Soft-mask: Adaptive Substructure Extractions for Graph Neural Networks

Mingqi Yang
Dalian University of Technology
Dalian, China
yangmq@mail.dlut.edu.cn

Yanming Shen*
Dalian University of Technology
Dalian, China
shen@dlut.edu.cn

Heng Qi
Dalian University of Technology
Dalian, China
hengqi@dlut.edu.cn

Baocai Yin
Dalian University of Technology
Peng Cheng Laboratory
Dalian, China
ybc@dlut.edu.cn

ABSTRACT

For learning graph representations, not all detailed structures within a graph are relevant to the given graph tasks. Task-relevant structures can be *localized* or *sparse* which are only involved in subgraphs or characterized by the interactions of subgraphs (a hierarchical perspective). A graph neural network should be able to efficiently extract task-relevant structures and be invariant to irrelevant parts, which is challenging for general message passing GNNs. In this work, we propose to learn graph representations from a sequence of subgraphs of the original graph to better capture task-relevant substructures or hierarchical structures and skip *noisy* parts. To this end, we design soft-mask GNN layer to extract desired subgraphs through the mask mechanism. The soft-mask is defined in a continuous space to maintain the differentiability and characterize the weights of different parts. Compared with existing subgraph or hierarchical representation learning methods and graph pooling operations, the soft-mask GNN layer is not limited by the fixed sample or drop ratio, and therefore is more flexible to extract subgraphs with arbitrary sizes. Extensive experiments on public graph benchmarks show that soft-mask mechanism brings performance improvements. And it also provides interpretability where visualizing the values of masks in each layer allows us to have an insight into the structures learned by the model.

CCS CONCEPTS

• **Computing methodologies** → **Neural networks**; *Supervised learning by classification.*

KEYWORDS

deep learning, graph neural networks, graph representation leanring

---

*Corresponding author

1 INTRODUCTION

Graph structure data is ubiquitous. Molecules, social networks, and many other applications can be modeled as graphs. Recently, graph neural networks (GNNs) have shown their power in graph representation learning [2, 9, 13]. General GNNs follow neighborhood aggregation (or message passing) scheme [8], where node representations are computed iteratively by aggregating transformed representations of its neighbors. The aggregation operation on each node is shared with the same parameters, with structural information learned implicitly.

One issue of this kind of scheme is that task-relevant structural information is likely to be mixed up with irrelevant (or noisy) parts, making it indistinguishable for the downstream processing, especially for long-range dependency captured by higher layers in a deep model [4, 17, 22]. To handle this, one possible strategy is to improve the ability to distinguish different graph topologies. Hopefully, useful structural information will be retained and passed to the higher layers for further processing. This strategy is followed by GIN [32] whose ability to distinguish different graph structures is equivalent to 1-order Weisfeiler-Lehman (WL) graph isomorphism test. Other GNN implementations [5, 18, 19] with discrimination power in analogy to high-order WL test are also proposed.

Ideally, GNNs should behave in an *extract* manner. For noisy graphs, the learned representations should correspond to the subgraph with noisy parts not involved. To achieve this, one basic idea is to restrict all layers learing on the same subgraph. It can be considered that a k-layer GNN learns on a sequence of subgraphs of length k.

For graphs characterized with hierarchical structures, to explicitly capture the hierarchical structures, the neural network should also encode them in a hierarchical manner. We explain this motivation empirically with a simple example of hierarchical graphs with a height of 2 as given in Figure 1 (a), where the structure is identified by three individual parts (in different colors) and their

Figure 1: An example of hierarchical graph structures.

interactions. Figure 1 (b) demonstrates that in this graph, nodes can be divided into two groups:

- **Leaf node.** Nodes only have connections within each part such as white nodes in Figure 1(b);
- **Root node.** Nodes have connections with other parts or shared with other parts such as black nodes in Figure 1(b).

Clearly, the interactions among different parts are only decided by root nodes. By removing all leaf nodes, we obtain a *reduced* subgraph that only demonstrates the interactions among different parts. To explicitly capture the interactions among different parts, the aggregation in lower layers should be only conducted on leaf nodes to encodes each part individually. Then root nodes are taken into consideration to encode the interactions of each part in higher layers. This requires that each layer of a neural network should be flexible to learn on any given subgraphs, and correspondingly the final representations are learned from a sequence of subgraphs. Since hierarchical graph structures can be viewed recursively as each individual part and their interactions, this mechanism is natural to be extended to hierarchical graphs with heights larger than 2.

According to the above analysis, learning graph representations from a sequence of subgraphs actually unifies subgraph representation learning and hierarchical representation learning. To reach this goal, there are two main challenges:

- How to efficiently represent any random subgraph in each layer while maintaining differentiability?
- How to decide the sequence of subgraphs learned by the model?

GAM [16] extracts desired substructures by representing a graph with a fixed-length sequence of nodes sampled from the original graph. Top-k graph poolings [7, 14, 15] score the importance of each node and drop low score nodes and related edges layer by layer. These strategies can dynamically focus on informative parts, making the neural network more robust in dealing with noisy graphs. However, GAM requires the sampled node sequence to be with the fixed length, and top-k poolings require a fixed drop ratio. These limitations can be obstacles to represent desired subgraphs with arbitrary scales.

To address these issues, we propose soft-mask GNN (SMG) layer. It extracts subgraphs of any size by controlling node masks. Then the problem of finding the desired subgraph is converted to finding proper mask assignments. We theoretically show that the learned representation by a soft-mask GNN layer with proper mask assignments is equivalent to the corresponding subgraph representation.

Stacking multiple such layers leads to the learned representation of a sequence of subgraphs that can be used to capture the *informative* substructures as well as hierarchical structures. Different from general GNNs that follow *dense* aggregation where the aggregation is conducted on all nodes, the aggregation in a soft-mask GNN layer can adaptively *shutdown* the aggregation of undesired parts while maintaining differentiability, which we call *sparse* aggregation.

The soft-mask GNN layer applies continuous mask values in order to maintain the differentiability of the networks. It characterizes the weights of different parts within a graph, which can also be considered as a global attention mechanism. This global attention mechanism provides interpretability, where visualizing mask value distributions in different layers on some public graph benchmarks provides insights of informative parts or hierarchical structures learned by the model.

Our contributions are summarized as follows:

- We propose the soft-mask GNN layer which achieves a kind of sparse aggregation in general GNNs. It is used to represent any given subgraph with an arbitrary size.
- We theoretically analyze that by learning a graph representation from a sequence of individual subgraphs of the original graph, our model is capable of extracting any desired substructures or hierarchical structures.
- We evaluate the soft-mask GNN on public graph benchmarks and show a significant improvement over state-of-the-art approaches. Furthermore, by visualizing the mask values in different layers, we provide insights on structural information learned by the model.

