DECOUPLE GRAPH NEURAL NETWORKS: TRAIN MULTIPLE SIMPLE GNNS SIMULTANEOUSLY INSTEAD OF ONE

Anonymous authors
Paper under double-blind review

ABSTRACT

Graph neural networks (GNN) suffer from severe inefficiency due to the exponential growth of node dependency with the increase of layers. It extremely limits the application of stochastic optimization algorithms so that the training of GNN is usually time-consuming. To address this problem, we propose to decouple a multi-layer GNN as multiple simple modules for more efficient training, which is comprised of classical *forward training* (*FT*) and designed *backward training* (*BT*). Under the proposed framework, each module can be trained efficiently in FT by stochastic algorithms without distortion of graph information owing to its simplicity. To avoid the only unidirectional information delivery of FT and sufficiently train shallow modules with the deeper ones, we develop a backward training mechanism that makes the former modules perceive the latter modules, inspired by the classical backward propagation algorithm. The backward training introduces the reversed information delivery into the decoupled modules as well as the forward information delivery. To investigate how the decoupling and greedy training affect the representational capacity, we theoretically prove that the error produced by linear modules will not accumulate on unsupervised tasks in most cases. The theoretical and experimental results show that the proposed framework is highly efficient with reasonable performance, which may deserve more investigation.

1 INTRODUCTION

In recent years, neural networks (Simonyan & Zisserman, 2015; He et al., 2016), due to the impressive performance, have been extended to graph data, known as graph neural networks (GNN) (Scarselli et al., 2008). As GNNs significantly improve the results of graph tasks, it has been extensively investigated from different aspects, such as graph convolution network (GCN) (Kipf & Welling, 2017; Niepert et al., 2016), graph attention networks (GAT) (Velickovic et al., 2018; Kreuzer et al., 2021), spatial-temporal GNN (STGNN) (Wang et al., 2020), graph auto-encoder (Kipf & Welling, 2016; Park et al., 2019), graph contrastive learning (Hassani & Khasahmadi, 2020), *etc*.

Except for the variants that originate from different perspectives, an important topic is motivated by the well-known inefficiency of GNN. In classical neural networks (He et al., 2016), the optimization is usually based on stochastic algorithms with limited batch (Duchi et al., 2011; Kingma & Ba, 2015) since samples are independent of each other. However, the aggregation-like operations defined in (Hamilton et al., 2017) result in the dependency of each node on its neighbors and the amount of dependent nodes for one node increases exponentially with the growth of layers, which results in the unexpected increases of batch size. Some works are proposed based on neighbor sampling (Hamilton et al., 2017; Chen et al., 2018b;a; Zeng et al., 2020) and graph approximation (Chiang et al., 2019) to limit the batch size, while some methods (Wu et al., 2019; Zhu & Koniusz, 2021) attempt to directly apply high-order graph operation and sacrifice the most non-linearity. The training stability is a problem for neighbor sampling methods (Hamilton et al., 2017; Chen et al., 2018b; Zeng et al., 2020) though VRGCN (Chen et al., 2018a) has attempted to control the variance via improving sampling. Note that the required nodes may still grow (slowly) with the increase of depth. ClusterGCN (Chiang et al., 2019) finds an approximate graph with plenty of connected components so that the batch size is strictly upper-bounded. The major challenge of these methods is the information missing during sampling. The simplified methods (Wu et al., 2019; Zhu & Koniusz, 2021) are efficient but the limited

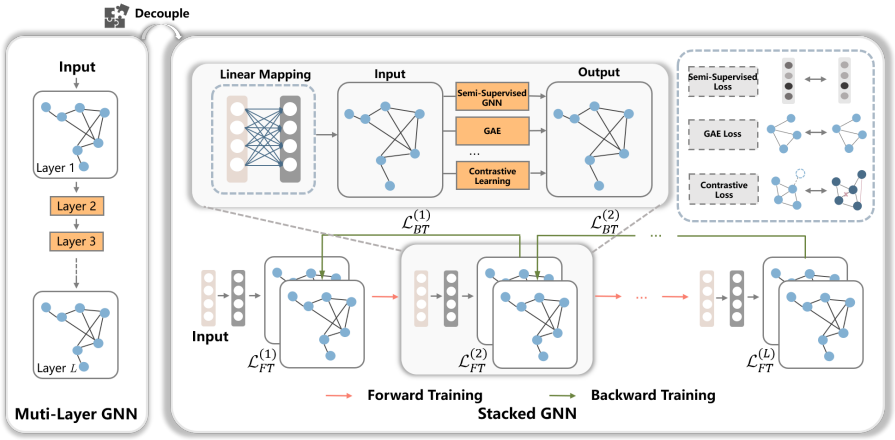


Figure 1: Illustration of a stacked graph neural network decoupled from an L-layer GNN. To train each module individually, some loss (*e.g.*, semi-supervised loss, unsupervised loss, contrastive loss) is required and it is denoted by $\mathcal{L}_{FT}^{(t)}$. To let the shallow modules perceive the deeper ones, $\mathcal{M}_t$ passes back the *expected* input features to $\mathcal{M}_{t-1}$ during the backward training. The divergence between the features output by $\mathcal{M}_t$ and the expected features of $\mathcal{M}_{t-1}$ formulates the BT loss $\mathcal{L}_{BT}^{(t)}$.

non-linearity may be the bottleneck of these methods. These methods may incorporate the idea of GIN (Xu et al., 2019) to improve the capacity (Zhu & Koniusz, 2021).

To apply stochastic optimization while retaining the exact graph structure, we propose a framework, namely stacked graph neural network (SGNN), which decouples a multi-layer GNN as multiple simple GNN modules and then trains them simultaneously rather than connecting them with the increase of the depth. Inspired by the backward propagation algorithm, we find that the main difference between stacked networks (Vincent et al., 2010) and classical networks is *no training information propagated from the latter modules to the former ones*. The lack of backward information delivery may be the main reason of the performance limitation of stacked models. The contributions are concluded as: **(1)** We accordingly propose a backward training strategy to let the former modules receive the information from the final loss and latter modules, which leads to a cycled training framework to control bias and train shallow modules correctly. **(2)** Under this framework, a multi-layer GNN can be decoupled into multiple simple GNNs, named as separable GNNs in this paper, so that every training step could use the stochastic optimization without any samplings or changes on graph. Therefore, SGNN could take both non-linearity and high efficiency into account. **(3)** We investigate how the decoupling and greedy training affect the representational capacity of the linear SGNN. It is proved that the error would not accumulate in most cases when the final objective is graph reconstruction.

2 Background

Graph Neural Networks Aiming to extend convolution operation into graph, graph convolution became a hot topic (Niepert et al., 2016; Bruna et al., 2014; Defferrard et al., 2016) and graph convolution network (GCN) (Kipf & Welling, 2017) has become an important baseline. By introducing self-attention techniques (Vaswani et al., 2017), graph attention networks (GAT) (Velickovic et al., 2018; Kreuzer et al., 2021) are proposed and applied to other applications (Kim et al., 2021; Guo et al., 2021). As Li et al. (2018) claimed that GNNs suffer from the over-smoothing problem, GALA (Park et al., 2019) develops the graph sharpening and ResGCN (Li et al., 2019) attempts to designs a deeper architecture. The theoretical works (Li et al., 2018; Oono & Suzuki, 2020; Cong et al., 2021) have different views towards the depth of GNNs. Some works (Li et al., 2018; Oono & Suzuki, 2020) claimed that the expressive power of GNN decreases with the increase of layers, while the others argue that the assumptions in (Oono & Suzuki, 2020) may not hold and deeper GNNs have stronger power (Cong et al., 2021). Moreover, some works (Morris et al., 2019; Xu et al., 2019) investigate the expressive capability by showing the connection between Weisfeiler-Lehman test (Weisfeiler & Leman, 1968) and GNNs. Nevertheless, *most of them neglect the inefficiency problem of GNNs*.

Efficient Graph Neural Networks To accelerate the optimization through batch gradient descent to GNN without too much deviation, several models (Hamilton et al., 2017; Chen et al., 2018b;a; Zeng et al., 2020) propose to sample data points according to graph topology. These models propose different sampling strategies to obtain stable results. GraphSAGE (Hamilton et al., 2017) produces a subgraph with limited neighbors for each node while FastGCN (Chen et al., 2018b) samples fixed nodes for each layer with the importance sampling. The variance of sampling is further controlled in (Chen et al., 2018a). ClusterGCN (Chiang et al., 2019) aims to generate an approximate graph with plenty of connected components so that each component can be used as a batch per step. SGC (Wu et al., 2019) simplifies GCN by setting all activations of middle layers as linear functions and SSGC (Zhu & Koniusz, 2021) further improves it. In summary, *SGNN proposed in this paper retains the non-linearity and requires no node sampling or sub-graph sampling.*

Connections to Existing Models Stacked Auto-Encoder (*SAE*) (Vincent et al., 2010) is a model applied to the pre-training of neural networks. It trains the current two-layer auto-encoder (Hinton & Salakhutdinov, 2006) and then feeds the latent features output by the middle layer to the next auto-encoder. The model is often used as a pre-training model instead of a formal model. MGAE (Wang et al., 2017) is an extension of SAE and its fundamental module is graph auto-encoder (Kipf & Welling, 2016). The main difference compared with the proposed model is whether each module could be perceived by modules from both forward and backward directions. The stack paradigm is similar to the classical boosting models (Freund & Schapire, 1997; Friedman, 2001; Chen & Guestrin) while some works (Schwenk & Bengio, 2000; Zhou et al., 2002) also investigated the boosting algorithm of neural networks. In recent years, some boosting GNN models (Ivanov & Prokhorenkova, 2021; Sun et al., 2021) are also developed. The most boosting algorithms (*e.g.*, (Schwenk & Bengio, 2000; Ivanov & Prokhorenkova, 2021)) aim to learn a prediction function gradually while the proposed SGNN aims to learn ideal embeddings gradually. Note that AdaGCN (Sun et al., 2021) is also trained gradually and the features are combined using AdaBoost (Freund & Schapire, 1997). More importantly, *all these boosting methods for GNNs are only trained forward and the backward training is missing*. Deep neural interface (Jaderberg et al., 2017) proposes to decouple neural networks to asynchronously accelerate the computation of gradients. *The decoupling is an acceleration trick to compute the gradients of L-layer networks*, while SGNN proposed in this paper explicitly separates an L-layer GNN into L simple modules. In other words, *the ultimate goal of SGNN is not to optimize an L-layer GNN*.

3 Proposed Method

Motivated by SAE (Vincent et al., 2010) and the fact that the simplified models (Wu et al., 2019; Zhu & Koniusz, 2021) are highly efficient for GNN, we therefore rethink the substantial difference between the stacked networks and multi-layer GNNs. To sum up, we attempt to answer the following two questions in this paper:

Q1: *How to decouple a complex GNN into multiple simple GNNs and train them simultaneously?*

Q2: *How does the decoupling affect the representational capacity and final performance?*

We will discuss the first question in this section and then elaborate on another one in Section 4.

