Adaptive Hierarchical Graph Neural Networks

Anonymous Author(s)
Affiliation
Address
email

Abstract

Graph Neural Networks (GNNs) have been increasingly deployed in a multitude of different applications that involve node-wise and graph-level tasks. The existing literature usually studies these questions independently while they are inherently correlated. We propose in this work a unified model AdaHGNN to learn node and graph level representation interactively. Compared with the existing GNN models and pooling methods, AdaHGNN enhances node representation with hierarchical semantics and avoids node feature and graph structure information losing during pooling. More specifically, we design a differentiable pooling operator to obtain a hierarchical structure that involves node-wise and meso/macro level semantic information. Besides, we propose the unpooling and *flyback* aggregators to use the hierarchical semantics to enhance node representation. The updated node representation could further enrich the generated graph representation in the next iteration. Experimental results on twelve real-world graphs demonstrate the effectiveness of AdaHGNN on multiple tasks, compared with several competing methods. In addition, the ablation and empirical studies confirm the function of different components of AdaHGNN.

1 Introduction

In many real-world applications, such as social networks, recommendation systems, and biological protein-protein networks, data can be naturally organised as graphs [7]. Nevertheless, how to work with this powerful data representation remains a challenge, since it requires integrating the rich inherent attributes and complex structural information. To address this challenge, Graph Neural Networks (GNNs), which generalises deep neural networks to graph-structured data, had drawn remarkable attention from academia and industry, and achieved state-of-the-art performances in a multitude of applications [16, 26]. The current literature on GNNs can be grouped into two categories. While some of them focus on learning node-level representations to perform tasks such as link prediction [9, 23], node classification [10, 18] and node clustering [25, 2], the others focus on learning graph-level representations for tasks, such as graph classification [20, 6, 22].

In a nutshell, the existing GNN models for node representation generation rely on a similar methodology that utilises a GNN layer to aggregate the sampled neighbouring node's features in a number of iterations, via non-linear transformation and aggregation functions. Its effectiveness has been widely proved, however, a major limitation of these GNN models is that they are inherently *flat* as they only propagate information across the observed edges of the graph. Thus, they lack the capacity to encode features in the high-order neighbourhood in the graphs [21, 1]. For example, in a citation network, *flat* GNN models could capture the micro relationships (e.g., co-authorships) between authors, but neglect their macro relationships (e.g., belonging to different research institutes).

On the other hand, the task of graph classification is to predict the label associated with an entire graph by utilising the given graph structure and initial node-level representations. Nevertheless, existing GNNs for graph classification are unable to infer and aggregate this information in a *hierarchical*

Submitted to 34th Conference on Neural Information Processing Systems (NeurIPS 2020). Do not distribute.

manner, which is crucial to better encode meso- and macro-level graph semantics hidden in the graph. To remedy this limitation, novel pooling approaches have been recently proposed where sets of nodes are recursively aggregated to form hyper-nodes in the pooled graph. DIFFPOOL [20] is a differentiable pooling operator but its assignment matrix is too *dense* [4] to apply on large graphs. TOPKPOOL [6], SAGPOOL [11], ASAP [14] and STRUCTPOOL [22] are four recently proposed methods that adopt the Top-k selection strategy to address the sparsity concerns of DIFFPOOL. They score nodes based on a learnable projection vector and select a fraction of high scoring nodes as hyper-nodes. However, the predefined pooling ratio limits the adaptivity of these models on graphs of different sizes, and the Top-k selection may easily lose important node features or graph structure by drooping low scoring nodes. Moreover, we argue that node-wise and graph-level tasks are inherently correlated that node representations form graph representation and graph representation could enrich node representation with meso/macro-level knowledge of the graph. It allows GNNs to overcome the limitation of *flat* propagation mode in capturing hierarchical semantics and the enriched node representation could further ameliorate the graph representation.