2 PRELIMINARIES

2.1 Notations

For a graph G, we denote the set of edges, nodes and node feature vectors respectively by E_G, V_G and X_G. $\mathcal{N}(v)$ is the set of *neighbors* of node v, i.e., $\mathcal{N}(v) = \{u \in V_G | (u, v) \in E_G\}$. Let $\hat{\mathcal{N}}(v) = \{v\} \cup \mathcal{N}(v)$. Let S be a subset of nodes, i.e., $S \subseteq V_G$. Then the subgraph G_S of G is the graph whose vertex set is S and whose edge set consists of all of the edges in E_G that have both endpoints in S. Also, we use $[n]$ to denote $\{1, 2, ..., n\}$ and $\{\{...\}\}$ to denote a multiset, i.e., a set with possibly repeating elements.

2.2 Graph Neural Networks

GNNs learn the representation vector of a node, $\mathbf{h}_v$, or the entire graph, $\mathbf{h}_G$, from a graph, utilizing both node features and the structure of the graph. Most proposed GNNs fit within the neighborhood aggregation framework [8]. Formally, the k-th layer of a GNN is

$$\mathbf{h}_v^{(k)} = \text{Update}(\mathbf{h}_v^{(k-1)}, \text{Aggregate}(\{\{\mathbf{h}_u^{(k-1)} | u \in \mathcal{N}(v)\}\})). \quad (1)$$

The node representations from the last iteration are then passed to a classier for node classification. For graph-level tasks such as graph classification, the Readout function is needed which aggregates all node representations from the last iteration to obtain the entire graph representation

$$\mathbf{h}_G = \text{Readout}(\{\{\mathbf{h}_v^{(K)} | v \in V_G\}\}), \quad (2)$$

where $\mathbf{h}_v^{(K)}$ is node representations learned by a K-layer GNN. $\mathbf{h}_G$ is then passed to a classifier for graph classification.

Aggregate(.) in Equation 1 is used to aggregate neighbors' representations to generate current node representation, and Readout(.) in Equation 2 is used to aggregate all node representations to generate the entire graph representation. In order for the GNNs to be invariant to graph isomorphism, Aggregate(.) and Readout(.) should be permutation invariant. Meanwhile, Equation 1 and Equation 2 should be differentiable to make the network trainable through backpropagation.

3 SOFT-MASK GNN

In this section, we first present the SMG layer and explain how it is used to extract the desired subgraph by controlling the mask assignments. Then, we show how a multi-layer SMG learns subgraph representations and hierarchical representations respectively with corresponding subgraph sequences. We also present a method to compute mask assignments, which can automatically extract substructures or hierarchical structures through backpropagation. Finally, we generalize the SMG to multi-channel scenarios.

3.1 Sparse Aggregation and Subgraph Representation Learning

GNN operation is viewed as a kind of smoothing operation [4, 17, 22], where each node exchanges feature information with its neighbors. This kind of scheme does not skip irrelevant parts explicitly. On the contrary, our SMG layer extracts substructures as follows:

$$\mathbf{h}_v^{(k)} = \text{ReLU}(\mathbf{W}_1^{(k)} m_v^{(k)} [\mathbf{h}_v^{(k-1)} || \sum_{u \in \mathcal{N}(v)} m_u^{(k)} \mathbf{h}_u^{(k-1)}]), \quad (3)$$

where $m_v^{(k)} \in [0,1]$ refers to the soft-mask of node v at the k-th layer, $||$ denotes concatenation operation, and $\mathbf{W}_1^{(k)} \in \mathbf{R}^{d \times d}$ is a trainable matrix, where d is the dimension of node representations. The computation of $m_v^{(k)}$ will be presented in Section 3.3. The SMG layer should satisfy the following constraints:

- The linear transformation $\mathbf{W}_1^{(k)}$ does not include constant part (i.e., bias);
- The activation function (ReLU) satisfies $\sigma(0) = 0$;
- The aggregation operator (SUM) is *zero invariant*, which is formally defined as follows:

Zero invariant. The aggregation function $f : \{\{\mathbb{R}^m\}\} \to \mathbb{R}^n$ is zero invariant if and only if

$$f(\mathcal{S}) = \begin{cases} \mathbf{0} & \mathcal{S} = \emptyset \\ f(\mathcal{S}/\{\mathbf{0}\}) & \text{otherwise,} \end{cases}$$

where $\mathcal{S}$ is a multiset of vectors. Note that the SUM operator is zero invariant while the MEAN is not.

With the above restrictions, setting $m_v^{(k)} = 0$ leads to $\mathbf{h}_v^{(k)} = \mathbf{0}$ and $\mathbf{h}_u^{(k)} = \text{ReLU}(\mathbf{W}_1^{(k)} m_u^{(k)} [\mathbf{h}_u^{(k-1)} || \sum_{u' \in \mathcal{N}(u)/\{v\}} m_{u'}^{(k)} \mathbf{h}_{u'}^{(k-1)}])$ for any $u \in \mathcal{N}(v)$, which is said that node v is inaccessible for its neighbors. Therefore, for any subgraph G_S of G, the 1-layer SMG together with a zero invariant Readout function can represent G_S and completely skip other parts by controlling the mask assignments

as follows:

$$\begin{aligned} \mathbf{h}_G\big|_{m_v^{(1)}=1, v \in V_{G_S}; m_v^{(1)}=0, v \in V_G/V_{G_S}} \\ = \sum_{v \in V_{G_S}} \mathbf{h}_v^{(1)} + \sum_{v \in V_G/V_{G_S}} \mathbf{h}_v^{(1)} \\ = \sum_{v \in V_{G_S}} \text{ReLU}(\mathbf{W}_1^{(1)}[\mathbf{x}_v || \sum_{u \in \mathcal{N}_{G_S}(v)} \mathbf{x}_u]) + \mathbf{0} \\ = \mathbf{h}_{G_S}\big|_{m_v^{(1)}=1, v \in V_{G_S}}. \end{aligned}$$

It removes the restriction of general GNNs that the aggregation is conducted on all nodes, and therefore we call it as *sparse* aggregation. The selected subgraph includes all nodes with mask 1 and related edges.