3.1 Preliminary

Each decoupled GNN model of the proposed model is named as a module and the t-th module is denoted by $\mathcal{M}_t$ for simplicity. The vector and matrix are denoted by lower-case and upper-case letters in bold, respectively. $\|\cdot\|$ represents the Frobenius norm. Given a graph, let $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ be adjacency matrix and $\boldsymbol{X} \in \mathbb{R}^{n\times d}$ be node features. A typical GNN layer can be usually defined as

$$\boldsymbol{H} = f(\boldsymbol{A}, \boldsymbol{X}, \boldsymbol{W}) = \varphi(\boldsymbol{P}\boldsymbol{X}\boldsymbol{W}), \tag{1}$$

where $\boldsymbol{W}$ is projection coefficient and $\boldsymbol{P} = \phi(\boldsymbol{A})$ is a function of $\boldsymbol{A}$. When we discuss each individual module, we assume that $\boldsymbol{W} \in \mathbb{R}^{d\times k}$ for simplicity. For example, GCN (Kipf & Welling, 2017) defines $\boldsymbol{P}$ as $\boldsymbol{P}_{\text{GCN}} = \boldsymbol{D}^{-\frac{1}{2}}(\boldsymbol{A}+\boldsymbol{I})\boldsymbol{D}^{-\frac{1}{2}}$ and $\boldsymbol{D}$ is the degree matrix of $\boldsymbol{A}+\boldsymbol{I}$. When multiple layers are integrated, the learned representation given by multiple GNN layers can be written as

$$\boldsymbol{H} = f_1 \circ f_2 \circ \cdots \circ f_L(\boldsymbol{A}, \boldsymbol{X}, \boldsymbol{W}_1, \ldots, \boldsymbol{W}_L) = \varphi_L(\boldsymbol{P}\varphi_{L-1}(\cdots \varphi_1(\boldsymbol{P}\boldsymbol{X}\boldsymbol{W}_1)\cdots)\boldsymbol{W}_L), \tag{2}$$

where L is the amount of layers. Assume that the average number of neighbors is c. To compute $\boldsymbol{H}$, each sample will need $\mathcal{O}(c^L)$ extra samples. If the depth is large and the graph is connected, then all nodes have to be engaged to compute for one node. The uncontrolled batch size results in the time-consuming training. In vanilla GNNs, the computational complexity is $\mathcal{O}(KL\|\boldsymbol{A}\|_0 \sum_{i=0}^{m-1} d_i d_{i+1})$ on sparse graphs where K is the number of iterations and d_i is the dimension of $\boldsymbol{W}_i$. For large-scale datasets, both time and space complexity are too expensive.

3.2 Stacked Graph Neural Networks

Although the stacked networks usually have more parameters than multi-layer networks, which frequently indicates that the stacked networks may be more powerful, they only serve as a technique for pre-training. Specifically speaking, they simply transfer the representations learned by the current network to the next one but *no feedback is passed back*. It causes the invisibility of the succeeding modules and the final objective. As a result of the unreliability of the former modules, the stacked model is conventionally used as an unsupervised pre-training model.

Rethinking the learning process of a network, multiple layers are optimized simultaneously by gradient-based methods where the gradient is calculated by the well-known backward propagation algorithm (Rumelhart et al., 1986). The algorithm consists of forward propagation (FP) and backward propagation (BP). FP computes the required values for BP, which can be viewed as an information delivery process. Note that FP is similar to the training of the stacked networks. Specifically, transferring the output of the current module to the next one in the stacked network is like the computation of neurons layer by layer during FP. Inspired by this, we aim to design a BP-like training strategy, namely *backward training (BT)*, so that the former modules could be tuned according to the feedback. The core idea of our stacked graph neural network (SGNN) is shown in Figure 1.

Before introducing SGNN in detail, we formally introduce the key concept and core motivation of how to accelerate GNN via SGNN.

Definition 3.1. *If a GNN model can be formulated as* $f(\boldsymbol{A}, \boldsymbol{X}, \boldsymbol{W}) = f_1(f_0(\boldsymbol{A}, \boldsymbol{X}), \boldsymbol{W})$*, then it is a separable GNN. If it can be further formulated as* $f(\boldsymbol{A}, \boldsymbol{X}, \boldsymbol{W}) = f_1(f_0(\boldsymbol{A}, \boldsymbol{X}), \boldsymbol{W}) = g_1(\boldsymbol{A}, g_0(\boldsymbol{X}, \boldsymbol{W}))$*, then it is a fully-separable GNN.*

To keep simplicity, define the set of separable GNNs as $\mathcal{F}_k = \{f : \mathbb{R}^{n\times n} \times \mathbb{R}^{n\times d} \times \mathbb{R}^{d\times k} \mapsto \mathbb{R}^{n\times k} | f(\boldsymbol{A}, \boldsymbol{X}, \boldsymbol{W}) = f_1(f_0(\boldsymbol{A}, \boldsymbol{X}), \boldsymbol{W})\}$ and the set of fully-separable GNNs as $\mathcal{F}_k^* = \{f : \mathbb{R}^{n\times n} \times \mathbb{R}^{n\times d} \times \mathbb{R}^{d\times k} \mapsto \mathbb{R} | f(\boldsymbol{A}, \boldsymbol{X}, \boldsymbol{W}) = f_1(f_0(\boldsymbol{A}, \boldsymbol{X}), \boldsymbol{W}) = g_1(\boldsymbol{A}, g_0(\boldsymbol{X}, \boldsymbol{W}))\}$. Note that most single-layer GNN models are fully-separable. For instance, SGC (Wu et al., 2019) is fully-separable where $f_0(\boldsymbol{A}, \boldsymbol{X}) = \boldsymbol{P}^m \boldsymbol{X}$ and $f_1(f_0(\boldsymbol{A}, \boldsymbol{X}), \boldsymbol{W}) = \varphi(f_0(\boldsymbol{A}, \boldsymbol{X}) \cdot \boldsymbol{W})$, while the single-layer GIN (Scarselli et al., 2008) is separable but not fully-separable since $\boldsymbol{P} \cdot \mathrm{MLP}(\boldsymbol{X}) \neq \mathrm{MLP}(\boldsymbol{P}\boldsymbol{X})$ usually holds. However, a single-layer GAT (Velickovic et al., 2018) is not separable since the graph operation is relevant to $\boldsymbol{W}$.

The separable property actually factorizes a GNN model to 2 parts, graph operation f_0 and neural operation f_1. Since all dependencies among nodes in GNNs are caused by the graph operation, one can compute $\boldsymbol{X}' = f_0(\boldsymbol{A}, \boldsymbol{X})$ once (like **preprocessing**) in separable GNNs and then the GNN is converted into a typical network. After computing $\boldsymbol{X}'$, the information contained in graph has been passed into $\boldsymbol{X}'$ and the succeeding sampling would not affect the topology of graph. Therefore, we can obtain a highly efficient GNN model that can be optimized by SGD, provided that each module is separable. On the other hand, *the fully-separable condition is essential for the backward training to pass back the information over multiple modules*. Since most single-layer GNNs are separable but not fully-separable, we show how to revise separable GNNs to **introduce the fully-separability**.

Then, we introduce the core idea of SGNN by aiming to handle the two main problems.

Forward Training (FT) *The first challenge* is how to set the training objective for each module $\mathcal{M}_t$. It is crucial to apply SGNN to both supervised and unsupervised scenes. Suppose that we have a separable GNN module $\mathcal{M}$ and let $\boldsymbol{H} = f(\boldsymbol{A}, \boldsymbol{X}, \boldsymbol{W})$ be the features learned by the separable GNN module. For the unsupervised cases, if $\mathcal{M}_t$ is a GAE, then the loss of FT is formulated as

$$\mathcal{L}_{FT} = \ell(\boldsymbol{A}, \boldsymbol{X}, \boldsymbol{H}) = d(\boldsymbol{A}, \kappa(\boldsymbol{H})), \tag{3}$$

where $d(\cdot, \cdot)$ represents the metric function and $\kappa : \mathbb{R}^{n\times k} \mapsto \mathbb{R}^{n\times n}$ is a mapping function. For instance, a simple loss introduced by (Kipf & Welling, 2016) is $d(\boldsymbol{A}, \kappa(H)) = KL(\boldsymbol{A} \| \sigma(\boldsymbol{H}\boldsymbol{H}^T))$

where $\sigma(\cdot)$ is the sigmoid function, and $KL(\cdot\|\cdot)$ is the Kullback-Leibler divergence. The other options include but not limited to symmetric content reconstruction (Park et al., 2019) and graph contrastive learning (Hassani & Khasahmadi, 2020). For modules with supervision information, a projection matrix $\boldsymbol{R} \in \mathbb{R}^{k\times c}$ is introduced to map the k-dimension embedding vector into soft labels with c classes. For the node classification, the loss can be simply set as

$$\mathcal{L}_{FT} = KL(\boldsymbol{Y}\|\text{softmax}(\boldsymbol{H}\boldsymbol{R})), \tag{4}$$

where $\boldsymbol{Y} \in \mathbb{R}^{n\times c}$ is the supervision information for supervised tasks. Note that the above loss is equivalent to the classical softmax regression if $\boldsymbol{H}$ is constant. The loss could also be link prediction, graph classification, *etc*. Although base modules can utilize diverse losses, we only discuss the situation that all modules use the same kind of loss in this paper for simplicity.

Algorithm 1: Procedure of Stacked Graph Neural Networks Composed of L Modules

Input : Adjacency matrix $\boldsymbol{A}$, feature matrix $\boldsymbol{X}$, balance coefficient η, the number of epochs K, L separable GNN modules $\{\mathcal{M}_t\}_{t=1}^{L}$, $\boldsymbol{H}_0 \leftarrow \boldsymbol{X}$.
Output : Features output by $\mathcal{M}_L$.
for $i = 1, \ldots, K$ **do**
  # Forward Train Stacked Graph Neural Networks
  for $t = 1, 2, \ldots, L-1$ **do**
    Feed the current features to $\mathcal{M}_t$ and reset $\boldsymbol{U}_t$: $\boldsymbol{X}_t \leftarrow \boldsymbol{H}_{t-1}, \boldsymbol{U}_t \leftarrow \boldsymbol{I}$.
    # Train with only $\mathcal{L}_{FT}$ at the first forward training
    $\boldsymbol{X}_t{}' = f_0^{(t)}(\boldsymbol{A}, \boldsymbol{X}_t)$ *# Preprocessing for mini-batch algorithms*
    Compute loss $\mathcal{L}^{(t)} \leftarrow \mathcal{L}_{FT}^{(t)}$ if $i == 1$ else $\mathcal{L}_{FT}^{(t)} + \eta\mathcal{L}_{BT}^{(t)}$
    Train $\mathcal{M}_t$ by optimizing $\min_{\boldsymbol{W}_t} \mathcal{L}^{(t)}$ based on mini-batch algorithms.
    Obtain the features: $\boldsymbol{H}_t \leftarrow f_1^{(t)}(\boldsymbol{X}_t{}', \boldsymbol{W}_t)$.
  $\boldsymbol{X}_L \leftarrow \boldsymbol{H}_t, \boldsymbol{U}_L \leftarrow \boldsymbol{I}, \boldsymbol{X}_L' = f_0^{(L)}(\boldsymbol{A}, X_L)$.
  Train $\mathcal{M}_L$ by optimizing $\min_{\boldsymbol{W}_L, \boldsymbol{U}_L} \mathcal{L}_{FT}^{(L)}$ based on mini-batch algorithms.
  # Backward Train Stacked Graph Neural Networks
  for $t = L-1, L-2, \ldots, 1$ **do**
    Compute the expected output feature of $\mathcal{M}_t$: $\boldsymbol{Z}_{t+1} \leftarrow (g_0^*)_{t+1}(\boldsymbol{X}_{t+1}, \boldsymbol{U}_{t+1})$.
    Train $\mathcal{M}_t$ by optimizing $\min_{\boldsymbol{W}_t, \boldsymbol{U}_t} \mathcal{L}_{FT}^{(t)} + \eta\mathcal{L}_{BT}^{(t)}$ based on mini-batch algorithms.