In this work, we propose a novel framework, *Adaptive Hierarchical Graph Neural Networks* (AdaHGNN), which integrates graph convolution and adaptive pooling operations into one framework to interactively generate both node and graph level representations. Different from the above-mentioned GNN models, we treat node and graph representation generation tasks in a unified framework, and argue that they can collectively optimise each other during training. In the hierarchical structure construction, an adaptive pooling operator preserves the important node features and topology structure based on a novel selection strategy. Beside, AdaHGNN could provide explainable results in terms of the scope of the graph, instead of only considering local neighbours.

More concretely, as shown in Figure 1-(a), we employ (i) an adaptive graph pooling operators to construct a hierarchical structure based on the generated primary node representation by a GNN layer, (ii) graph unpooling operators to further distribute the explored meso- and macro-level semantics to the corresponding nodes of the original graph and (iii) a *flyback* to finally integrate all received messages as the evolved node representations. The proposed hyper-node construction approach enables a true adaptive hierarchical structure construction process without losing neither important node features nor graph structure information. Besides, the attention-enhanced *flyback* aggregator provides reasonable explanation in terms of the importance of messages from different levels. Experimental results reveal the effectiveness of our model and the ablation and empirical studies confirm the function of different components of our model.

Our contributions can be summarised as follows: (1) we propose a novel framework AdaHGNN that integrates node and graph level tasks in one unified process and achieves mutual optimisation between them; (2) AdaHGNN proposes an adaptive and efficient pooling operator to construct the hierarchical structure without introducing any hyper-parameters; (3) we make the first attempt to give reasonable explanations in terms of the scope of the graph (instead of only considering local neighbours); (4) extensive experiments on twelve real datasets demonstrate the promising performance of AdaHGNN.

2 Related Work

Graph neural networks. The existing GNN models can be generally categorised into spectral and spatial approaches. The spectral approach utilises the Fourier transformation to define convolution operation in the graph domain [3]. However, its incurred heavy computation cost hinders it from being applied to large-scale graphs. Later on, a series of spatial models drawn remarkable attention due to their effectiveness and efficiency in node-wise tasks [10, 7, 15, 18], such as link prediction, node classification and node clustering. They mainly rely on the *flat* message-passing mechanism that defines convolution by iteratively aggregating messages from the neighbouring nodes. Recent studies have proved that the spatial approach is a special form of Laplacian smoothing and is limited to summarising each node's local information [13, 5]. Besides, they are either unable to capture global information or incapable of aggregating messages in a hierarchical manner to support graph classification tasks.

Graph pooling. Pooling operation overcomes GNN's weakness in generating graph-level representation by recursively merge sets of nodes to form hyper-nodes in the pooled graph. DIFFPOOL [20] is a differentiable pooling operator that learns a soft assign matrix that maps each node to a set of clusters to form hyper-nodes. Since this assignment is rather *dense*, that incurs high computation

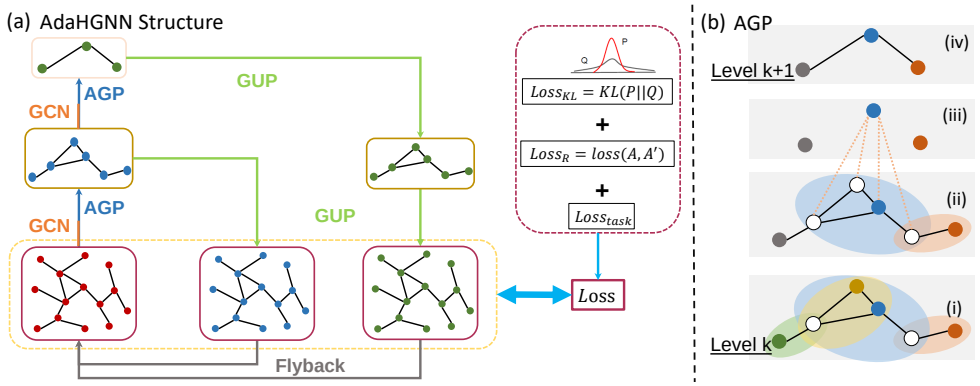

Figure 1: (a) An illustration of AdaHGNN with 3 levels. AGP: adaptive graph pooling, GUP: graph unpooling. (b) An example of performing adaptive graph pooling on a graph: (i) ego-network formation, (ii-iii) hyper-node generation, (iv) maintaining hyper-graph connectivity.