In a multi-layer SMG, nodes with $m_v^{(k)} = 0$ are not actually removed. The GNN operation in the following layers would also involve them. In order to represent the subgraph G_S, a basic idea is to restrict all layers to learn on the same subgraph G_S by assigning $m_v^{(k)} = 1$ for all $v \in V_{G_S}$ and $m_u^{(k)} = 0$ for all $u \in V_G/V_{G_S}$ in all layers. This corresponds to a specific subgraph sequence that all subgraphs are the same. However, not all masks assigned on nodes in different layers have an effect on the final representations, and masks in some layers can take arbitrary values. Since different masks lead to a different subgraph sequence, this means that the required subgraph representation may correspond to a different subgraph sequence. To explain this, we use $\mathbf{H}_G^{(k)} = \{\{\mathbf{h}_v^{(k)} | v \in V_G\}\} = \text{SMG}_k(\mathbf{X}_G, \mathbf{A}_G, \mathbf{M})$ to denote the set of node representations learned by a k-layer SMG, where the input $\mathbf{M} \in [0,1]^{k \times n}$ is the preassigned masks at all k layers with $\mathbf{M}_{k,v} = m_v^{(k)}$. $\mathbf{M}$ defines a sequence of subgraphs of length k. $\hat{\mathbf{H}}_G^{(k)} = \{\{\hat{\mathbf{h}}_v^{(k)} | v \in V_G\}\} = \text{SMG}_k(\mathbf{X}_G, \mathbf{A}_G, \mathbf{1})$ is the set of node representations learned by k-layer SMG with $\mathbf{M} = \mathbf{1}^{k \times n}$, and correspondingly $\hat{\mathbf{h}}_v^{(k)} = \text{ReLU}(\mathbf{W}_1^{(k)}[\hat{\mathbf{h}}_v^{(k-1)} || \sum_{u \in \mathcal{N}(v)} \hat{\mathbf{h}}_u^{(k-1)}])$.

LEMMA 3.1. *For any subgraph G_S of G, Let $\mathbf{H}_G^{(K)} = \{\{\mathbf{h}_v^{(k)} | v \in V_G\}\}$ and $\hat{\mathbf{H}}_{G_S}^{(K)} = \{\{\hat{\mathbf{h}}_v^{(k)} | v \in V_{G_S}\}\}$. Then we have $\mathbf{h}_v^{(K)} = \hat{\mathbf{h}}_v^{(K)}$ for any $v \in V_{G_S}$, if the following condition holds,*

$$\mathbf{M}_{k,v} = m_v^{(k)} = \begin{cases} 0 & \{v\} \subseteq V_G/V_{G_S} \\ & \wedge \mathcal{N}_G(v) \cap V_{G_S} \neq \emptyset \\ & \wedge k\%2 = 1 \\ 1 & \{v\} \subseteq V_{G_S} \wedge \{k\} \subseteq [K]. \end{cases}$$

We prove Lemma 3.1 in Appendix A. Lemma 3.1 shows that we can obtain the node representations that are equivalent to node representations of any subgraph by controlling the assignments of $\mathbf{M}$. Meanwhile, $\mathbf{M}$ (or the subgraph sequences) is not unique for the given subgraph. This subgraph extraction mechanism takes effects implicitly by imposing restrictions on a combination of linear transformations, non-linear activation functions and aggregators. Compared with existing graph sampling [16] or graph pooling [3, 7, 14, 15], the benefit is that it does not require setting a fixed sample or drop ratio and can extract the desired subgraph with arbitrary sizes. For learning the graph-level representations, we have the following theorem.

THEOREM 3.2. *Let* $\mathbf{h}_G = SUM(\mathbf{H}_G^{(K)})$ *and* $\hat{\mathbf{h}}_{G_S} = SUM(\hat{\mathbf{H}}_{G_S}^{(K)})$, *where* G_S *can be any subgraph of* G. *Then there exist assignments of* $\mathbf{M}$ *such that* $\mathbf{h}_G = \hat{\mathbf{h}}_{G_S}$.

We prove Theorem 3.2 in Appendix B. Lemma 3.1 and Theorem 3.2 indicate that the representation of any given subgraph corresponds to a group of $\mathbf{M}$ (or a group of subgraph sequences). Then, the problem of representing the desired subgraphs is converted to finding proper $\mathbf{M}$.

3.2 Hierarchical Representation Learning

Figure 2: Capture the hierarchical structures by stacking multiple SMGs.

Graphs built from the real-world can inherently have hierarchical structures. In the previous section, we have shown how a multi-layer SMG represents a given subgraph with proper assignments of $\mathbf{M}$. In this section, we show how to control the assignments of $\mathbf{M}$ to capture the hierarchical structures of a graph.

A hierarchical graph as given in Figure 1(a) is characterized by each individual part and their interactions. Meanwhile, each part is composed of interactions of several smaller parts, making the graph structure identification recursive. To capture hierarchical structures, the neural network should first encode each local part individually and then encodes their interactions. From the node perspective, leaf nodes are aggregated first and then root nodes are taken into consideration. This corresponds to stacking multiple SMGs, where each SMG_{k_i} learns node representations of the subgraph G_{S_i} with input node features being node representations learned by previous $\text{GNN}_{k_{i-1}}$. In the i-th GNN_{k_i}, $\mathbf{H}_G^{\star} = \text{SMG}_{k_i}(\mathbf{H}_G^{\star-k_i}, \mathbf{A}_G, \mathbf{M}_{G_{S_i}})$, where $\mathbf{M}_{G_{S_i}}$ is the assignment corresponding to G_{S_i} according to Lemma 3.1, and k_i is the number of layers of SMG_{k_i}. In the initial step, $\mathbf{H}_G^0 = \mathbf{X}_G$.

We explain this process with the hierarchical graph example in Figure 1. The hierarchical structure is characterized by three individual parts and their interactions. We use a stacked 2 SMGs to

learn on this graph as showed in Figure 2. Based on the observation of Lemma 3.1, the representation learned by SMG_{k_1} corresponds to subgraph 1 with only leaf nodes involved, leading to the learned node representations limited within the corresponding part. Also, the representation learned by SMG_{k_2} corresponds to subgraph 2 consisting of root nodes (colored with black) and leaf nodes (colored with gray) which capture each part as well as their interactions. Consistently, to extend to hierarchical graphs with heights larger than 2, stacking more SMGs with each one learning on a specific subgraph is required. Interactions of different parts at low levels are captured by lower SMGs, and those at high levels are captured by higher SMGs.