Backward Training (BT) *The second challenge* is how to train multiple separable GNNs simultaneously in order to ensure performance. Roughly speaking, the gradients of all layers in neural networks are computed exactly due to the repeated delivery of information by FP and BP. *BP lets the shallow layers perceive the deep ones through the feedback, while the tail modules are invisible to the head ones in FT.* We accordingly design the backward training (**BT**) for SGNN. To achieve the reverse information delivery, a separable GNN layer is modified by **introducing the fully-separability**,

$$\boldsymbol{H} = \hat{f}(\boldsymbol{A}, \boldsymbol{X}, \boldsymbol{W}) = \hat{f}_1(\hat{f}_0(\boldsymbol{A}, \boldsymbol{X}), \boldsymbol{W}) = \hat{f}_1(f^*(\boldsymbol{A}, \boldsymbol{X}, \boldsymbol{U}), \boldsymbol{W}),\ \forall \hat{f} \in \mathcal{F}_k, f^* \in \mathcal{F}_d^*, \tag{5}$$

where $\boldsymbol{U} \in \mathbb{R}^{d\times d}$ and $\boldsymbol{W} \in \mathbb{R}^{d\times k}$. Note that $f^*(\boldsymbol{A}, \boldsymbol{X}, \boldsymbol{U}) = g_1^*(\boldsymbol{A}, g_0^*(\boldsymbol{X}, \boldsymbol{U}))$. Clearly, if $\boldsymbol{U}$ is fixed as a constant, then the modified layer is equivalent to the original separable GNN layer $\hat{f}$. Denote $\boldsymbol{Z} = g_0^*(\boldsymbol{X}, \boldsymbol{U})$ and $\boldsymbol{Z}$ is the *expected features*. Specifically, $\boldsymbol{Z}_t$ is the learned feature during the backward training of $\mathcal{M}_t$, and it is also the expected input of $\mathcal{M}_t$, *i.e.*, from $\mathcal{M}_{t-1}$. In the forward training, the delivery of information is based on the learned features $\boldsymbol{H}_t$, and $\boldsymbol{Z}$ plays the similar role in the backward training. The loss of backward training attempts to shrink the difference between the output feature $\boldsymbol{H}_t$ of $\mathcal{M}_t$ and expected input $\boldsymbol{Z}_{t+1}$ of $\mathcal{M}_{t+1}$,

$$\mathcal{L}_{BT}^{(t)} = d(\boldsymbol{H}_t, \boldsymbol{Z}_{t+1}) = d(f_t^*(\boldsymbol{A}, \boldsymbol{X}_t, \boldsymbol{U}_t), \boldsymbol{Z}_{t+1}). \tag{6}$$

Note that $\mathcal{L}_{BT}$ is only activated after the first forward training leading to the final loss of $\mathcal{M}_t$ as

$$\mathcal{L}^{(t)} = \mathcal{L}_{FT}^{(t)} + \eta\mathcal{L}_{BT}^{(t)}, \tag{7}$$

and it is updated during each backward training. The introduction of $\boldsymbol{Z}$ will not limit the application of stochastic optimization since the expected features can also be sampled at each iteration without restrictions. The procedure is summarized as Algorithm 1.

(a) GCN on Cora (b) SGNN on Cora (c) GCN on Citeseer (d) SGNN on Citeseer

Figure 2: Visualization of SGNN comprised of 3 modules and 3-layer GCN on node classification. For SGNN, the output of $\mathcal{M}_3$ is visualized. For GCN, the output of the final GCN-layer is visualized.

Remark that $\boldsymbol{U}_t$ remains as the identity matrix during FT. This setting leads to each forward computation across L base modules being equivalent to a forward propagation L-layer GNN. In other words, an SGNN with L modules can be regarded as a decomposition of an L-layer GNN. One may concern that why not to learn $\boldsymbol{U}_t$ and $\boldsymbol{W}_t$ together in FT. In this case, we prefer to use $\boldsymbol{U}_t$ only for learning the expected features of $\mathcal{M}_{t+1}$ and the capability improvement from the co-learning $\boldsymbol{U}_t$ in FT could also be implemented by $\boldsymbol{W}$, which is equivalent to use GIN (Xu et al., 2019) as base modules. The complexity can be found in Appendix D due to the limitation of space.

4 Theoretical Analysis

To answer the second question raised in the beginning of Section 3, we discuss the impact of the decoupling in this section. Intuitively speaking, if an L-layer GNN achieves satisfactory results, then there exists $\{\boldsymbol{W}_t\}_{t=1}^{L}$ such that an SGNN with L modules could achieve the same results. However, each $\mathcal{M}_t$ is trained greedily according to the forward training loss $\mathcal{L}_{FT}$, while middle layers of a multi-layer GNN are trained according to the same objective. The major concern is whether the embedding learned by a greedy strategy leads to an irreversible deviation in the forward training.

In this section, we investigate the possible side effects on a specific SGNN comprised of unsupervised modules defined in Eq. (3) with linear activations. The conclusion is not apparent since simply setting $\boldsymbol{W}$ as an identity matrix does not prove it for GNN due to the existence of $\boldsymbol{P}$. Remark that a basic premise is that the previous module has achieved a reasonable result.

Given a linear separable-GNN module $\mathcal{M}$ which is defined as $\boldsymbol{H} = \boldsymbol{P}\boldsymbol{X}\boldsymbol{W} \in \mathbb{R}^{n\times k}$, suppose that the forward training uses the reconstruction error $\|\boldsymbol{P} - \boldsymbol{H}\boldsymbol{H}^T\|$ as $\mathcal{L}_{FT}$. We first introduce the matrix angle to better understand whether the preconditions of Theorem 4.1 are practicable.

Definition 4.1. *Given two matrices $\boldsymbol{B}_1, \boldsymbol{B}_2 \in \mathbb{R}^{n\times n}$, we define the matrix angle as $\theta(\boldsymbol{B}_1, \boldsymbol{B}_2) = \arccos\langle \boldsymbol{B}_1, \boldsymbol{B}_2\rangle / (\|\boldsymbol{B}_1\| \cdot \|\boldsymbol{B}_2\|)$.*

Before elaborating on theorems, we introduce the following assumption, which separates the discussions into two cases.

Assumption 4.1. *$\boldsymbol{X}\boldsymbol{X}^T$ does not share the same eigenspace with $\boldsymbol{P} - \boldsymbol{X}\boldsymbol{X}^T$.*

Note that the above assumption is weak and frequently holds in most cases. For simplicity, $\boldsymbol{U}_r \in \mathbb{R}^{n\times r}$ ($r \in \mathbb{N}_+$) is the eigenvectors associated with r leading eigenvalues. Under this assumption, we find that the error of $\mathcal{M}$ is upper-bounded by ε.

Theorem 4.1. *Let $\delta = 1 - \cos(\theta_*/2)$ and $\theta_* = \theta((\boldsymbol{P} - \boldsymbol{X}\boldsymbol{X}^T)\boldsymbol{Q}, \boldsymbol{Q}(\boldsymbol{P} - \boldsymbol{X}\boldsymbol{X}^T))$ where $\boldsymbol{Q} = \boldsymbol{U}_o\boldsymbol{U}_o^T - \boldsymbol{I}/2$ and $o = \min(\mathrm{rank}(\boldsymbol{X}), k)$. Under Assumption 4.1, if $\|\boldsymbol{P} - \boldsymbol{H}\boldsymbol{H}^T\| = \varepsilon \le \mathcal{O}(\delta)$ and $\sigma_* \le \mathcal{O}(\sqrt{\delta})$ where σ_* is the $(o+1)$-th largest singular value of $\boldsymbol{X}$, then there exists $\boldsymbol{W} \in \mathbb{R}^{d\times k}$ so that $\|\boldsymbol{P} - \boldsymbol{H}\boldsymbol{H}^T\| \le \varepsilon$. In other words, if ε is small enough, then $\boldsymbol{H}\boldsymbol{H}^T$ could be a better approximation than $\boldsymbol{X}\boldsymbol{X}^T$.*

Specially, if $\mathrm{rank}(\boldsymbol{X}) \le k$ or $k = d$, $\sigma_* = 0$ so that $\sigma_* \le \mathcal{O}(\sqrt{\delta})$ holds. From the above theorem, we claim that *the error through $\mathcal{M}$ will not accumulate (i.e., bound by ε)* provided that the input $\boldsymbol{X}$, the output of the previous modules, is well-trained. We further provide an upper-bound of error if Assumption 4.1 does not hold. The following theorem shows the increasing speed of error is at most linear with the tail singular-values.

Theorem 4.2. *If Assumption 4.1 does not hold, then there exists $\boldsymbol{W} \in \mathbb{R}^{d\times k}$ so that $\|\boldsymbol{P} - \boldsymbol{H}\boldsymbol{H}^T\| \le \varepsilon + \mathcal{O}(\sigma_*^2)$.*

Table 1: Node clustering results

Datasets	Cora		Citeseer		PubMed		Reddit	
	ACC	NMI	ACC	NMI	ACC	NMI	ACC	NMI
K-Means	0.4922	0.3210	0.5401	0.3054	0.5952	0.2780	0.1927	0.2349
ARGA	0.6400	0.4490	0.5730	0.3500	0.6807	0.2757	N/A	N/A
MGAE	0.6806	0.4892	0.6691	0.4158	0.5932	0.2957	N/A	N/A
GraphSAGE	0.6163	0.4826	0.5664	0.3425	0.5554	0.0943	0.6225	0.7291
FastGAE	0.3527	0.1553	0.2672	0.1178	0.4262	0.0442	0.1115	0.0715
ClusterGAE	0.4579	0.2261	0.4182	0.1767	0.3913	0.0001	N/A	N/A
GAE	0.5960	0.4290	0.4080	0.1760	0.6861	0.2957	N/A	N/A
AGC (SGC)	0.6892	0.5368	0.6700	0.4113	0.6978	0.3159	0.5833	0.6894
SGNN-FT	0.6278	0.5075	0.6141	0.3776	0.6444	0.2312	0.5943	0.7156
SGNN-BT	**0.7463**	**0.5546**	0.6730	0.4159	0.6951	**0.3337**	**0.7042**	**0.7601**
S^2GC	0.6960	0.5471	**0.6911**	**0.4287**	**0.7098**	0.3321	0.7011	0.7509
GAE-S^2GC	0.6976	0.5317	0.6435	0.3969	0.6528	0.2452	0.6272	0.7158
SGNN-S^2GC	0.7223	0.5404	0.6822	0.4243	0.7084	0.3302	0.7023	0.7575

Corollary 4.1. *Given an SGNN with L linear modules $\{\mathcal{M}_t\}_{t=1}^{L}$ with $\mathcal{L}_{FT}^{(t)} = \|\boldsymbol{P} - \boldsymbol{H}_t\boldsymbol{H}_t^T\|$, if $\boldsymbol{P} - \boldsymbol{H}_1\boldsymbol{H}_1^T$ and $\boldsymbol{H}_1\boldsymbol{H}_1^T$ share the same eigenspace, then $\mathcal{L}_{FT}^{(L)} \leq \mathcal{L}_{FT}^{(1)} + \sum_{t=1}^{L-1} \mathcal{O}(\sigma_*^2(\boldsymbol{H}_t))$.*

Based on Theorem 4.2, we conclude that the residual would not accumulate rapidly when Assumption 4.1 does not hold. All proofs and more discussions are put in Appendix A-C.

5 Experiments

In this section, we conduct experiments to investigate *whether the performance of SGNN could approach the performance of the original L-layer GNN in a highly-efficient way* and *what the impact of the non-linearity and flexibility brought by the decoupling is*. To sufficiently answer the above 2 problems, both node clustering and semi-supervised node classification are used. Due to the limitation of space, only 4 common datasets, including Cora, Citeseer, Pubmed (Sen et al., 2008), and Reddit (Hamilton et al., 2017). More experiments (*e.g.*, on OGB datasets) can be found in Appendix E.