cost, it is not scalable for large graphs [4]. Following this direction, a Top-k based pooling layer (TOPKPOOL) is proposed to select important nodes from the original graph to build a pooled graph [6]. SAGPOOL [11] and ASAP [14] further use attention and self-attention for cluster assignment. They address the problem of sparsity in DIFFPOOL, but they drop either important node features or the rich graph structure in the non-selected nodes, mainly due to the Top-k selection strategy. The introduced hyper-parameter k has been proved significant to the performances (see Appendix A.1), and it limits the adaptivity of these models on graphs of different sizes. Lately, similar to DIFFPOOL, STRUCTPOOL [22] designs strategies to involve both node features and graph structure, and includes conditional random fields technique to ameliorate the cluster assignment. However, STRUCTPOOL treats the graph assignment as a *dense* clustering problem, which gives rise to a high computation complexity as in DIFFPOOL.

3 Proposed Approach

3.1 Preliminaries

An attributed graph with n nodes can be formally represented as $G = (V, E, X)$, where $V = \{v_1, \ldots, v_n\}$ is the node set, $E \subseteq V \times V$ donets the set of edges and $X \in \mathbb{R}^{n \times \pi}$ represents nodes' features (π is dimension of node features). Its adjacency matrix can present as: $A \in \{0, 1\}^{n \times n}$.

For node-wise tasks, e.g., link prediction and node classification, the goal is to learn a mapping function $f_n : G \rightarrow H$, where $H \in \mathbb{R}^d$ and each row $h_i \in H$ corresponds to the node v_i's representation. For graph-level task, e.g., graph classification, similarly it aims to learn a mapping $f_g : D \rightarrow H$, where $D = \{G_1, G_2, \ldots\}$ is a set of graphs, each row $h_i \in H$ corresponds to the graph G_i's representation. The effectiveness of the mapping function f_n and f_g is evaluated by applying H to different tasks.

Primary node representation. We use Graph Convolution Network (GCN) [10] to obtain the node representation: $H^{\ell+1} = \text{ReLU}(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{\frac{1}{2}}H^{\ell}W^{\ell})$, where $\hat{A} = A + I$, $\hat{D} = \sum_j \hat{A}_{ij}$ and $W^{\ell} \in \mathbb{R}^d$ is a trainable weight matrix for layer ℓ. H^{ℓ} is the generated node representation of layer ℓ as the initial node representation $H^0 = X$. In this work, we use one GCN layer to accelerate the process.

This node representation is generated based on each target node's local neighbours as they aggregate information following the adjacency matrix A. However, GCN cannot capture meso/macro level knowledge, even with stacking multiple layers. Hence we call this generated node representation as primary node representation.

3.2 Adaptive Graph Pooling for Hierarchical Structure Construction

Our proposed model, AdaHGNN, addresses the above challenges by adaptively constructing a hierarchical structure to realise the collective optimisation of the node and graph level tasks within one unified framework. The key intuition is that applying an adaptive graph pooling operator to

explicitly present the hierarchical semantics of G and improve the node representation generation with the explored semantic information. Usually, AdaHGNN has multiple hierarchies (K hierarchy levels), in this section, we present how level k's hyper graph is adaptively constructed based on graph of level $k-1$, i.e., $G_{k-1} = (V_{k-1}, E_{k-1}, X_{k-1})$. The number of levels of AdaHGNN is treated as a hyper-parameter to be discussed in Appendix A.4 and A.5. Please refer to Algorithm 1 in Appendix A.7 for a pseudo code of the working of AdaHGNN.