We have shown how to capture hierarchical structures by stacking multi SMGs with each one corresponding to a subgraph as analyzed in Lemma 3.1. This assumes the knowledge of the hierarchical structures of the graph. However, the hierarchical structures are not a priori knowledge and should be captured by the neural network itself. Thanks to

$$\begin{aligned}
\mathbf{H}_G^{(K)} &= \text{SMG}_{k_L}(\mathbf{H}_G^{(K-k_L)}, \mathbf{A}_G, \mathbf{M}_{G_{S_L}}) \\
&= \text{SMG}_{k_L}(\text{SMG}_{k_{L-1}}(\dots \text{SMG}_{k_1}(\mathbf{X}_G, \mathbf{A}_G, \mathbf{M}_{G_{S_1}}), \\
&\quad \mathbf{A}_G, \mathbf{M}_{G_{S_{L-1}}}), \mathbf{A}_G, \mathbf{M}_{G_{S_L}}) \\
&= \text{SMG}_K(\mathbf{X}_G, \mathbf{A}_G, \overset{L}{\underset{i=1}{||}} \mathbf{M}_{G_{S_i}}) \\
&= \text{SMG}_K(\mathbf{X}_G, \mathbf{A}_G, \mathbf{M}_G),
\end{aligned}$$

which means that any required stacked L SMGs with each one learning on a specific subgraph are equivalent to the same number layers of SMG with mask assignments $\mathbf{M}_G = ||_{i=1}^L \mathbf{M}_{G_{S_i}}$. We can obtain the required stacked L SMGs with proper assignments of $\mathbf{M}_G$. Then, the problem of capturing hierarchical structures is converted to finding the assignments of $\mathbf{M}_G$.

The flexibility of $\mathbf{M}_G$ enables SMG to extract much richer structural information, beyond the general dense aggregation based GNNs. We have shown that a single layer SMG can only capture subgraphs, while a multi-layer SMG can capture hierarchical structures of graphs. When $L > 1$ and G_{S_i} is restricted to be a subgraph of $G_{S_{i-1}}$ for all $i \in [L-1]$, SMG works in a similar way as top-k based graph poolings [3, 7, 14, 15] that iteratively remove some nodes and related edges layer by layer. However, the interactions of different parts as given in Figure 1 cannot be captured by this kind of scheme, since previously skipped nodes in lower layers will not be involved in higher layers.

3.3 Mask Assignments Computations

In this section, we present the method to compute the assignments of $\mathbf{M}_G$. We use a separate GNN layer, called GNN-w to learn the assignments of $\mathbf{M}_G$. The value assigned on node v in the k-th layer is

$$\begin{aligned}
\mathbf{M}_{Gk,v} &= m_v^{(k)} \\
&= \text{MLP}^{(k)}(\text{ReLU}([\mathbf{L}_1^{(k)}(m_v^{(k-1)}\mathbf{h}_v^{(k-1)})|| \\
&\quad \sum_{u \in \mathcal{N}(v)} \mathbf{L}_2^{(k)}(m_u^{(k-1)}\mathbf{h}_u^{(k-1)})])),
\end{aligned} \quad (4)$$

where $\mathbf{L}_1^{(k)}$ and $\mathbf{L}_2^{(k)}$ are affine maps, MLP is a 2-layer perceptron with output feature dimension of 1. The activation function in

the last layer of MLP is Sigmoid, making $m_v^{(k)} \in (0, 1)$. Note that the constant parts (i.e. bias) in $\mathbf{L}_1^{(k)}$, $\mathbf{L}_2^{(k)}$ and MLP is required. Otherwise, $m_v^{(k-1)} \mathbf{h}_v^{(k-1)} = \mathbf{0}$ and $m_u^{(k-1)} \mathbf{h}_u^{(k-1)} = \mathbf{0}$ will lead $m_v^{(k)}$ always fixed to 0.5.

The value of $m_v^{(k)}$ assigned on each node is from 0 to 1, not discrete mask value 0 or 1 as expected. Fortunately, in a SMG layer, $\mathbf{h}_v^{(k)} = f(\{m_i^{(k)} | i \in \hat{\mathcal{N}}(v)\}) = \text{ReLU}(\mathbf{W}_1^{(k)} m_v^{(k)} [\mathbf{h}_v^{(k-1)} \| \sum_{u \in \mathcal{N}(v)} m_u^{(k)} \mathbf{h}_u^{(k-1)}])$ is continuous for $m_i^{(k)} \in [0, 1]$ where $i \in \hat{\mathcal{N}}(v)$. Therefore we have $\lim_{m_i^{(k)} \to 0^+} f(m_i^{(k)}, ...) = f(0, ...)$ and $\lim_{m_i^{(k)} \to 1^-} f(m_i^{(k)}, ...) = f(1, ...)$ for any $i \in \hat{\mathcal{N}}(v)$, making it possible to use $m_v^{(k)}$ to approximate the mask value 0 or 1.

The benefit of soft-mask is that the weights are taken into consideration. As $\mathbf{h}_v^{(k)}$ represents a k-hop subtree rooted on v that captures the structural information of local k-hop substructure rooted on v, $m_v^{(k)}$ multiplies with $\mathbf{h}_v^{(k)}$ gives the weight of that substructure for the following aggregation operation that takes $\mathbf{h}_v^{(k)}$ as inputs. For nodes with 0 weights, it is equivalent to drop them and related edges.

To maintain zero invariant, we use SUM as Readout function as presented in Theorem 3.2. We also use jumping concatenation as given in [33],

$$\mathbf{h}_G = \overset{K}{\underset{k=1}{\|}} \left(\sum_{i=1}^{N} \mathbf{h}_i^{(k)} \right). \tag{5}$$

Equation 5 utilizes node representations from different layers to compute the entire graph representation $\mathbf{h}_G$. Since a node representation in the k-th layer $\mathbf{h}_v^{(k)}$ encodes the k-hop substructure with the weight $m_v^{(k)}$, when implementing Readout function as Equation 5, the weight of $\mathbf{h}_v^{(k)}$ is considered over all substructures with different hops.

3.4 Multi-channel Soft-mask GNN Model

We generalize the soft-mask mechanism to a multi-channel scenario. To this end, GNN-w computes a mask value for each channel of a node representation such that $m_v^{(k)} \in (0, 1)^d$, where d is the dimension of node representations. This is done by setting the output feature dimension of MLP in Equation 4 to be d. Correspondingly, the SMG layer (Equation 3) is rewritten as

$$\mathbf{h}_v^{(k)} = \text{ReLU}(m_v^{(k)} \odot \mathbf{W}_2^{(k)} [\mathbf{h}_v^{(k-1)} \| \sum_{u \in \mathcal{N}(v)} m_u^{(k)} \odot \mathbf{h}_u^{(k-1)}]), \tag{6}$$

where $\odot$ is element-wise multiplication, and $\mathbf{W}_2^{(k)} \in \mathbf{R}^{d \times d}$. Before the GNN operation in the first layer, we use a linear transformation to make the dimension of the input node features equal to the number of hidden units. Intuitively, multi-channel SMG allows us to conduct sparse aggregation on each channel respectively. General SMG can be regarded as a special case of multi-channel SMG where the values on each channel are the same.

