{\mathbf{U}}\tilde{\mathbf{U}}^T$ as

$$\begin{aligned}\tilde{\mathbf{U}}\tilde{\mathbf{U}}^T&=\sum_k\big((\mathbf{U}_{\mathbf{W}_k}\otimes\mathbf{U_A})\mathbf{M}_k\big)\sum_{k'}\big((\mathbf{U}_{\mathbf{W}_{k'}}\otimes\mathbf{U_A})\mathbf{M}_{k'}\big)^T\\\tilde{\mathbf{U}}\tilde{\mathbf{U}}^T&=\sum_{k,k'}\big((\mathbf{U}_{\mathbf{W}_k}\otimes\mathbf{U_A})\mathbf{M}_k\big)\big((\mathbf{U}_{\mathbf{W}_{k'}}\otimes\mathbf{U_A})\mathbf{M}_{k'}\big)^T\\\tilde{\mathbf{U}}\tilde{\mathbf{U}}^T&\overset{(a)}{=}\sum_k\big((\mathbf{U}_{\mathbf{W}_k}\otimes\mathbf{U_A})\mathbf{M}_k\big)\big((\mathbf{U}_{\mathbf{W}_k}\otimes\mathbf{U_A})\mathbf{M}_k\big)^T\end{aligned}\tag{12}$$

where (a) follows from the orthogonality of $\mathbf{M}_k$ and $\mathbf{M}_{k'}$ for $k\neq k'$.

We will now take a closer look at $\sum_k\big((\mathbf{U}_{\mathbf{W}_k}\otimes\mathbf{U_A})\mathbf{M}_k\big)\big((\mathbf{U}_{\mathbf{W}_k}\otimes\mathbf{U_A})\mathbf{M}_k\big)^T$. The entries of summands, $\big((\mathbf{U}_{\mathbf{W}_k}\otimes\mathbf{U_A})\mathbf{M}_k\big)\big((\mathbf{U}_{\mathbf{W}_k}\otimes\mathbf{U_A})\mathbf{M}_k\big)^T$, are equivalent to inner product between the rows of $(\mathbf{U}_{\mathbf{W}_k}\otimes\mathbf{U_A})\mathbf{M}_k$ for a fixed k. Recall that for a fixed k, the mask matrix satisfies $(\mathbf{M}_k)_{i,i}=1$ if k is of the form $i=k+(j-1)n$ for $j=1,2,\cdots,d$, and $(\mathbf{M}_k)_{i,i}=0$ elsewhere. We now define i_ω and i_α as indices such that $i_\omega=\lfloor i/n\rfloor+1$ and $i_\alpha=\mod(i,\lfloor i/n\rfloor)$. Similarly, let $i'_\omega=\lfloor i'/n\rfloor+1$ and $i'_\alpha=\mod(i',\lfloor i'/n\rfloor)$.

Following above definitions, a moment of thought reveals that the nonzero entries of ith row of $\big((\mathbf{U}_{\mathbf{W}_k}\otimes\mathbf{U_A})\mathbf{M}_k\big)$ is given by $(\mathbf{U}_{\mathbf{W}_k})_{i_\omega,[m=1,2,\ldots,d]}(\mathbf{U_A})_{i_\alpha,k}$. We therefore investigate $(\tilde{\mathbf{U}}\tilde{\mathbf{U}}^T)_{i,i'}$ i.e., the inner product between ith and i'th rows of $\big((\mathbf{U}_{\mathbf{W}_k}\otimes\mathbf{U_A})\mathbf{M}_k\big)$ summed over all $k=1,2,\ldots,n$. To start, the inner product between ith and i'th rows of $\big((\mathbf{U}_{\mathbf{W}_k}\otimes\mathbf{U_A})\mathbf{M}_k\big)$ is as follows

$$\begin{aligned}&\langle[(\mathbf{U}_{\mathbf{W}_k})_{i_\omega,[m=1,2,\ldots,d]}(\mathbf{U_A})_{i_\alpha,k}],[(\mathbf{U}_{\mathbf{W}_k})_{i'_\omega,[m=1,2,\ldots,d]}(\mathbf{U_A})_{i'_\alpha,k}]\rangle\\&=\sum_m(\mathbf{U}_{\mathbf{W}_k})_{i_\omega,m}(\mathbf{U_A})_{i_\alpha,k}(\mathbf{U}_{\mathbf{W}_k})_{i'_\omega,m}(\mathbf{U_A})_{i'_\alpha,k}\\&=\sum_m(\mathbf{U}_{\mathbf{W}_k})_{i_\omega,m}(\mathbf{U}_{\mathbf{W}_k})_{i'_\omega,m}(\mathbf{U_A})_{i_\alpha,k}(\mathbf{U_A})_{i'_\alpha,k}\\&=(\mathbf{U_A})_{i_\alpha,k}(\mathbf{U_A})_{i'_\alpha,k}\sum_m(\mathbf{U}_{\mathbf{W}_k})_{i_\omega,m}(\mathbf{U}_{\mathbf{W}_k})_{i'_\omega,m}.\end{aligned}\tag{13}$$