Ego-network formation. We initially consider the graph pooling as an ego-network selection problem, as each ego node only consider whether to aggregate its local neighbours to form a hyper node, resolving the *dense* issue of DIFFPOOL. As shown in Figure 1-(b)-(i), each ego-network c_λ contains the ego and its local neighbours $\mathcal{N}_i^\lambda$ within λ-hops., i.e., $\mathcal{N}_i^\lambda = \{v_j \mid if\ d(v_i, v_j) \leq \lambda\}$, where $d(v_i, v_j)$ means the distance between v_i and v_j. Thus an ego-network with ego node v_i can be formally presented as: $c_\lambda(v_i) = \{v_j \mid \forall v_j \in \mathcal{N}_i^\lambda\}$, and a list of ego-networks $C_\lambda = \{c_\lambda(v_1), \ldots, c_\lambda(v_n)\}$ can be formed from the graph G.

Hyper-node generation. A graph G with n nodes has n ego-networks, forming hyper graph with all ego-networks will blur the useful hierarchical semantics and lead to a high computation cost. Hence, we need to select a fraction of ego-networks to present the hierarchical semantics of G. We do the selection based on a fitness score ϕ_i that evaluates the relation strengths of the ego v_i to its local neighbours $v_j \in c_\lambda(v_i)$. One ego-network's fitness score is decided by the scores between each node and the ego, thus we define a function f_ϕ to calculate the fitness score ϕ_{ij} between v_i and v_i:

計算兩點間的fitness score

$$f_\phi(v_i, v_j) = f_\phi^s(v_i, v_j) \times f_\phi^c(v_i, v_j) = \text{Softmax}(\overrightarrow{a}^{\text{T}}\, \sigma(Wh_j \parallel Wh_i)) \times \text{Sigmoid}(h_j^{\text{T}} \cdot h_i), \quad (1)$$

where $\overrightarrow{a} \in \mathbb{R}^{2\pi}$ is the weight vector, $\parallel$ is the concatenation operator and σ is an activation function (LeakReLU). $f_\phi^s(v_i, v_j) = \text{Softmax}(\overrightarrow{a}^{\text{T}}\, \sigma(Wh_j \parallel Wh_i)) = \frac{exp(\overrightarrow{a}^{\text{T}}\, \sigma(Wh_j \| Wh_i))}{\sum_{v_r \in \mathcal{N}_j^\lambda} exp(\overrightarrow{a}^{\text{T}}\, \sigma(Wh_j \| Wh_r))}$ calculates one component of ϕ_{ij} considering node features and graph structure information summarised in the node representation h, and its output lies in $(0, 1)$ as a valid probability for ego-network selection. Meanwhile, inspired by [8] which has demonstrated the importance the linearity relation between two features, we further add another component $f_\phi^c(v_i, v_j) = \text{Sigmoid}(h_j^{\text{T}} \cdot h_i)$ to supercharge f_ϕ with the linearity between node v_j and ego v_i. As a consequence, nodes have similar features and structure information to ego will have higher fitness scores. After, we summarise the fitness score of ego-network $c_\lambda(v_i)$ as: $\phi_i = \frac{1}{|\mathcal{N}_i^\lambda|} \sum_{v_j \in \mathcal{N}_i^\lambda} \phi_{ij}$, where $|\mathcal{N}_i^\lambda|$ indicates the number of nodes in $\mathcal{N}_i^\lambda$.