4 DISCUSSION

Comparisons with GAT. In GAT, the attention coefficients of a node are different for its neighbors. This is because, in node level predictions, a node has varying impacts to its neighbors. For graph level predictions, the aim is to embed node features as well as graph topology, and therefore the impact of nodes should be considered over the entire graph. We can see that GAT applies *neighborhood* attention for learning node representations, while SMG applies *global* attention for learning the entire graph representation. Also note that due to the neighborhood normalization in GAT, GAT cannot completely skip some nodes and related edges. In our experiments, we give detailed comparisons about these two kinds of strategies on graph level tasks, even though there are relatively few attempts that apply GAT to graph level predictions.

Comparisons with top-k based poolings. Top-k based poolings [3, 7, 14, 15] compute the attention coefficients among all nodes, making the learned representations focus on informative parts. To handle this, all top-k based methods physically remove nodes and related edges with low attention weights. The drawback of this kind of mechanism is that it requires to manually set the drop-ratio, which can be a limitation for capturing desired subgraphs with arbitrary sizes. Soft-mask mechanism is proposed without this limitation, and therefore SMG should be more adaptive in graphs with arbitrary scales. Furthermore, in top-k based poolings, the learned subgraph of the current layer is the subgraph of the previous layer. This can be a limitation for the model to extract hierarchical structures, as we explained in Section 3.2.

5 EXPERIMENTS

In this section, we evaluate soft-mask GNN and its variants on both graph classification and graph regression tasks. The code is available at https://github.com/qslim/soft-mask-gnn.

5.1 Datasets

For graph classification task, our experiments are conducted on 10 real-world graph datasets from [11, 34], including both bioinformatics datasets and social network datasets. Detailed statistics are given in Table 1. For graph regression task, we use QM9 dataset [23, 24, 30], which is composed of 134K small organic molecules. The task is to predict 12 targets for each molecule. All data is obtained from pytorch-geometric library [6].

5.2 Graph Classification Task

We follow the standard ways to evaluate models on classification datasets which perform 10-fold cross validation and report average accuracy [18, 32, 35]. We use Adam optimizer [12] with learning rate $\in \{0.005, 0.001, 0.0005\}$ and learning rate decay $\in [0.7, 1]$ every $\{50, 100\}$ epochs. Other hyperparameters settings are: (1) the number of hidden units $\in \{32, 64, 128\}$ for bioinformatics datasets, and $\{64, 128, 256\}$ for social network datasets; (2) the number of layers $\in \{2, 3, 4, 5\}$; (3) the batch size $\in \{64, 128\}$; (4) the dropout ratio $\in \{0, 0.5\}$. We run 10 independent times with selected hyperparameters to obtain the final accuracy on each dataset. For graphs without node attributes, we use one-hot encoding of node degrees.

We evaluate the following variants of our proposed soft-mask GNN. **SMG** represents soft-mask GNN with GNN-w implemented

Table 1: Graph classification results. The top 2 performance approaches are highlighted in bold. We report the results in the original papers by default. When the results are not given in the original papers, we report the best testing results given in [10, 31, 36].

dataset	MUTAG	PROTEINS	NCI1	COLLAB	ENZYMES	IMDB-B	IMDB-M	RDT-B	RDT-M5K	RDT-12K
# graphs	188	1113	4110	5000	600	1000	1500	2000	4999	11929
# classes	2	2	2	3	6	2	3	2	5	11
avg # nodes	17.9	39.1	29.9	74.5	32.6	19.8	13.0	429.6	508.5	391.4
GK [26]	81.58±2.11	71.67±0.55	62.49±0.27	72.84±0.28	32.70±1.20	65.87±0.98	43.89±0.38	77.34±0.18	41.01±0.17	31.82±0.08
RW [29]	79.17±2.1	59.57±0.1	NA	NA	24.16±1.64	NA	NA	NA	NA	NA
PK [20]	76±2.7	73.68±0.7	82.54±0.5	NA	NA	NA	NA	NA	NA	NA
WL [25]	84.11±1.9	74.68±0.5	**84.46 ± 0.5**	NA	52.22±1.26	73.40±4.63	49.33±4.75	81.0±3.10	49.44±2.36	38.18±1.30
FGSD [28]	**92.12**	73.42	79.80	80.02	NA	73.62	**52.41**	NA	NA	NA
AWE [10]	87.87±9.76	NA	NA	73.93±1.94	35.77±5.93	74.45±5.83	51.54±3.61	87.89±2.53	50.46±1.91	39.20±2.09
DGCNN [36]	85.83±1.66	75.54±0.94	74.44±0.47	73.76±0.49	51.0±7.29	70.03±0.86	47.83±0.85	NA	NA	NA
PSCN [21]	88.95±4.4	75±2.5	74.44±0.5	73.76±0.5	NA	45.23±2.84	45.23±2.84	86.30±1.58	49.10±0.70	41.32±0.42
DCNN [1]	NA	61.29±1.60	56.61±1.04	52.11±0.71	NA	49.06±1.37	33.49±1.4	NA	NA	NA
ECC [27]	76.11	NA	76.82	NA	45.67	NA	NA	NA	NA	NA
DGK [34]	87.44±2.72	75.68±0.54	80.31±0.46	73.09±0.25	53.43±0.91	66.96±0.96	44.55±0.52	78.04±0.39	41.27±0.18	32.22±0.10
CapsGNN [31]	86.67±6.88	76.28±3.63	78.35±1.55	79.62±0.91	54.67±5.67	73.10±4.83	50.27±2.54	NA	52.88±1.48	46.62±1.90
DiffPool [35]	NA	76.25	NA	75.48	**62.53**	NA	NA	NA	NA	47.08
GIN [32]	89.4±5.6	76.2±2.8	82.7±1.7	80.2±1.9	NA	**75.1 ± 5.1**	52.3±2.8	92.4±2.5	**57.5 ± 1.5**	NA
Top-k Pool [15]	NA	71.86±0.97	67.45±1.11	NA	NA	NA	NA	NA	NA	NA
PPGNN [18]	**90.55 ± 8.7**	**77.2 ± 4.73**	83.19 ± 1.11	80.16±1.11	NA	72.6±4.9	50±3.15	NA	NA	NA
SMG	89.2±6.22	**76.8 ± 4.15**	**83.3 ± 1.88**	**83.2 ± 1.60**	59.0±5.07	74.8±6.63	52.0±4.11	**92.9 ± 2.84**	57.3±2.09	**51.2 ± 1.40**
SMG-JK	89.3±7.12	76.3±4.57	83.0±2.25	**82.7 ± 1.57**	**60.3 ± 5.26**	75.0±6.24	52.3±3.20	**93.1 ± 2.25**	**57.5 ± 1.56**	**51.3 ± 1.52**
M-SMG-JK	89.6±7.38	76.1±4.04	82.8±1.73	82.6±1.54	57.3±5.63	**75.0 ± 5.95**	**52.7 ± 3.71**	91.7±2.08*	57.0±1.85*	49.2±1.13*
Rank	3rd	2nd	2nd	1st	2nd	2nd	1st	1st	1st	1st

by Equation 4. **M-SMG** represents multi-channel soft-mask GNN implemented by Equation 6. **JK** represents the Readout operation implemented by Equation 5.