5.1 Node Clustering

Experimental Settings We first testify the effectiveness of SGNN on the node clustering. We compare our method against 10 methods, including a baseline clustering model K-means, three GCN models without considering training efficiency (GAE (Kipf & Welling, 2016), ARGA (Pan et al., 2018), MGAE (Wang et al., 2017)), and six fast GCN models with GAE-loss (GraphSAGE (Hamilton et al., 2017), FastGAE (Chen et al., 2018b), ClusterGAE (Chiang et al., 2019), AGC (Zhang et al., 2019) (an unsupervised extension of SGC (Wu et al., 2019)), S^2GC (Zhu & Koniusz, 2021), and GAE-S^2GC).

The used codes are based on the publicly released implementation. To ensure fairness, all multi-layer GNN models consist of two layers and SGNN is comprised of two modules. For models that could be trained by stochastic algorithms, the size of mini-batch is set as 128. The learning rate is set as 0.001 and the number of epochs is set as 100. We set the size first layer as 128 and the second layer size as 64. In particular, when training every module of SGNN, 20% entries of the adjacency matrix are ignored. $\boldsymbol{U}$ is initialized as an identity matrix, and η is set as 10^3 by default. The number of backward training is set as 5 or 10. To investigate the effectiveness of backward training, we report the experimental results with sufficient training for each module, which is denoted by SGNN-FT, while SGNN with the proposed backward training is represented by SGNN-BT. To study the performance of SGNN with different base models, we choose a fully-separable GNN, S^2GC, as the base model and this method is marked as SGNN-S^2GC. As S^2GC does not use the GAE framework, we also added a competitor, namely GAE-S^2GC, which uses S^2GC as the encoder, to ensure fairness. All methods are run five times and the average scores are recorded. The result are summarized in Table 1.

Table 2: Test accuracy (%) averaged over 10 runs on 3 citation datasets

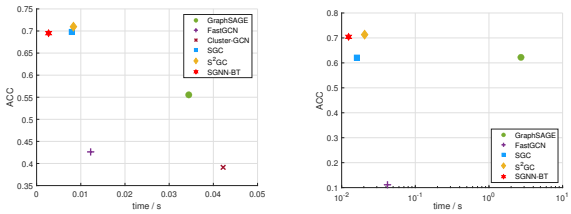
Datasets	Cora	Citeseer	Pubmed
GAT	83.0 ± 0.7	72.5 ± 0.7	79.0 ± 0.3
DGI	82.3 ± 0.6	71.8 ± 0.7	76.8 ± 0.6
GCNII	$\mathbf{85.5 \pm 0.5}$	$\underline{73.4 \pm 0.6}$	$\mathbf{80.2 \pm 0.4}$
FastGCN†	79.8 ± 0.3	68.8 ± 0.6	77.4 ± 0.3
GraphSAGE†	77.4 ± 0.6	66.7 ± 0.3	79.0 ± 0.3
Cluster-GCN†	65.3 ± 3.9	57.7 ± 2.3	71.5 ± 1.9
GCN	81.4 ± 0.4	70.9 ± 0.5	79.0 ± 0.4
SGC	81.0 ± 0.0	71.9 ± 0.1	78.9 ± 0.0
SGNN-BT	81.4 ± 0.5	70.2 ± 0.4	77.8 ± 0.3
S^{2}GC	83.5 ± 0.0	$\mathbf{73.6 \pm 0.1}$	$\mathbf{80.2 \pm 0.0}$
SGNN-S^{2}GC	$\underline{83.8 \pm 0.2}$	73.2 ± 0.3	$\mathbf{80.2 \pm 0.4}$

Table 3: Test accuracy (%) averaged over 5 runs on Reddit

Methods	Reddit
GAT	N/A
DGI	94.0
SAGE-mean	95.0
SAGE-GCN	93.0
FastGCN	93.7
Cluster-GCN	**96.6**
GCN	N/A
SGC	94.9
SGNN-BT	95.10 ± 0.02
S^{2}GC	<u>95.3</u>
SGNN-S^{2}GC	95.28 ± 0.03

Performance From Table 1, we find that SGNN outperforms in most datasets. If the released codes could not run on Reddit due to out-of-memory (OOM), we put the notation "N/A" instead of results. In particular, SGNN-BT obtains good improvements on Reddit with high efficiency. Specifically speaking, it is about 8% higher than the well-known GraphSAGE. SGNN-FT performs above the average on some datasets. It usually outperforms GraphSAGE but fails to exceed SGC. Due to the deeper structure caused by multiple modules, the performance of SGNN excels the simple GAE. It also outperforms SGC due to more non-linearity brought by multiple modules. Note that S^{2}GC and SGC are strong competitors, while SGNN can easily employ them as a base module since they are separable. From the ablation experiments, SGNN-BT works better than SGNN-FT, which indicates the necessity of the backward training. We also investigate how the number of modules L affects the node clustering accuracy and the results averaged over 5 runs are reported in Appendix E.

Efficiency Figure 3 shows the consuming time of several GNNs with higher efficiency on Pubmed and Reddit. Instead of neglecting the preprocessing operation, we measure the efficiency through a more rational way. We record the totally consuming time after loading data into RAM and then divide the total number of updating parameters of GNNs. The measurement could reflect the real difference of diverse training techniques aiming to apply batch-based algorithms to GNN. It should be emphasized the reason why SGC is worse than SGNN regarding the consuming time. The key point is the different costs of their *preprocessing* operation. For an L-order SGC, the computation cost of $\boldsymbol{P}^L\boldsymbol{X}$ is at least $\mathcal{O}(\|\boldsymbol{A}\|_0 Ld)$ while SGNN with L first-order modules totally requires $\mathcal{O}(\|\boldsymbol{A}\|_0 \sum_{i=1}^{L} d_i)$ for the same preprocessing operation. The metric also provides a fair comparison between SGC and other models since the stopping criteria are always different.



(a) Pubmed (b) Reddit

Figure 3: Performance and training efficiency of several scalable GNNs. The efficiency metric is computed by "Consuming Time / # Iterations".

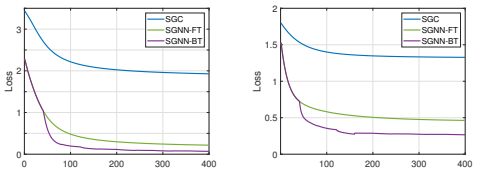
5.2 Node Classification

Experimental Setting We also conduct experiments of semi-supervised classification on four datasets. The split of datasets following (Wu et al., 2019) could be found in Appendix E. We compare SGNN against GCN (Kipf & Welling, 2017), GAT (Velickovic et al., 2018), DGI (Velickovic et al., 2019), FastGCN (Chen et al., 2018b), GraphSAGE (Hamilton et al., 2017), Cluster-GCN (Chiang et al., 2019), SGC (Wu et al., 2019), GCNII (Chen et al., 2020), and S^{2}GC (Zhu & Koniusz, 2021). Similarly, we testify SGNN with two different base models, namely SGNN-BT and SGNN-S^{2}GC. For GraphSAGE, we use the mean operator by default and some notations are added if the extra

operators are used. On citation networks, the learning rate is set as 0.01, while it is 10^{-4} on Reddit. Since the nodes for training are less than 200 on citation networks, we use all training points in each iteration for all methods while we sample 256 points as a mini-batch for approaching expected features during backward training of SGNN. On Reddit, the batch size of all batch-based models is set as 512. We do not apply the early stopping criterion used in (Kipf & Welling, 2017) and the max iteration follows the setting of SGC. The embedding dimensions of each module are the same as the setting in node clustering. For the sake of fairness, we report the results obtained by SGNN with two modules using first-order operation. The forward training loss is defined in Eq. (4). Moreover, all compared models share an identical implementation of their mini-batch iterators, loss function and neighborhood sampler (when applicable). The balance coefficient of $\mathcal{L}_{FT}$ and $\mathcal{L}_{BT}$ is set as 1 by default. We report the results averaged over 10 runs on citation datasets and 5 runs on Reddit in Table 2 and Table 3. The hyper-parameters are shared for different datasets which are optimized on Cora.

Performance The results of compared methods in Table 2 are taken from the corresponding papers. When the experimental results are missed, we run the publicly released codes and the corresponding records are superscripted by †. From Tables 2 and 3, we conclude that SGNN outperforms the models with neighbor sampling such as GraphSAGE, FastGCN, and ClusterGCN on citation networks and the performance of SGNN exceeds most models on Reddit. On simple citation networks, SGNN loses the least accuracy compared with other batch-based models, which is close to GCN. Owing to the separability of each module, the batch sampling requires no neighbor sampling and causes no loss of graph information. Note that we simply employ the single-layer GCN as the base modules in our experiments, while some high-order methods that obtain competitive results are also available for SGNN. Although some methods achieve preferable results, they either fail to run or obtain unsatisfactory results on large-scale datasets.

Visualization to Show Impact of the Decoupling In Figure 2, we visualize the output of a 3-module SGNN and a 3-layer GCN to directly show that *the decoupling would not cause the trivial features*, which corresponds to the theoretical conclusion in Section 4. To show the benefit of the non-linearity brought by SGNN and the backward training, the convergence curves of SGC, SGNN-FT, and SGNN-BT are shown in Figure 4. Note that the figure shows the variation of the *final loss*. In SGNN, the final loss is the loss of $\mathcal{M}_L$, while it is the unique training loss in SGC. SGC with L-order graph operation is used. From this figure, we can conclude that: (1) The non-linearity does lead to a better loss value; (2) The backward training significantly decreases the loss. More visualization and discussions could be found in Appendix E. In summary, *the decoupling empirically does not cause the negative impact.*



(a) Cora (b) Pubmed

Figure 4: Convergence curve of the *final loss*. The order of SGC is set as L. The backward training significantly decreases the final loss and the non-linearity also plays an important role.

6 Conclusion and Future Works

In this paper, we propose the Stacked Graph Neural Networks (SGNN). We first decouple a multi-layer GNN into multiple simple GNNs, which is formally defined as separable GNNs in our paper to ensure the availability of batch-based optimization without loss of graph information. The bottleneck of the existing stacked models is that the information delivery is only unidirectional, and therefore a backward training mechanism is developed to make the former modules perceive the latter ones. We also theoretically prove that the residual of linear SGNN would not accumulate in most cases for unsupervised graph tasks. The theoretical and experimental results show that the proposed framework is more than an efficient method and it may deserve further investigation in the future. The theoretical analysis focuses on linear SGNN and the generalization bound is also not investigated in this paper. Therefore, they will be the core of our future work. Moreover, as $\mathcal{L}_{FT}$ could be any losses, how to choose the most appropriate loss for each module will be also a crucial topic in our future works.

REFERENCES

Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. In *2nd International Conference on Learning Representations*, 2014.

Jianfei Chen, Jun Zhu, and Le Song. Stochastic training of graph convolutional networks with variance reduction. In *International Conference on Machine Learning*, pp. 942–950, 2018a.

Jie Chen, Tengfei Ma, and Cao Xiao. Fastgcn: Fast learning with graph convolutional networks via importance sampling. *arXiv preprint arXiv:1801.10247*, 2018b.

Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119, pp. 1725–1735, 2020.

Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In Balaji Krishnapuram, Mohak Shah, Alexander J. Smola, Charu C. Aggarwal, Dou Shen, and Rajeev Rastogi (eds.), *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pp. 785–794.

Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 257–266, 2019.

Weilin Cong, Morteza Ramezani, and Mehrdad Mahdavi. On provable benefits of depth in training graph convolutional networks. *Advances in Neural Information Processing Systems*, 34, 2021.

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, 2016.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011.

Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. Syst. Sci.*, 55(1):119–139, 1997.

Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pp. 1189–1232, 2001.

Dongyan Guo, Yanyan Shao, Ying Cui, Zhenhua Wang, Liyan Zhang, and Chunhua Shen. Graph attention tracking. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 9543–9552, 2021.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pp. 1024–1034, 2017.

Kaveh Hassani and Amir Hosein Khasahmadi. Contrastive multi-view representation learning on graphs. In *International Conference on Machine Learning*, pp. 4116–4126. PMLR, 2020.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.