加總中心點和所有鄰居點的fitness score

After obtaining fitness scores of ego-networks, we propose an approach to select a fraction of ego-networks to form hyper nodes. It is a truly adaptive selection strategy without the need for predefined hyper-parameters and avoiding the limitations of Top-k selection strategy [6], as described in Appendix A.1. Our key intuition is that a big ego-network, that should be merged as a hyper node, could be composed of multiple small ego-networks. Therefore, we intend to firstly find proper small ego-networks, then recursively aggregate them to form a big hyper node that contains all of these small ego-networks. Specifically, we select ego-networks by selecting a fraction of egos $\hat{N}_p$ as: $\hat{N}_p = \{v_i \mid \phi_i > \phi_j, \forall v_j \in \mathcal{N}_i^1\}$, where $\mathcal{N}_i^1$ means the neighbour nodes of node v_i within one hop. In order to allow overlap between different selected ego-networks, we utilise $\mathcal{N}_i^1$ instead of $\mathcal{N}_i^\lambda$. Following this, we can select a fraction of ego-networks to form hyper nodes at level k.

Proposition 1. *Let G be a connected graph with n nodes, and n ego-networks can be formed from the graph G, i.e., $C_\lambda = \{c_\lambda(v_1), c_\lambda(v_2), \ldots, c_\lambda(v_n)\}$. Each ego-network $c_\lambda(v_i)$ will be assigned with a fitness score ϕ_i. Then, there exist at least one ego-network $c_\lambda(v_i)$ satisfies $\phi_i > \phi_j$, $\forall v_j \in \mathcal{N}_i^1$.*

Proof. See Appendix A.6 for the proof. □

Meanwhile, we would also retain nodes that do not belong to any selected ego-networks to maintain the structure of graph: $\hat{N}_r = \{v_j \mid v_j \notin c_\lambda(v_i), \forall v_i \in \hat{N}_p\}$, In this way, a hyper node formation matrix $S_k \in \mathbb{R}^{n \times (|\hat{N}_c| + |\hat{N}_t|)}$ can be formed, where $(|\hat{N}_p| + |\hat{N}_r|)$ is number of nodes of the generated hyper graph, rows of S_k corresponds to the n nodes of G_{k-1} and columns of S_k corresponds to the selected ego-networks ($\hat{N}_p$) or the remaining nodes ($\hat{N}_c$). We have $S_k[i, j] = \phi_{ij}$ if node v_i belongs to the selected ego-network $c_\lambda(v_i)$ and $S_k[i, j] = 1$ if node v_j is a remaining node corresponds to node v_i in the hyper graph otherwise $S_k[i, j] = 0$. The weighted hyper node formation matrix S_k can better maintain the relation between different hyper nodes in the pooled graph.

Maintaining hyper-graph connectivity. As shown in Figure 1-(b)-(iii-iv), after selecting the ego-networks and retaining nodes in level $k-1$, we construct the new adjacent matrix A_k for the hyper graph using $\hat{A}_{k-1}$ and S_k as follows: $A_k = S_k^{\mathrm{T}} \hat{A}_{k-1} S_k$. This formula makes any two hyper nodes connected if they share any common nodes or any two nodes are already neighbours in G_{k-1}. In addition, A_k will retain the edge weights passed by S_{k-1} that involves the relation weights between different hyper nodes. At last, we obtain a generated hyper graph $G_k = (V_k, E_k, X_k)$ at level k.

Hyper-node feature initialisation. All nodes in the hypergraph G_k need an initial feature vector to support the graph convolution operation. For the remaining nodes $\hat{N}_r$ that do not belong to any hyper nodes, we could keep its representation of H_{k-1} as its initial node feature at level k. Given a generated hyper node v_i at level k, we argue that a hyper node's initial feature should contain the ego's representation and other nodes $v_j \in \mathcal{N}_\lambda(v_i)$ belong to the ego-network as well. Recall that we have the fitness score as calculated in Eq. 1, between node v_j to the ego v_i. However, this is not equivalent to the contribution of node v_j's feature to the hyper node feature, since we need to compare the relation strength between node v_j and ego v_i with the relation between ego v_i and other $v_r \in c_\lambda(v_i)$. Therefore, we further propose a hyper node feature initialisation method through a self-attention mechanism. It calculates the contribution of node v_j to the ego v_i according to the node representations. And we further aggregate all weighted node representations as the hyper node's initial feature. Specifically, it can be described as following:

$$X_k(i,:) = H_{k-1}(i,:) + \sum_{v_j}^{c_\lambda(v_i)\setminus v_i} \alpha_{ij} H_{k-1}(j,:), \tag{2}$$

where H_{k-1} is the generated node representation by $(k-1)$-th GNN layer at level $k-1$, α_{ij} describes the contribution of node v_j to the initial feature of $c_\lambda(v_i)$ at level k. And α_{ij} can be learnt as follows: $\alpha_{ij} = \frac{exp(\vec{a}^{\mathrm{T}}\, \sigma(W(\phi_{ij} \cdot h_j) \| h_i))}{\sum_{v_r \in c_\lambda(v_i)} exp(\vec{a}^{\mathrm{T}}\, \sigma(W(\phi_{ir} \cdot h_r) \| h_i))})$. Therefore, hyper node's initial feature contains ego v_i's representation $H_{k-1}(i,:)$ and other nodes' weighted representations.

3.3 Graph Unpooling

Different from the existing graph pooling models [20, 4, 11, 14, 22], we aim to treat node and graph level tasks in a unified framework, thus we further design a mechanism to allow the learned hierarchical semantics to enrich the node representations of the original graph G as shown in Figure 1-(a). Vice versa, the updated node representation can further ameliorate the graph representation in the next training iteration. However, a reasonable unpooling operation has not been well studied in the literature. For instance, Gao et al. [6] tried to directly relocate the hyper node back into the original graph and utilise other GNN layers to spread its message. However, these additional aggregation operations are computationally costly.

We treat the unpooling process as a special message-passing problem and design a suitable pipeline to allow the *top-down* message-passing mechanism. It overcomes the limitations of the existing *flat* message-passing GNN models and endows GNN models with meso/macro level knowledge. Specifically, we utilise S_k to restore the generated ego-network representation of level k to level $k-1$ until we arrive at the original graph G, i.e., $k \to 0$, as follows: $\hat{H}_k = (S_1 \dots (S_k(S_{k-1} H_k)))$, where $\hat{H}_k \in \mathbb{R}^{n \times d}$. At the end of each iteration, nodes of the original graph G will receive a list of messages from the high levels $\{\hat{H}_1, \dots, \hat{H}_k\}$.

3.4 Flyback Aggregation

Since the hyper graphs at different levels presents different level's semantic, we design a new attention score enhanced mechanism to integrate the hierarchical semantics: $H = H_0 + \sum_i^k \beta_i \hat{H}_i$, where the attention score β_k, that estimates the importance of received message from level k, is calculated as: $\beta_k(v_i) = \frac{\exp(\vec{a}^{\,T}\, \sigma(W H_k(v_i) \| H_0(v_i)))}{\sum_k \exp(\vec{a}^{\,T}\, \sigma(W \hat{H}_j(v_i) \| H_0(v_i)))}$. We term this process as the *flyback* aggregator, which not only considers the attention scores of different levels but also provides reasonable explanations about how AdaHGNN utilises the hierarchical semantics to enhance the node representations. We will discuss the explainability of AdaHGNN in Section 4.2.