Comparisons with baselines. Table 1 presents a summary of classification results. Baselines include kernel-based methods and GNN methods. The last row indicates the ranking of our models. M-SMG-JK fails to converge on REDDIT due to a large number of nodes, and therefore their results (marked with *) are computed with the MEAN readout function. Note that SMG based models achieve the top 2 performance in 9 out of 10 datasets. Especially, SMG and SMG-JK are more competitive on large graphs, e.g., COLLAB and three REDDIT datasets. This is because, in a large graph, task-relevant structures should not be characterized in every single detailed structure. It is more efficient to focus on the relevant substructures or the hierarchical structure of a graph.

Representation Power of SMG Models. Figure 3 gives training set performance. Since our proposed soft mask mechanism tries to minimize the impact of irrelevant parts and only learns from the desired subgraph in each layer, we can see that SMG and SMG-JK boost training set performance significantly. Our models fit especially well on large and sparse graphs as given in REDDIT datasets, where GAT fails to fit them. The multi-layer SMG always outperforms SMG-1-layer, except for COLLAB where both models achieve the highest accuracies. This is because graphs in COLLAB have dense connections as visualized in Figure 5 in Appendix C, and exploring 1-hop neighbors can be sufficient to get an overview of graphs. For large and sparse graphs, structural information may be characterized by long-range dependencies that can be captured by exploring larger hops in higher layers. GAT suffers from the impact of irrelevant parts, making them not benefit from the deeper

structures used for capturing long-range dependencies. The performance differences between GAT and SMG show that skipping noisy nodes and related edges makes the model better fit training data.

Visualizing Weight Distributions. We give weights distributions [1] of REDDIT on all layers computed by SMG as shown in Figure 4. More visualization results on other datasets are provided in the Appendix. We interpret weight distributions from two perspectives: (1) How the weights are related to graph structures in a single layer; (2) How weights change over different layers. All graphs are randomly sampled from the corresponding datasets. We use the depth of color to represent the weights of nodes.

From visualizations, the weights of nodes have significant differences and weight distributions do have strong relations to graph structures. Graphs in REDDIT are characterized by tree-like structures, where most nodes lie in the root and are densely connected. This is also reflected from the layout in Figure 4 where nodes close to the root are at the center of the layout and their dense connections form a clique with overlaps. Viewed on different layers, the weights move from leaf parts to the root part. In the first layer, the weights are completely distributed on leaf parts with almost zero weights on the root part, indicating that the learned node representations skip the root part. In higher layers, the weights of the root part are increased. This can be the evidence that the hierarchical structures are captured by SMG. Node representations in the last layer (Layer 4 in Figure 4) are used to compute the final graph representation and the zero weights are not involved in the final graph representation since the Readout operation is zero invariant.

---

[1]Graphs are visualized by graphviz. There are different layout engines provided by Graphviz. For large graphs in REDDIT and COLLAB, we use sfdp to provide a friendly view. For small graphs in PROTEINS and NCI1, we use neato to give detailed structure information.

Figure 3: Training set performance.

Table 2: Mean absolute errors on the QM9 dataset. The top 2 best-performing GNNs are highlighted in bold.

target	μ	α	ϵ_{homo}	ϵ_{lumo}	$\Delta\epsilon$	$\langle R^2 \rangle$	$ZPVE$	U_0	U	H	G	C_v
DTNN [30]	**0.244**	0.95	0.00388	0.00512	0.0112	**17**	0.00172	2.43	2.43	2.43	2.43	0.27
MPNN [8]	0.358	0.89	0.00541	0.00623	0.0066	28.5	0.00216	2.05	2	2.02	2.02	0.42
PPGNN [18]	**0.0934**	**0.318**	**0.00174**	**0.0021**	**0.0029**	**3.78**	0.000399	**0.022**	0.0504	0.0294	0.024	0.144
SMG-JK	0.4709	0.3415	0.0033	0.0036	0.0049	23.64	**0.000236**	0.0248	**0.0247**	**0.0225**	0.0244	**0.130**
M-SMG-JK	0.4395	**0.2899**	**0.0030**	**0.0032**	**0.0045**	21.90	**0.000196**	0.0212	0.0202	0.0214	0.0212	0.1157
Rank	4th	1st	2nd	2nd	2nd	3rd	1st	1st	1st	1st	1st	1st

Weights distributions in Layer 4 show that only some of the node representations participate in the computation of the final graph representation.

Note that the weight differences become less significant in higher layers. This is because the weight of a node actually characterizes the weight of a subtree rooted on that node. In higher layers, the subtrees become larger and have more overlaps, making them similar. On most datasets we test, weight distributions change dynamically in different layers. The substructures assigned with low weights in lower layers can be assigned with high weights in higher layers, which is consistent with our analysis. Note that this kind of phenomenon cannot be captured by top-k based poolings.

5.3 Graph Regression Task

Following the standard dataset splits described in [18, 30], the QM9 dataset is randomly split into 80% train, 10% validation and 10% test. We evaluate SMG-JK and M-SMG-JK with the following parameters: number of layers $\in \{3, 4\}$; hidden units = 64; batch size = 64; learning rate $\in \{0.001, 0.0005, 0.0001\}$; learning rate decay $\in [0.75, 1]$ every $\{20, 30, 50\}$ epochs. Both models are trained for 500 epochs. Table 2 compares the mean absolute error of our methods with state-of-the-art approaches. All results of these approaches are taken from the original papers. Note that our methods achieve the lowest error on 7 out of the 12 targets.