G. E Hinton and R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

Sergei Ivanov and Liudmila Prokhorenkova. Boost then convolve: Gradient boosting meets graph neural networks. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

Max Jaderberg, Wojciech Marian Czarnecki, Simon Osindero, Oriol Vinyals, Alex Graves, David Silver, and Koray Kavukcuoglu. Decoupled neural interfaces using synthetic gradients. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70, pp. 1627–1635, 2017.

Byung-Hoon Kim, Jong Chul Ye, and Jae-Jin Kim. Learning dynamic graph representation of brain connectome with spatio-temporal attention. In *Advances in Neural Information Processing Systems*, volume 34, pp. 4314–4327. Curran Associates, Inc., 2021.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun (eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

Thomas N Kipf and Max Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

Devin Kreuzer, Dominique Beaini, Will Hamilton, Vincent Létourneau, and Prudencio Tossou. Rethinking graph transformers with spectral attention. In *Advances in Neural Information Processing Systems*, volume 34, pp. 21618–21629, 2021.

Guohao Li, Matthias Muller, Ali Thabet, and Bernard Ghanem. Deepgcns: Can gcns go as deep as cnns? In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 9267–9276, 2019.

Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 4602–4609, 2019.

Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In *International Conference on Machine Learning*, pp. 2014–2023, 2016.

Kenta Oono and Taiji Suzuki. Graph neural networks exponentially lose expressive power for node classification. In *International Conference on Learning Representations*, 2020.

Shirui Pan, Ruiqi Hu, Guodong Long, Jing Jiang, Lina Yao, and Chengqi Zhang. Adversarially regularized graph autoencoder for graph embedding. In *IJCAI*, pp. 2609–2615, 2018.

Jiwoong Park, Minsik Lee, Hyung Jin Chang, Kyuewang Lee, and Jin Young Choi. Symmetric graph convolutional autoencoder for unsupervised graph representation learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 6519–6528, 2019.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2008.

Holger Schwenk and Yoshua Bengio. Boosting neural networks. *Neural Comput.*, 12(8):1869–1887, 2000.

Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Gallagher, and Tina Eliassi-Rad. Collective classification in network data. *AI Mag.*, 29(3):93–106, 2008.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

Ke Sun, Zhanxing Zhu, and Zhouchen Lin. Adagcn: Adaboosting graph convolutional networks into deep models. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017*, pp. 5998–6008, 2017.

Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *6th International Conference on Learning Representations*, 2018.

Petar Velickovic, William Fedus, William L. Hamilton, Pietro Liò, Yoshua Bengio, and R. Devon Hjelm. Deep graph infomax. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019.

Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, 11:3371–3408, 2010.

Chun Wang, Shirui Pan, Guodong Long, Xingquan Zhu, and Jing Jiang. Mgae: Marginalized graph autoencoder for graph clustering. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pp. 889–898, 2017.

Xiaoyang Wang, Yao Ma, Yiqi Wang, Wei Jin, Xin Wang, Jiliang Tang, Caiyan Jia, and Jian Yu. Traffic flow prediction via spatial temporal graph neural network. In *Proceedings of The Web Conference 2020*, pp. 1082–1092, 2020.

Boris Weisfeiler and Andrei Leman. The reduction of a graph to canonical form and the algebra which appears therein. *Nauchno-Technicheskaya Informatsia*, 2(9):12–16, 1968.

Felix Wu, Amauri H. Souza Jr., Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Q. Weinberger. Simplifying graph convolutional networks. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97, pp. 6861–6871, 2019.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019.

Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor K. Prasanna. Graphsaint: Graph sampling based inductive learning method. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

Xiaotong Zhang, Han Liu, Qimai Li, and Xiao-Ming Wu. Attributed graph clustering via adaptive graph convolution. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pp. 4327–4333, 2019.

Zhi-Hua Zhou, Jianxin Wu, and Wei Tang. Ensembling neural networks: Many could be better than all. *Artif. Intell.*, 137(1-2):239–263, 2002.

Hao Zhu and Piotr Koniusz. Simple spectral graph convolution. In *9th International Conference on Learning Representations*, 2021.

A LEMMA FOR PROOFS

Lemma A.1. *For two given symmetric matrices $\boldsymbol{A}$ and $\boldsymbol{B}$, $\boldsymbol{A}$ and $\boldsymbol{B}$ share the same eigenspace if and only if $\boldsymbol{A}$ and $\boldsymbol{B}$ commute.*

Proof. First, if $\boldsymbol{A}$ and $\boldsymbol{B}$ share the same eigenspace, then there exists $\boldsymbol{P}$ such that

$$\boldsymbol{A} = \boldsymbol{P}\boldsymbol{\Lambda}_A\boldsymbol{P}^T, \boldsymbol{B} = \boldsymbol{P}\boldsymbol{\Lambda}_B\boldsymbol{P}^T. \tag{A.1}$$

Accordingly, we have $\boldsymbol{A}\boldsymbol{B} = \boldsymbol{P}\boldsymbol{\Lambda}_A\boldsymbol{\Lambda}_B\boldsymbol{P}^T = \boldsymbol{B}\boldsymbol{A}$.

Then, we turn to prove the converse. If $\boldsymbol{A}$ and $\boldsymbol{B}$ commute, suppose that $\boldsymbol{A}\boldsymbol{u} = \lambda\boldsymbol{u}$ and then

$$\boldsymbol{A}\boldsymbol{B}\boldsymbol{u} = \boldsymbol{B}\boldsymbol{A}\boldsymbol{u} = \lambda\boldsymbol{B}\boldsymbol{u}. \tag{A.2}$$

Apply eigenvalue decomposition, we have $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^T$. Note that if $\boldsymbol{A}\boldsymbol{u}_1 = \lambda_1\boldsymbol{u}_1$, $\boldsymbol{A}\boldsymbol{u}_2 = \lambda_2\boldsymbol{u}_2$, and $\lambda_1 \neq \lambda_2$, then $\boldsymbol{u}_2^T\boldsymbol{B}\boldsymbol{u}_1 = 0$ since $\boldsymbol{B}\boldsymbol{u}_1$ is also an eigenvector associated with λ_1. Therefore, $\boldsymbol{U}^T\boldsymbol{B}\boldsymbol{U}$ is a block-diagonal matrix, *i.e.*,

$$\boldsymbol{U}^T\boldsymbol{B}\boldsymbol{U} = \begin{bmatrix} \boldsymbol{C}_1 & & & \\ & \boldsymbol{C}_2 & & \\ & & \ddots & \\ & & & \boldsymbol{C}_k \end{bmatrix}. \tag{A.3}$$

Apply eigendecomposition to $\boldsymbol{C}_i$,

$$\boldsymbol{C}_i = \boldsymbol{U}_i\boldsymbol{\Lambda}_i\boldsymbol{U}_i^T. \tag{A.4}$$

Denote

$$\boldsymbol{T} = \begin{bmatrix} \boldsymbol{U}_1 & & & \\ & \boldsymbol{U}_2 & & \\ & & \ddots & \\ & & & \boldsymbol{U}_k \end{bmatrix}, \tag{A.5}$$

and $\boldsymbol{V} = \boldsymbol{U}\boldsymbol{T}$, which leads to

$$\boldsymbol{T}^T\boldsymbol{U}^T\boldsymbol{B}\boldsymbol{U}\boldsymbol{T} = \boldsymbol{\Lambda}_B \text{ and } \boldsymbol{T}^T\boldsymbol{U}^T\boldsymbol{A}\boldsymbol{U}\boldsymbol{T} = \boldsymbol{\Lambda}_A \tag{A.6}$$

where $\boldsymbol{V}^T\boldsymbol{V} = \boldsymbol{I}$. Hence, the lemma is proved. □

B PROOF OF THEOREM 4.1

Theorem. *Let $\delta = 1 - \cos(\theta_*/2)$ and $\theta_* = \theta((\boldsymbol{P} - \boldsymbol{X}\boldsymbol{X}^T)\boldsymbol{Q}, \boldsymbol{Q}(\boldsymbol{P} - \boldsymbol{X}\boldsymbol{X}^T))$ where $\boldsymbol{Q} = \boldsymbol{U}_o\boldsymbol{U}_o^T - \boldsymbol{I}/2$ and $o = \min(\text{rank}(\boldsymbol{X}), k)$. Under Assumption 4.1, if $\|\boldsymbol{P} - \boldsymbol{H}\boldsymbol{H}^T\| = \varepsilon \leq \mathcal{O}(\delta)$ and $\sigma_* \leq \mathcal{O}(\sqrt{\delta})$ where σ_* is the $(o+1)$-th largest singular value of $\boldsymbol{X}$, then there exists $\boldsymbol{W} \in \mathbb{R}^{d\times k}$ so that $\|\boldsymbol{P} - \boldsymbol{H}\boldsymbol{H}^T\| \leq \varepsilon$ In other words, if ε is small enough, then $\boldsymbol{H}\boldsymbol{H}^T$ could be a better approximation than $\boldsymbol{X}\boldsymbol{X}^T$.*

Proof. Use the notation $\ell(\cdot,\cdot,\cdot)$ as the reconstruction loss

$$\ell(\boldsymbol{P}, \boldsymbol{X}, \boldsymbol{W}) = \|\boldsymbol{P} - \boldsymbol{P}\boldsymbol{X}\boldsymbol{W}\boldsymbol{W}^T\boldsymbol{X}^T\boldsymbol{P}^T\|. \tag{B.1}$$

According to the conditions, we define

$$\boldsymbol{E} = \boldsymbol{P} - \boldsymbol{X}\boldsymbol{X}^T \Rightarrow \|\boldsymbol{E}\| = \varepsilon. \tag{B.2}$$

Apply SVD, we can factorize $\boldsymbol{X}$ as

$$\boldsymbol{X} = \boldsymbol{U}_o\boldsymbol{\Sigma}_o\boldsymbol{V}_o^T + \boldsymbol{U}_e\boldsymbol{\Sigma}_e\boldsymbol{V}_e^T, \tag{B.3}$$

where $e = d - o$. Clearly, we have $\boldsymbol{V}_o^T\boldsymbol{V}_e = \boldsymbol{0}$ and thus

$$\boldsymbol{P} = \boldsymbol{X}\boldsymbol{X}^T + \boldsymbol{E} = \boldsymbol{U}_k\boldsymbol{\Sigma}_k^2\boldsymbol{U}_k^T + \boldsymbol{U}_e\boldsymbol{\Sigma}_e^2\boldsymbol{U}_e^T + \boldsymbol{E}. \tag{B.4}$$

Therefore, $\boldsymbol{H}\boldsymbol{H}^T$ can be written as

$$\boldsymbol{H}\boldsymbol{H}^T = \boldsymbol{P}(\boldsymbol{U}_o\boldsymbol{\Sigma}_o\boldsymbol{V}_o^T + \boldsymbol{U}_e\boldsymbol{\Sigma}_e\boldsymbol{V}_e^T)\boldsymbol{W}\boldsymbol{W}^T(\boldsymbol{V}_o\boldsymbol{\Sigma}_o\boldsymbol{U}_o^T + \boldsymbol{V}_e\boldsymbol{\Sigma}_e\boldsymbol{U}_e^T)\boldsymbol{P}. \tag{B.5}$$

Let $\boldsymbol{W}_0$ be a valid solution as

$$\boldsymbol{W}_0 = \begin{cases} \boldsymbol{W}_0 & s.t.\ \boldsymbol{V}_k^T\boldsymbol{W}_0 = \boldsymbol{\Sigma}_k^{-2} \quad \text{if } \mathrm{rank}(\boldsymbol{X}) = k \\ [\boldsymbol{W}_r; \boldsymbol{0}] & s.t.\ \boldsymbol{V}_r^T\boldsymbol{W}_r = \boldsymbol{\Sigma}_r^{-2} \quad \text{if } \mathrm{rank}(\boldsymbol{X}) = r \leq k \end{cases}. \tag{B.6}$$