Then we could complete the proof of **(A.1)** by analyzing the cases when (1) $i\neq i'$, and (2) $i=i'$ respectively. Within the space limitation, we hide the details as well as the similar proofs of **(A.2)**. □

For the decomposition of $\mathbf{A}$ such that $\mathbf{A}_k=\mathbf{U_A}\mathbf{S}_k\mathbf{V}^T_\mathbf{A}$ where the singular value decomposition of $\mathbf{A}$ is given by $\mathbf{A}=\mathbf{U_A}\mathbf{S}\mathbf{V}^T_\mathbf{A}$, we recall Theorem 3 and 4 to conclude Corollary 2 and 3 as follows.

Proof of Corollary 2. Let $\sigma_{\mathbf{A}_k}=\lambda_k(\mathbf{A})$ and $\sigma_{\mathbf{W}_k}=\sup_i\max_j\lambda_j(\mathbf{W}^{(i)}_k)$. By Lemma 3, we have $\max_j\lambda_j(\sum_k(\mathbf{W}^{(i)}_k\otimes\mathbf{A}_k))\leq\max_k\sigma_{\mathbf{A}_k}\sigma_{\mathbf{W}_k}$. Noting that $\mathbf{P}^{(i)}$ is diagonal with entries at most 1, we have $\max_j\lambda_j\big(\mathbf{P}^{(l)}\sum_{k_l}(\mathbf{W}^{(l)}_{k_l}\otimes\mathbf{A}_{k_l})\cdots\mathbf{P}^{(2)}\sum_{k_2}(\mathbf{W}^{(2)}_{k_2}\otimes\mathbf{A}_{k_2})\mathbf{P}^{(1)}\sum_{k_1}(\mathbf{W}^{(1)}_{k_1}\otimes\mathbf{A}_{k_1})\big)\leq(\max_k\sigma_{\mathbf{A}_k}\sigma_{\mathbf{W}_k})^l$. Therefore, if $\underline{\forall k=\{1,2,\ldots,n\}}$ $\sigma_{\mathbf{A}_k}\sigma_{\mathbf{W}_k}<1$, then $\lim_{l\to\infty}\max_j\lambda_j\big(\sum_k(\mathbf{W}^{(i)}_k\otimes\mathbf{A}_k)\big)=0$. Hence $\lim_{l\to\infty}\mathcal{I}(\mathbf{x};\mathbf{y}^{(l)})=0$ results by Lemma 2. □

Proof of Corollary 3. Let $\gamma_{\mathbf{W}_k}=\inf_i\min_j\lambda_j(\mathbf{W}^{(i)}_k)$. Note that $\min_j\lambda_j\big(\mathbf{P}^{(i)}\sum_k\mathbf{W}^{(i)}_k\otimes\mathbf{A}_k\big)\geq a\min_k\lambda_k(\mathbf{A})\gamma_{\mathbf{W}_k}$ by Lemma 3 and that $\min_j\lambda_j(\mathbf{P}^i)=a$. Moreover, $\min_j\lambda_j\big(\mathbf{P}^{(l)}\sum_{k_l}(\mathbf{W}^{(l)}_{k_l}\otimes\mathbf{A}_{k_l})\cdots\mathbf{P}^{(2)}\sum_{k_2}(\mathbf{W}^{(2)}_{k_2}\otimes\mathbf{A}_{k_2})\mathbf{P}^{(1)}\sum_{k_1}(\mathbf{W}^{(1)}_{k_1}\otimes\mathbf{A}_{k_1})\big)\geq(a\min_k\lambda_k(\mathbf{A})\gamma_{\mathbf{W}_k})^l$. Therefore, if $a\sigma_{\mathbf{A}_k}\gamma_{\mathbf{W}_k}\geq 1$, $\underline{\forall k\in\{1,2,\ldots,n\}}$, then $\mathcal{I}(\mathbf{x};\mathbf{y}^{(l)})=\mathcal{H}(\mathbf{x})$ $\forall l\in\mathbb{N}^+$ by Lemma 2, hence $\mathcal{L}(\mathbf{y}^{(l)})=0$. □