6 CONCLUSION

Motivated by effectively skipping irrelevant parts of graphs, we propose soft-mask GNN (SMG) layer, which learns graph representations from a sequence of subgraphs. We show its capability for explicitly extracting desired substructures or hierarchical structures. Experimental results on benchmark graph classification and graph regression datasets demonstrate that SMG gains significant improvements and the visualizations of masks provide interpretability of structures learned by the model.

ACKNOWLEDGMENTS

REFERENCES

[1] James Atwood and Don Towsley. 2016. Diffusion-convolutional neural networks. In *Advances in Neural Information Processing Systems*. 1993–2001.
[2] Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. 2017. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine* 34, 4 (2017), 18–42.
[3] Cătălina Cangea, Petar Veličković, Nikola Jovanović, Thomas Kipf, and Pietro Liò. 2018. Towards sparse hierarchical graph classifiers. *arXiv preprint arXiv:1811.01287* (2018).
[4] Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, and Xu Sun. 2020. Measuring and Relieving the Over-Smoothing Problem for Graph Neural Networks from the Topological View.. In *AAAI*. 3438–3445.

Figure 4: Weight distributions of 4 graphs sampled from REDDIT-BINARY, REDDIT-MULTI-5K and REDDIT-MULTI-12K.

[5] Zhengdao Chen, Soledad Villar, Lei Chen, and Joan Bruna. 2019. On the equivalence between graph isomorphism testing and function approximation with gnns. In *Advances in Neural Information Processing Systems*. 15868–15876.

[6] Matthias Fey and Jan E. Lenssen. 2019. Fast Graph Representation Learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*.

[7] Hongyang Gao and Shuiwang Ji. 2019. Graph U-Nets. *arXiv preprint arXiv:1905.05178* (2019).

[8] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. 2017. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 1263–1272.

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

[10] Sergey Ivanov and Evgeny Burnaev. 2018. Anonymous Walk Embeddings. In *Proceedings of the 35th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 80)*, Jennifer Dy and Andreas Krause (Eds.). PMLR, Stockholmsmässan, Stockholm Sweden, 2191–2200. http://proceedings.mlr.press/v80/ivanov18a.html

[11] Kristian Kersting, Nils M. Kriege, Christopher Morris, Petra Mutzel, and Marion Neumann. 2016. Benchmark Data Sets for Graph Kernels. http://graphkernels.cs.tu-dortmund.de.

[12] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[13] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[14] Boris Knyazev, Graham W Taylor, and Mohamed R Amer. 2019. Understanding attention in graph neural networks. *arXiv preprint arXiv:1905.02850* (2019).

[15] Junhyun Lee, Inyeop Lee, and Jaewoo Kang. 2019. Self-Attention Graph Pooling. *arXiv preprint arXiv:1904.08082* (2019).

[16] John Boaz Lee, Ryan Rossi, and Xiangnan Kong. 2018. Graph classification using structural attention. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 1666–1674.

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

[18] Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. 2019. Provably Powerful Graph Networks. *arXiv preprint arXiv:1905.11136* (2019).

[19] Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. 2019. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 4602–4609.

[20] Marion Neumann, Roman Garnett, Christian Bauckhage, and Kristian Kersting. 2016. Propagation kernels: efficient graph kernels from propagated information. *Machine Learning* 102, 2 (2016), 209–245.

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

[22] Kenta Oono and Taiji Suzuki. 2019. Graph neural networks exponentially lose expressive power for node classification. *arXiv preprint arXiv:1905.10947* (2019).

[23] Raghunathan Ramakrishnan, Pavlo O Dral, Matthias Rupp, and O Anatole Von Lilienfeld. 2014. Quantum chemistry structures and properties of 134 kilo molecules. *Scientific data* 1 (2014), 140022.

[24] Lars Ruddigkeit, Ruud Van Deursen, Lorenz C Blum, and Jean-Louis Reymond. 2012. Enumeration of 166 billion organic small molecules in the chemical universe database GDB-17. *Journal of chemical information and modeling* 52, 11 (2012), 2864–2875.

[25] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. 2011. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research* 12, Sep (2011), 2539–2561.

[26] Nino Shervashidze, SVN Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten Borgwardt. 2009. Efficient graphlet kernels for large graph comparison. In *Artificial Intelligence and Statistics*. 488–495.

[27] Martin Simonovsky and Nikos Komodakis. 2017. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 3693–3702.

[28] Saurabh Verma and Zhi-Li Zhang. 2017. Hunt for the unique, stable, sparse and fast feature learning on graphs. In *Advances in Neural Information Processing Systems*. 88–98.

[29] S Vichy N Vishwanathan, Nicol N Schraudolph, Risi Kondor, and Karsten M Borgwardt. 2010. Graph kernels. *Journal of Machine Learning Research* 11, Apr (2010), 1201–1242.

[30] Zhenqin Wu, Bharath Ramsundar, Evan N Feinberg, Joseph Gomes, Caleb Geniesse, Aneesh S Pappu, Karl Leswing, and Vijay Pande. 2018. MoleculeNet: a benchmark for molecular machine learning. *Chemical science* 9, 2 (2018), 513–530.

[31] Zhang Xinyi and Lihui Chen. 2019. Capsule Graph Neural Network. In *International Conference on Learning Representations*. https://openreview.net/forum?id=Byl8BnRcYm

[32] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How Powerful are Graph Neural Networks?. In *International Conference on Learning Representations*. https://openreview.net/forum?id=ryGs6iA5Km

[33] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. 2018. Representation learning on graphs with jumping knowledge networks. *arXiv preprint arXiv:1806.03536* (2018).

[34] Pinar Yanardag and SVN Vishwanathan. 2015. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 1365–1374.

[35] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. 2018. Hierarchical graph representation learning with differentiable pooling. In *Advances in Neural Information Processing Systems*. 4800–4810.