By the above definition, $\boldsymbol{V}_e^T\boldsymbol{W} = \boldsymbol{0}$. Therefore, with $\mathrm{rank}(\boldsymbol{X}) > k$,

$$\begin{aligned}
&\|\boldsymbol{H}\boldsymbol{H}^T - \boldsymbol{P}\| \\
=&\|\boldsymbol{U}_k\boldsymbol{\Sigma}_k^3\boldsymbol{V}_k\boldsymbol{W}_0\boldsymbol{W}_0^T\boldsymbol{V}_k\boldsymbol{\Sigma}_k^3\boldsymbol{U}_k^T + \boldsymbol{E}\boldsymbol{U}_k\boldsymbol{\Sigma}_k\boldsymbol{V}_k\boldsymbol{W}_0\boldsymbol{W}_0^T\boldsymbol{V}_k\boldsymbol{\Sigma}_k^3\boldsymbol{U}_k^T \\
&+ \boldsymbol{U}_k\boldsymbol{\Sigma}_k^3\boldsymbol{V}_k\boldsymbol{W}_0\boldsymbol{W}_0^T\boldsymbol{V}_k\boldsymbol{\Sigma}_k\boldsymbol{U}_k^T\boldsymbol{E} + \boldsymbol{E}\boldsymbol{U}_k\boldsymbol{\Sigma}_k\boldsymbol{V}_k\boldsymbol{W}_0\boldsymbol{W}_0^T\boldsymbol{V}_k\boldsymbol{\Sigma}_k\boldsymbol{U}_k^T\boldsymbol{E} - \boldsymbol{P}\| \\
=&\|\boldsymbol{U}_k\boldsymbol{\Sigma}_k^2\boldsymbol{U}_k^T + \boldsymbol{E}\boldsymbol{U}_k\boldsymbol{U}_k^T + \boldsymbol{U}_k\boldsymbol{U}_k^T\boldsymbol{E} + \boldsymbol{E}\boldsymbol{U}_k\boldsymbol{\Sigma}_k^{-2}\boldsymbol{U}_k^T\boldsymbol{E} - \boldsymbol{U}_k\boldsymbol{\Sigma}_k^2\boldsymbol{U}_k^T - \boldsymbol{U}_e\boldsymbol{\Sigma}_e^2\boldsymbol{U}_e^T - \boldsymbol{E}\| \\
=&\|\boldsymbol{E}\boldsymbol{U}_k\boldsymbol{U}_k^T + \boldsymbol{U}_k\boldsymbol{U}_k^T\boldsymbol{E} + \boldsymbol{E}\boldsymbol{U}_k\boldsymbol{\Sigma}_k^{-2}\boldsymbol{U}_k^T\boldsymbol{E} - \boldsymbol{U}_e\boldsymbol{\Sigma}_e^2\boldsymbol{U}_e^T - \boldsymbol{E}\| \\
\leq&\|\boldsymbol{E}\boldsymbol{U}_k\boldsymbol{U}_k^T + \boldsymbol{U}_k\boldsymbol{U}_k^T\boldsymbol{E} - \boldsymbol{E}\| + \|\boldsymbol{E}\boldsymbol{U}_k\boldsymbol{\Sigma}_k^{-2}\boldsymbol{U}_k^T\boldsymbol{E}\| + \|\boldsymbol{U}_e\boldsymbol{\Sigma}_e^2\boldsymbol{U}_e^T\|.
\end{aligned} \tag{B.7}$$

Similarly, if $\mathrm{rank}(\boldsymbol{X}) = r \leq k$,

$$\begin{aligned}
\|\boldsymbol{H}\boldsymbol{H}^T - \boldsymbol{P}\| =&\|\boldsymbol{E}\boldsymbol{U}_r\boldsymbol{U}_r^T + \boldsymbol{U}_r\boldsymbol{U}_r^T\boldsymbol{E} + \boldsymbol{E}\boldsymbol{U}_r\boldsymbol{\Sigma}_r^{-2}\boldsymbol{U}_r^T\boldsymbol{E} - \boldsymbol{E}\| &\text{(B.8)}\\
\leq&\|\boldsymbol{E}\boldsymbol{U}_r\boldsymbol{U}_r^T + \boldsymbol{U}_r\boldsymbol{U}_r^T\boldsymbol{E} - \boldsymbol{E}\| + \|\boldsymbol{E}\boldsymbol{U}_r\boldsymbol{\Sigma}_r^{-2}\boldsymbol{U}_r^T\boldsymbol{E}\|. &\text{(B.9)}
\end{aligned}$$

Now we focus on the general case, $\mathrm{rank}(\boldsymbol{X}) > k$ and the conclusion can be easily extended into the low-rank case. Note that

$$\|\boldsymbol{E}(\boldsymbol{U}_k\boldsymbol{U}_k^T - \frac{1}{2}\boldsymbol{I})\| = \|(\boldsymbol{U}_k\boldsymbol{U}_k^T - \frac{1}{2}\boldsymbol{I})\boldsymbol{E}\|, \tag{B.10}$$

and the first term can be written as

$$\begin{aligned}
&\|\boldsymbol{E}\boldsymbol{U}_k\boldsymbol{U}_k^T + \boldsymbol{U}_k\boldsymbol{U}_k^T\boldsymbol{E} - \boldsymbol{E}\|^2 \\
=&\|\boldsymbol{E}(\boldsymbol{U}_k\boldsymbol{U}_k^T - \frac{1}{2}\boldsymbol{I}) + (\boldsymbol{U}_k\boldsymbol{U}_k^T - \frac{1}{2}\boldsymbol{I})\boldsymbol{E}\|^2 \\
=&\|\boldsymbol{E}(\boldsymbol{U}_k\boldsymbol{U}_k^T - \frac{1}{2}\boldsymbol{I})\|^2 + \|(\boldsymbol{U}_k\boldsymbol{U}_k^T - \frac{1}{2}\boldsymbol{I})\boldsymbol{E}\|^2 + 2\langle\boldsymbol{E}(\boldsymbol{U}_k\boldsymbol{U}_k^T - \frac{1}{2}\boldsymbol{I}), (\boldsymbol{U}_k\boldsymbol{U}_k^T - \frac{1}{2}\boldsymbol{I})\boldsymbol{E}\rangle \\
=&2\|\boldsymbol{E}(\boldsymbol{U}_k\boldsymbol{U}_k^T - \frac{1}{2}\boldsymbol{I})\|^2 + 2s\|\boldsymbol{E}(\boldsymbol{U}_k\boldsymbol{U}_k^T - \frac{1}{2}\boldsymbol{I})\|^2 \\
=&2(1+s)\|\boldsymbol{E}(\boldsymbol{U}_k\boldsymbol{U}_k^T - \frac{1}{2}\boldsymbol{I})\|^2,
\end{aligned}$$

where

$$s = \frac{\langle\boldsymbol{E}(\boldsymbol{U}_k\boldsymbol{U}_k^T - \frac{1}{2}\boldsymbol{I}), (\boldsymbol{U}_k\boldsymbol{U}_k^T - \frac{1}{2}\boldsymbol{I})\boldsymbol{E}\rangle}{\|\boldsymbol{E}(\boldsymbol{U}_k\boldsymbol{U}_k^T - \frac{1}{2}\boldsymbol{I})\|^2}. \tag{B.11}$$

Due to that

$$\begin{aligned}
\|\boldsymbol{E}(\boldsymbol{U}_k\boldsymbol{U}_k^T - \frac{1}{2}\boldsymbol{I})\|^2 &= \mathrm{tr}(\boldsymbol{E}^2(\boldsymbol{U}_k\boldsymbol{U}_k^T - \frac{1}{2}\boldsymbol{I})) \\
&= \mathrm{tr}(\boldsymbol{E}^2(\boldsymbol{U}_k\boldsymbol{U}_k^T\boldsymbol{U}_k\boldsymbol{U}_k^T - 2\times\frac{1}{2}\boldsymbol{U}_k\boldsymbol{U}_k^T + \frac{1}{4}\boldsymbol{I})) \\
&= \frac{1}{4}\mathrm{tr}(\boldsymbol{E}^2) = \frac{1}{4}\varepsilon^2,
\end{aligned}$$

we have

$$\|\boldsymbol{E}\boldsymbol{U}_k\boldsymbol{U}_k^T + \boldsymbol{U}_k\boldsymbol{U}_k^T\boldsymbol{E} - \boldsymbol{E}\| = \sqrt{2(1+s)}\|\boldsymbol{E}(\boldsymbol{U}_k\boldsymbol{U}_k^T - \frac{1}{2}\boldsymbol{I})\| = \sqrt{\frac{1+s}{2}}\varepsilon. \tag{B.12}$$

Let $\boldsymbol{Q} = (\boldsymbol{U}_k\boldsymbol{U}_k^T - \boldsymbol{I}/2)$ and s can be reformulated as

$$s = \cos(\boldsymbol{E}\boldsymbol{Q}, \boldsymbol{Q}\boldsymbol{E}), \tag{B.13}$$

and we have the following definition

$$\theta_* = \arccos(s) = \theta(\boldsymbol{EQ}, \boldsymbol{QE}). \tag{B.14}$$

According to Lemma A.1, Assumption 4.1 indicates that

$$\boldsymbol{E}(\boldsymbol{U}_k\boldsymbol{U}_k^T - \frac{1}{2}\boldsymbol{I}) = (\boldsymbol{U}_k\boldsymbol{U}_k^T - \frac{1}{2}\boldsymbol{I})\boldsymbol{E} \Rightarrow s < 1. \tag{B.15}$$

And therefore, $\sqrt{(1+s)/2} \in [0, 1)$. Let $\delta = 1 - \sqrt{(1+s)/2} = 1 - \cos(\theta_*/2) > 0$ and the above equation can be reformulated as

$$\|\boldsymbol{E}\boldsymbol{U}_k\boldsymbol{U}_k^T + \boldsymbol{U}_k\boldsymbol{U}_k^T\boldsymbol{E} - \boldsymbol{E}\| = (1-\delta)\varepsilon. \tag{B.16}$$

The second term can be formulated as

$$\begin{aligned}\|\boldsymbol{E}\boldsymbol{U}_k\boldsymbol{\Sigma}_k^{-2}\boldsymbol{U}_k^T\boldsymbol{E}\| \leq& \|\boldsymbol{U}_k\boldsymbol{\Sigma}_k^{-2}\boldsymbol{U}_k^T\|\varepsilon^2 &\text{(B.17)}\\ =&\varepsilon^2\sqrt{\mathrm{tr}(\boldsymbol{U}_k\boldsymbol{\Sigma}_k^{-4}\boldsymbol{U}_k^T)} = \varepsilon^2\sqrt{\mathrm{tr}(\boldsymbol{\Sigma}^{-4}\boldsymbol{U}_k^T\boldsymbol{U}_k)} &\text{(B.18)}\\ =&\varepsilon^2\sqrt{\sum_{i=1}^{k}\frac{1}{\sigma_i^4}} \leq \sqrt{r}\frac{\varepsilon^2}{\sigma_k^2}, &\text{(B.19)}\end{aligned}$$

while the third term is

$$\|\boldsymbol{U}_e\boldsymbol{\Sigma}_e^2\boldsymbol{U}_e^T\| = \|\boldsymbol{\Sigma}_e^2\| = (\sum_{i=k+1}^{n}\sigma_i^4)^{1/2} \leq \sqrt{n-k}\sigma_*^2, \tag{B.20}$$

To sum up, the error of $\mathcal{M}$ is bounded as

$$\|\boldsymbol{H}\boldsymbol{H}^T - \boldsymbol{P}\| \leq (1-\delta)\varepsilon + \sqrt{k}\frac{\varepsilon^2}{\sigma_k^2} + \sqrt{e}\sigma_*^2. \tag{B.21}$$

If

$$\sqrt{k}\frac{\varepsilon}{\sigma_k^2} \leq \frac{\delta}{2} \Rightarrow \varepsilon \leq \frac{\delta\sigma_k^2}{2\sqrt{k}}, \tag{B.22}$$

and

$$\sqrt{n-k}\sigma_*^2 \leq \frac{\delta}{2} \Rightarrow \sigma_* \leq \sqrt{\frac{\delta}{2\sqrt{n-k}}}. \tag{B.23}$$

then $\|\boldsymbol{H}\boldsymbol{H}^T - \boldsymbol{P}\| \leq \varepsilon$. In other words, when $\varepsilon \leq \mathcal{O}(\delta)$ and $\sigma_* \leq \mathcal{O}(\sqrt{\delta})$, the error will be bounded by ε.

For the case that $\mathrm{rank}(\boldsymbol{X}) = r < k$, it is not hard to verify that

$$\begin{aligned}\|\boldsymbol{H}\boldsymbol{H}^T - \boldsymbol{P}\| \leq& \|\boldsymbol{E}\boldsymbol{U}_r\boldsymbol{U}_r^T + \boldsymbol{U}_r\boldsymbol{U}_r^T\boldsymbol{E} - \boldsymbol{E}\| + \|\boldsymbol{E}\boldsymbol{U}_r\boldsymbol{\Sigma}_r^{-2}\boldsymbol{U}_r^T\boldsymbol{E}\| &\text{(B.24)}\\ \leq& (1-\delta)\varepsilon + \sqrt{r}\frac{\varepsilon^2}{\sigma_r^2}. &\text{(B.25)}\end{aligned}$$

As

$$\sqrt{r}\frac{\varepsilon}{\sigma_r^2} \leq \frac{\delta}{2} \Rightarrow \varepsilon \leq \frac{\delta\sigma_r^2}{2\sqrt{r}} = \mathcal{O}(\delta), \tag{B.26}$$

and $\sigma_* = 0 \leq \mathcal{O}(\sqrt{\delta})$, we get $\|\boldsymbol{H}\boldsymbol{H}^T - \boldsymbol{P}\| \leq \varepsilon$. Hence, we have

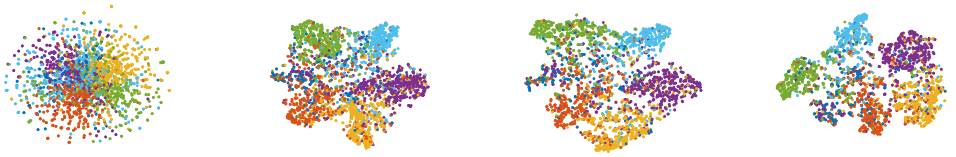
$$\min_{\boldsymbol{W}} \ell(\boldsymbol{P}, \boldsymbol{X}, \boldsymbol{W}) \leq \ell(\boldsymbol{P}, \boldsymbol{X}, \boldsymbol{W}_0) \leq \varepsilon, \tag{B.27}$$

and the theorem is proved. □

Table 4: Data Information

Dataset	Nodes	Edges	Classes	Features	Train / Val / Test Nodes
Cora	2,708	5,429	7	1,433	140 / 500 / 1,000
Citeseer	3,327	4,732	6	3,703	120 / 500 / 1,000
Pubmed	19,717	44,338	3	500	60 / 500 / 1,000
Reddit	233K	11.6M	41	602	152K / 24K / 55K

(a) $\boldsymbol{H}_0$: Input of $\mathcal{M}_1$ (b) $\boldsymbol{H}_1$: From $\mathcal{M}_1$ to $\mathcal{M}_2$ (c) $\boldsymbol{H}_2$: From $\mathcal{M}_2$ to $\mathcal{M}_3$ (d) $\boldsymbol{H}_3$: Output of $\mathcal{M}_3$



(e) $\boldsymbol{H}_0$: Input of $\mathcal{M}_1$ (f) $\boldsymbol{H}_1$: From $\mathcal{M}_1$ to $\mathcal{M}_2$ (g) $\boldsymbol{H}_2$: From $\mathcal{M}_2$ to $\mathcal{M}_3$ (h) $\boldsymbol{H}_3$: Output of $\mathcal{M}_3$

Figure 5: Visualization of a trained SGNN comprised of 3 modules on node classification of Cora and Citeseer. The first line shows the visualization of Cora and the bottom line shows the visualization of Citeseer.

C Proof of Theorem 4.2

Theorem. *If Assumption 4.1 does not hold, then there exists* $\boldsymbol{W} \in \mathbb{R}^{d\times k}$ *so that* $\|\boldsymbol{P} - \boldsymbol{H}\boldsymbol{H}\| \leq \varepsilon + \mathcal{O}(\sigma_*^2)$.

Proof. According to Ineq. (B.7),

$$\|\boldsymbol{H}\boldsymbol{H}^T - \boldsymbol{P}\| \leq \|\boldsymbol{E}\boldsymbol{U}_k\boldsymbol{U}_k^T + \boldsymbol{U}_k\boldsymbol{U}_k^T\boldsymbol{E} - \boldsymbol{E}\| + \|\boldsymbol{E}\boldsymbol{U}_k\boldsymbol{\Sigma}_k^{-2}\boldsymbol{U}_k^T\boldsymbol{E}\| + \|\boldsymbol{U}_e\boldsymbol{\Sigma}_e^2\boldsymbol{U}_e^T\|. \tag{C.1}$$

Suppose that $\boldsymbol{E} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^T$ so that $\boldsymbol{P} = \boldsymbol{U}(\boldsymbol{S} + \boldsymbol{\Lambda})\boldsymbol{U}^T$ where $\boldsymbol{S} = \boldsymbol{\Sigma}\boldsymbol{\Sigma}^T = \mathrm{diag}(\boldsymbol{\Sigma}_k^2; \boldsymbol{0})$.

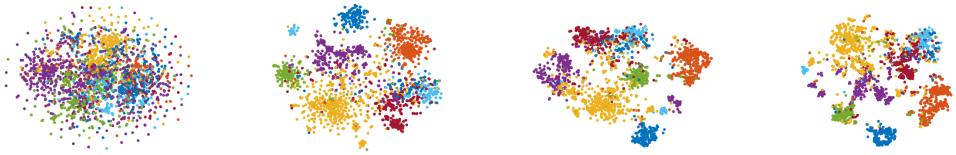
$$\begin{aligned}\boldsymbol{P} = \boldsymbol{U}(\boldsymbol{S} + \boldsymbol{\Lambda})\boldsymbol{U}^T &= [\boldsymbol{U}_r, \boldsymbol{U}_e]\begin{bmatrix}\boldsymbol{\Sigma}_r^2 + \boldsymbol{\Lambda}_r & \boldsymbol{0}\\ \boldsymbol{0} & \boldsymbol{\Sigma}_e^2 + \boldsymbol{\Lambda}_e\end{bmatrix}\begin{bmatrix}\boldsymbol{U}_r^T\\ \boldsymbol{U}_e^T\end{bmatrix}\\ &= \boldsymbol{U}_k(\boldsymbol{\Sigma}_k^2 + \boldsymbol{\Lambda}_k)\boldsymbol{U}_k^T + \boldsymbol{U}_e(\boldsymbol{\Sigma}_e^2 + \boldsymbol{\Lambda}_e)\boldsymbol{U}_e^T\end{aligned}$$

Then let $\boldsymbol{V}_k\boldsymbol{W} = (\boldsymbol{\Sigma}_k^3 + \boldsymbol{\Lambda}_k\boldsymbol{\Sigma}_k)^\dagger(\boldsymbol{\Sigma}_k^2 + \boldsymbol{\Lambda}_k)^{1/2}$

$$\begin{aligned}\|\boldsymbol{P} - \boldsymbol{H}\boldsymbol{H}^T\| &= \|\boldsymbol{U}(\boldsymbol{S} + \boldsymbol{\Lambda})\boldsymbol{\Sigma}\boldsymbol{V}^T\boldsymbol{W}\boldsymbol{W}^T\boldsymbol{V}\boldsymbol{\Sigma}^T(\boldsymbol{S} + \boldsymbol{\Lambda})\boldsymbol{U}^T - \boldsymbol{P}\|\\ &= \|\boldsymbol{U}_k(\boldsymbol{\Sigma}_k^3 + \boldsymbol{\Lambda}_k\boldsymbol{\Sigma}_k)\boldsymbol{V}_k^T\boldsymbol{W}\boldsymbol{W}^T\boldsymbol{V}_k(\boldsymbol{\Sigma}_k^3 + \boldsymbol{\Lambda}_k\boldsymbol{\Sigma}_k)\boldsymbol{U}_k^T - \boldsymbol{P}\|\\ &= \|\boldsymbol{U}_k\hat{\boldsymbol{I}}(\boldsymbol{\Sigma}_k^2 + \boldsymbol{\Lambda}_k)\hat{\boldsymbol{I}}\boldsymbol{U}_k^T - \boldsymbol{U}_k(\boldsymbol{\Sigma}_k^2 + \boldsymbol{\Lambda}_k)\boldsymbol{U}_k^T - \boldsymbol{U}_e(\boldsymbol{\Sigma}_e^2 + \boldsymbol{\Lambda}_e)\boldsymbol{U}_e^T\| \qquad \text{(C.2)}\\ &= \|\boldsymbol{U}_e(\boldsymbol{\Sigma}_e^2 + \boldsymbol{\Lambda}_e)\boldsymbol{U}_e^T\| \leq \|\boldsymbol{\Sigma}_e^2\| + \|\boldsymbol{\Lambda}_e\|\\ &\leq \varepsilon + \mathcal{O}(\sigma_*^2)\end{aligned}$$

where $\hat{\boldsymbol{I}} = (\boldsymbol{\Sigma}_k^3 + \boldsymbol{\Lambda}_k\boldsymbol{\Sigma}_k)(\boldsymbol{\Sigma}_k^3 + \boldsymbol{\Lambda}_k\boldsymbol{\Sigma}_k)^\dagger = \boldsymbol{\Sigma}_k(\boldsymbol{\Sigma}_k^2 + \boldsymbol{\Lambda}_k)(\boldsymbol{\Sigma}_k^2 + \boldsymbol{\Lambda}_k)^\dagger\boldsymbol{\Sigma}_k^{-1}$. Clearly, $\hat{\boldsymbol{I}} = \mathbb{1}\{\boldsymbol{\Sigma}_r^2 + \boldsymbol{\Lambda}_r \neq 0\}$. Therefore, Hence, the theorem is proved. □

Corollary C.1. *If Assumption 4.1 does not hold and* $\mathrm{rank}(\boldsymbol{X}) \leq k$, *then there exists* $\boldsymbol{W} \in \mathbb{R}^{d\times k}$ *so that* $\|\boldsymbol{P} - \boldsymbol{H}\boldsymbol{H}\| \leq \varepsilon$.



(a) $\boldsymbol{H}_0$: Input of $\mathcal{M}_1$ (b) $\boldsymbol{H}_1$: From $\mathcal{M}_1$ to $\mathcal{M}_2$ (c) $\boldsymbol{H}_2$: From $\mathcal{M}_2$ to $\mathcal{M}_3$ (d) $\boldsymbol{H}_3$: Output of $\mathcal{M}_3$

(e) $\boldsymbol{H}_0$: Input of $\mathcal{M}_1$ (f) $\boldsymbol{H}_1$: From $\mathcal{M}_1$ to $\mathcal{M}_2$ (g) $\boldsymbol{H}_2$: From $\mathcal{M}_2$ to $\mathcal{M}_3$ (h) $\boldsymbol{H}_3$: Output of $\mathcal{M}_3$

Figure 6: Visualization of a trained SGNN comprised of 3 modules on node clustering of Cora and Citeseer. The first line is visualization on Cora and the second line is visualization on Citeseer.