[36] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. 2018. An end-to-end deep learning architecture for graph classification. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

A   PROOF OF LEMMA 3.1

PROOF. When $K=1$, for any $v \in V_{G_S}$, $m_v^{(1)} = 1$ and

$$\mathbf{h}_v^{(1)} = \text{ReLU}(\mathbf{W}_1^{(1)}\mathbf{x}_v + \sum_{u \in \mathcal{N}_G(v) \cap V_{G_S}} m_u^{(1)}\mathbf{W}_2^{(1)}\mathbf{x}_u +$$
$$\sum_{u \in \mathcal{N}_G(v) \cap (V_G/V_{G_S})} m_u^{(1)}\mathbf{W}_2^{(1)}\mathbf{x}_u).$$

To prove $\sum_{u \in \mathcal{N}_G(v) \cap (V_G/V_{G_S})} m_u^{(1)}\mathbf{W}_2^{(1)}\mathbf{x}_u = \mathbf{0}$, we only need to consider the case that $\mathcal{N}_G(v) \cap (V_G/V_{G_S}) \neq \emptyset$. For any $u \in \mathcal{N}_G(v) \cap (V_G/V_{G_S})$, we have (i) $\{u\} \subseteq V_G/V_{G_S}$; (ii) $\mathcal{N}_G(u) \cap V_{G_S} \neq \emptyset$, since $\{v\} \subseteq \mathcal{N}_G(u) \cap V_{G_S}$; (iii) $K\%2 = 1$. According to given condition of $\mathbf{M}$, $m_u^{(1)} = 0$. Then we have

$$\mathbf{h}_v^{(1)} = \text{ReLU}(\mathbf{W}_1^{(1)}\mathbf{x}_v + \sum_{u \in \mathcal{N}_G(v) \cap V_{G_S}} \mathbf{W}_2^{(1)}\mathbf{x}_u + \mathbf{0})$$
$$= \text{ReLU}(\mathbf{W}_1^{(1)}\mathbf{x}_v + \sum_{u \in \mathcal{N}_{G_S}(v)} \mathbf{W}_2^{(1)}\mathbf{x}_u)$$
$$= \mathbf{t}_v^{(1)}.$$

Suppose when $K = k - 1$, for any $v \in V_{G_S}$, $\mathbf{h}_v^{(k-1)} = \mathbf{t}_v^{(k-1)}$ holds. For any $v \in V_{G_S}$,

$$\mathbf{h}_v^{(k)} = \text{ReLU}(\mathbf{W}_1^{(k)}\mathbf{h}_v^{(k-1)} + \sum_{u \in \mathcal{N}_G(v) \cap V_{G_S}} m_u^{(k)}\mathbf{W}_2^{(k)}\mathbf{h}_u^{(k-1)}$$
$$+ \sum_{u \in \mathcal{N}_G(v) \cap (V_G/V_{G_S})} m_u^{(k)}\mathbf{W}_2^{(k)}\mathbf{h}_u^{(k-1)}).$$

If $k\%2 = 1$, $m_u^{(k)} = 0$ for any $u \in \mathcal{N}_G(v) \cap (V_G/V_{G_S})$ (the same as $K=1$) and $\sum_{u \in \mathcal{N}_G(v) \cap (V_G/V_{G_S})} m_u^{(k)}\mathbf{W}_2^{(k)}\mathbf{h}_u^{(k-1)} = \mathbf{0}$. If $k\%2 = 0$, then $(k-1)\%2 = 1$, $m_u^{(k-1)} = 0$ for any $u \in \mathcal{N}_G(v) \cap (V_G/V_{G_S})$ (the same as $K=1$), thus $\mathbf{h}_u^{(k-1)} = \mathbf{0}$ and $\sum_{u \in \mathcal{N}_G(v) \cap (V_G/V_{G_S})} m_u^{(k)}\mathbf{W}_2^{(k)}\mathbf{h}_u^{(k-1)} = \mathbf{0}$. Therefore, we have

$$\mathbf{h}_v^{(k)} = \text{ReLU}(\mathbf{W}_1^{(k)}\mathbf{h}_v^{(k-1)} + \sum_{u \in \mathcal{N}_G(v) \cap V_{G_S}} m_u^{(k)}\mathbf{W}_2^{(k)}\mathbf{h}_u^{(k-1)})$$
$$= \text{ReLU}(\mathbf{W}_1^{(k)}\mathbf{t}_v^{(k-1)} + \sum_{u \in V_{G_S}} m_u^{(k)}\mathbf{W}_2^{(k)}\mathbf{t}_u^{(k-1)})$$
$$= \mathbf{t}_v^{(k)}.$$

Finally, we can conclude that $\forall v \in V_{G_S}, \mathbf{h}_v^{(K)} = \mathbf{t}_v^{(K)}$.   □

B   PROOF OF THEOREM 3.2

PROOF. From Lemma 3.1, the problem is converted to find the assignments of $\mathbf{M}$ such that $\mathbf{h}_v^{(K)} = \mathbf{0}$ for any $v \in V_G/V_{G_S}$. Meanwhile, the assignments of $\mathbf{M}$ should be consistent with that in Lemma 3.1. Thus, a simple assignments of $\mathbf{M}$ is

$$\mathbf{M}_{k,v} = m_v^{(k)} = \begin{cases} 0 & \{v\} \subseteq V_G/V_{G_S} \\ & \wedge((\mathcal{N}_G(v) \cap V_{G_S} \neq \emptyset \\ & \wedge k\%2 = 1) \vee k = K) \\ 1 & \{v\} \subseteq V_{G_S} \wedge \{k\} \subseteq [K]. \end{cases}$$

Note that

$$\mathbf{h}_G = \text{SUM}(\{\{\mathbf{h}_v^{(K)} | v \in V_G\}\})$$

$$= \text{SUM}(\{\{\mathbf{h}_v^{(K)} | v \in V_{G_S}\}\} \cup \{\{\mathbf{h}_v^{(K)} | v \in V_G/V_{G_S}\}\}),$$

where $\{\{\mathbf{h}_v^{(K)} | v \in V_G/V_{G_S}\}\} = \{\{\mathbf{0}\}\}$. According to Lemma 1, we have

$$\mathbf{h}_G = \text{SUM}(\{\{\mathbf{h}_v^{(K)} | v \in V_{G_S}\}\})$$

$$= \text{SUM}(\{\{\mathbf{t}_v^{(K)} | v \in V_{G_S}\}\})$$

$$= \mathbf{t}_{G_S}.$$

□

C MORE VISUALIZATION RESULTS

Graphs in COLLAB have dense connections. For nodes with a large number of neighbors, their values of representation vectors may increase rapidly. Therefore, in a general GNN, these nodes will dominate the entire graph representation. Figure 5 shows that in a soft-mask GNN, nodes with many neighbors learn relative low weights to restrict the fast growth of values of representation vectors.

For small graphs in NCI1 as given in Figure 6, task-relevant structures should be considered in the entire graph. The weight distributions in different layers show that the aggregation operation is conducted on most of the nodes, but with different weights. Graphs in PROTEINS as given in Figure 7 have node attributes, thus the weight distributions are also related to node attributes. Nodes with 0 weights in the first layer mean that node attributes are not included in the final graph representation, which provides a way to find out task-relevant node attributes.

Figure 5: Weight distributions of 2 graphs sampled from COLLAB.

Figure 6: Weight distributions of 2 graphs sampled from NCI1.

Figure 7: Weight distributions of 2 graphs sampled from PROTEINS.