(a) $\boldsymbol{H}_1$ after 1 epoch (b) $\boldsymbol{H}_1$ after 2 epochs (c) $\boldsymbol{H}_1$ after 3 epochs (d) $\boldsymbol{H}_1$ after 5 epochs

Figure 7: t-SNE Visualization of the output of $\mathcal{M}_1$ from a trained SGNN comprised of 3 modules on node classification of Citeseer.

Proof. Suppose that $\boldsymbol{E} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^T$ so that $\boldsymbol{P} = \boldsymbol{U}(\boldsymbol{S}+\boldsymbol{\Lambda})\boldsymbol{U}^T$ where $\boldsymbol{S} = \boldsymbol{\Sigma}\boldsymbol{\Sigma}^T = \mathrm{diag}(\boldsymbol{\Sigma}_r^2; \boldsymbol{0})$.

$$\boldsymbol{P} = \boldsymbol{U}(\boldsymbol{S}+\boldsymbol{\Lambda})\boldsymbol{U}^T = [\boldsymbol{U}_r, \boldsymbol{U}_e] \begin{bmatrix} \boldsymbol{\Sigma}_r^2 + \boldsymbol{\Lambda}_r & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{\Lambda}_e \end{bmatrix} \begin{bmatrix} \boldsymbol{U}_r^T \\ \boldsymbol{U}_e^T \end{bmatrix} = \boldsymbol{U}_r(\boldsymbol{\Sigma}_r^2 + \boldsymbol{\Lambda}_r)\boldsymbol{U}_r^T + \boldsymbol{U}_e\boldsymbol{\Lambda}_e\boldsymbol{U}_e^T \tag{C.3}$$

Let $\boldsymbol{W} = [\boldsymbol{W}_r; \boldsymbol{0}]$ subjected to $\boldsymbol{V}_r^T\boldsymbol{W}_r = (\boldsymbol{\Sigma}_r^3 + \boldsymbol{\Lambda}_r\boldsymbol{\Sigma}_r)^\dagger(\boldsymbol{\Sigma}^2 + \boldsymbol{\Lambda}_r)^{1/2}$ Then

$$\begin{aligned} \|\boldsymbol{H}\boldsymbol{H}^T - \boldsymbol{P}\| &= \|\boldsymbol{U}(\boldsymbol{S}+\boldsymbol{\Lambda})\boldsymbol{\Sigma}\boldsymbol{V}^T\boldsymbol{W}\boldsymbol{W}^T\boldsymbol{V}\boldsymbol{\Sigma}^T(\boldsymbol{S}+\boldsymbol{\Lambda})\boldsymbol{U}^T - \boldsymbol{P}\| \\ &= \|\boldsymbol{U}_r(\boldsymbol{\Sigma}_r^3 + \boldsymbol{\Lambda}_r\boldsymbol{\Sigma}_r)\boldsymbol{V}_r^T\boldsymbol{W}\boldsymbol{W}^T\boldsymbol{V}_r(\boldsymbol{\Sigma}_r^3 + \boldsymbol{\Lambda}_r\boldsymbol{\Sigma}_r)\boldsymbol{U}_r^T - \boldsymbol{P}\| \\ &= \|\boldsymbol{U}_r\hat{\boldsymbol{I}}(\boldsymbol{\Sigma}_r^2 + \boldsymbol{\Lambda}_r)\hat{\boldsymbol{I}}\boldsymbol{U}_r^T - \boldsymbol{U}_r(\boldsymbol{\Sigma}_r^2 + \boldsymbol{\Lambda}_r)\boldsymbol{U}_r^T - \boldsymbol{U}_e\boldsymbol{\Lambda}_e\boldsymbol{U}_e^T\|, \end{aligned}$$

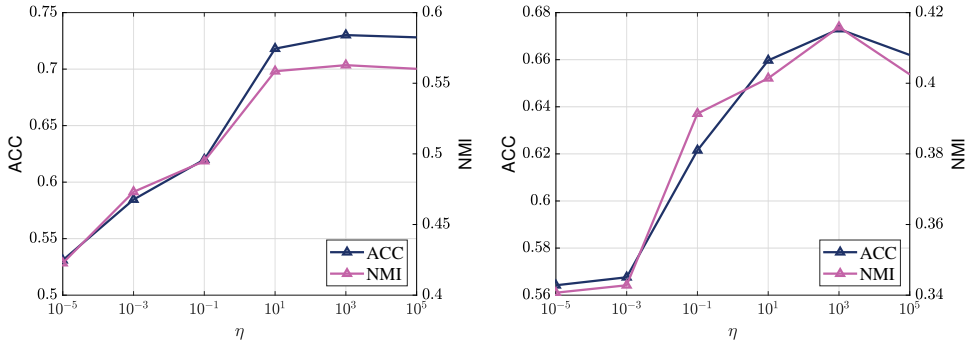
where $\hat{\boldsymbol{I}} = (\boldsymbol{\Sigma}_r^3 + \boldsymbol{\Lambda}_r\boldsymbol{\Sigma}_r)(\boldsymbol{\Sigma}_r^3 + \boldsymbol{\Lambda}_r\boldsymbol{\Sigma}_r)^\dagger = \boldsymbol{\Sigma}_r(\boldsymbol{\Sigma}_r^2 + \boldsymbol{\Lambda}_r)(\boldsymbol{\Sigma}_r^2 + \boldsymbol{\Lambda}_r)^\dagger\boldsymbol{\Sigma}_r^{-1}$. Clearly, $\hat{\boldsymbol{I}} = \mathbb{1}\{\boldsymbol{\Sigma}_r^2 + \boldsymbol{\Lambda}_r \neq 0\}$. Therefore,

$$\|\boldsymbol{H}\boldsymbol{H}^T - \boldsymbol{P}\| = \|\boldsymbol{U}_e\boldsymbol{\Lambda}_e\boldsymbol{U}_e^T\| = \|\boldsymbol{\Lambda}_e\| \leq \varepsilon. \tag{C.4}$$

Hence, the corollary is proved. □

D COMPLEXITY

As each base module is assumed as a separable GNN, both FT and BT of $\mathcal{M}_t$ can be divided into two steps, the preprocessing step for graph operation and the training step for parameters learning. Denote the output dimension of $\mathcal{M}_t$ as d_t and the dimension of original content feature as $d_0 = d$. The preprocessing to compute $\boldsymbol{X}_t' = f_0(\boldsymbol{A}, \boldsymbol{X}_t)$ requires $\mathcal{O}(\|\boldsymbol{A}\|_0 d_t)$ cost. Suppose that each module is trained E iterations and the batch size is set as m. Then the computation cost of the training step is $\mathcal{O}(Emd_{t-1}d_t)$. Note that only the GNN mapping is considered and the computation of the loss is ignored. Overall, the computational complexity of an SGNN with L modules is approximately



(a) Cora (b) Citeseer

Figure 8: Impact of η to node clustering on Cora and Citeseer

$\mathcal{O}(K\|\boldsymbol{A}\|_0 \sum_{t=0}^{L-1} d_t + Em \sum_{t=1}^{L} d_{t-1}d_t)$. Remark that the graph is only used once during every epoch and no sampling is processed on the graph such that the graph structure is completely retained which is unavailable in the existing fast GNNs. The coefficient of $\|\boldsymbol{A}\|_0$ is only $K \sum_{t=1}^{L-1} d_t$. The space complexity is only $\mathcal{O}(\|\boldsymbol{A}\|_0 + md_{t-1}d_t)$. Therefore, the growth of graph scale will not affect the efficiency of SGNN. Due to the efficiency, all experiments can be conducted on a PC with an NVIDIA 1660 (6GB) and 16GB RAM.

E More Experiments

E.1 Omitted Information in Section 5

The details of four common datasets are shown in Table 4. We also provide more visualizations which are not included in the main paper in Figures 5 and 6 due to the limitation of space. We run SGNN with 3 GNN modules and visualize the input and output of $\mathcal{M}_1$, $\mathcal{M}_2$, and $\mathcal{M}_3$ through t-SNE on Cora and Citeseer, for node clustering and node classification. The purpose of these two figures is to empirically investigate *whether the decoupling would cause the accumulation of residuals and errors*. The experimental results support the theoretical results that are provided in Section 4. One may concern the impact of η (trade-off coefficient between $\mathcal{L}_{FT}$ and $\mathcal{L}_B T$) on the performance. We testify SGNN with different η from $\{10^{-5}, 10^{-3}, 10^{-1}, 10^{1}, 10^{3}, 10^{5}\}$ and find that $\eta = 10^3$ **usually leads to good results**. Accordingly, we only report results SGNN with $\eta = 10^3$ in this paper. Moreover, we show the impact of η to node clustering on Cora and Citeseer in Figure 8.

Moreover, we show the output of $\mathcal{M}_1$ of different periods in Figure 7, in order to show the impact of the backward training. From the figure, we find that BT indeed affects the latent features, which is particularly apparent in Figure 7d.

For SGNN-S^{2}GC and GAE-S^{2}GC, all extra hyper-parameters are simply set according to the original paper of S^{2}GC for both node clustering and node classification. We do not tune the hyper-parameters of S^{2}GC manually.

E.2 Experiments on More Datasets

We further show some experiments of node classification on two OGB datasets, OGB-Products and OGB-Arxiv, which are downloaded from https://ogb.stanford.edu/docs/nodeprop/. The OGB-Products contains more than 2 million nodes and OGB-Arxiv contains more than 150 thousand nodes.

It should be emphasized that we only use the simple single-layer GCN as the base module of SGNN. The performance can be further improved by incorporating different models such as GCNII, GIN, *etc*. In particular, we only tune hyper-parameters on Arxiv, and we simply report results of SGNN with settings from Reddit.

Table 5: Node Classification Results on Large Datasets

Datasets	Products		Arxiv	
	Test Acc	Val Acc	Test Acc	Val Acc
MLP	61.06	75.54	55.50	57.65
GCN	75.64	92.00	**71.74**	**73.00**
GraphSAGE	<u>78.50</u>	**92.24**	71.49	<u>72.77</u>
Cluster-GCN	**78.97**	<u>92.12</u>	N/A	N/A
Softmax	47.70	N/A	52.77	N/A
SGC	68.87	N/A	68.78	N/A
S^2GC	70.22	N/A	70.15	N/A
SGNN-BT	74.44	91.13	<u>71.57</u>	71.66

Table 6: Investigation about the impact of L on SGNN and GAE regarding node clustering.

	Cora			Citeseer			Pubmed		
L	SGNN	GAE	FastGAE	SGNN	GAE	FastGAE	SGNN	GAE	FastGAE
2	0.75	0.60	0.35	0.67	0.41	0.27	0.70	0.69	0.43
3	0.66	0.63	0.33	0.65	0.58	0.25	0.64	0.64	0.42
4	0.68	0.65	0.33	0.59	0.58	0.24	0.64	0.60	0.41
5	0.69	0.62	0.33	0.53	0.45	0.24	0.64	0.60	0.41
6	0.69	0.53	0.32	0.44	0.32	0.24	0.64	0.48	0.41
7	0.68	0.52	0.32	0.44	0.31	0.24	0.64	0.46	—

E.3 Investigation of Depth

We also conduct sufficient experiments on the depth of SGNN. All results are reported in Table 6. To ensure fairness, we also show the performance of GAE with the same depth though deeper GCN and GAE usually return unsatisfied results. Note that the dimensions are set as $[256, 128, 64, 32, 16, 16, 16]$.

F Source Code

All codes will be publicly available on our GitHub homepage after the formal publication. The source codes are also uploaded in supplementary materials.