3.5 Training Strategy

Two challenges remain when training the model. The first is how to distinguish nodes' representation between different ego-networks, since nodes belong to one common ego-networks share similar features according to the calculation of fitness score Eq. 1, thus they should be close to each other in the representation latent space. To address this problem, we introduce a self-optimisation strategy [17], that brings nodes of the same ego-network close and let nodes of different ego-networks far away. to enhance the distinction between different ego-networks. Therefore, apart from the task-related loss function $\mathcal{L}_{task}$, we further input the obtained representations into a self-optimising algorithm to strengthen each ego-network: $\mathcal{L}_{KL} = KL(P \parallel Q) = \sum_{v_j} \sum_{v_i} p_{ij} log \frac{p_{ij}}{q_{ij}}$, where q_{ij} measures the similarity between node representation h_j and ego representation h_i. We measure it with Student's t-distribution so that it could handle different scaled ego-networks and is computationally convenient [17, 2]: $q_{ij} = \frac{(1+||h_j - h_i||^2/\mu)^{-1}}{\sum_{v_{i'}} (1+||h_j - h_{i'}||^2/\mu)^{-1}}$, where v'_i presents all egos of $\hat{N}_p$ and μ is the degrees of freedom of the Student's t-distribution. Here, we choose $\mu = 1$ for all experiments and treat $Q = [q_{ij}]$ as the distribution of ego-network formation distribution. On the other hand, p_{ij} is the target distribution P by first raising q_{il} to the second power and then normalising by frequency per ego-network: $p_{ij} = \frac{q_{ij}^2/g_i}{\sum_{v_{i'}} (q_{ij}^2/g_{i'})}$, where $g_i = \sum_{v_i} q_{ij}$ are soft ego-network frequencies. In the target distribution P, each ego-network formation in Q is squared and normalised so that the formations will have higher confidence. By minimising the KL divergence loss between Q and P distribution, $\mathcal{L}_{KL}$ then forces the current distribution Q to approach the target distribution P, so the connection of nodes of ego-networks will be further strengthened and differentiated from other ego-networks.

The second challenge is to avoid the over-smoothing problem. GNN is proved as a special form of Laplacian smoothing [13, 5] which naturally assimilates the nearby nodes' representations, and our unpooling operation will further exacerbate this problem due to the fact that it distributes information of hyper node representation to all nodes of the ego-network. To address this challenge, we introduce the reconstruction loss, which could drive the node representations to retain the topology structure information of G, to avoid the over-smoothing. Specifically, the loss function is defined as: $\mathcal{L}_R = \sum_n loss(A_{i,j}, A'_{i,j})$, where $A' = Sigmoid(H^T H)$. Therefore, our proposed training strategy uses the following loss function:

$$\mathcal{L} = \mathcal{L}_{task} + \gamma \mathcal{L}_{KL} + \delta \mathcal{L}_R, \tag{3}$$

where $\mathcal{L}_{task}$ is a flexible task-specific loss function, and γ and δ are two hyper-parameters that we will present them in Appendix A.4. Note that for link prediction task we have $\mathcal{L} = \mathcal{L}_R + \gamma \mathcal{L}_{KL}$, since $\mathcal{L}_{task}$ equals to $\mathcal{L}_R$. And we summarise the process of AdaHGNN in Algorithm 1 in Appendix A.7 to present the model from a general view.

4 Experiments

4.1 Experimental Setup

We evaluate our proposed model, AdaHGNN, on twelve benchmark datasets, and compare with nine state-of-the-art methods over both node and graph level tasks, i.e., link prediction, node classification, and graph classification. Code and data are available at `https://www.dropbox.com/sh/evel6yvy7n6ts75/AACTxsUaeiwdCzFb0PWyz_5oa?dl=0`.

Datasets. We use six datasets for node-wise tasks: four of them are citation networks, i.e., ACM [2], Citeseer [10], Cora [10] and DBLP [2], one is Wiki [19] webpage network, and one is Emails communication graph from SNAP [12] containing no node features. Details about the datasets are referred to Table 5 in Appendix A.2. For the graph classification task, we employ six bioinformatics datasets (NCI1, NCI109, DD, MUTAG, Mutagenicity and PROTEINS) from `https://chrsmrrs.github.io/datasets/`, and their details are summarised in Table 6 in Appendix A.2.

Competing methods. We adopt five different pooling approaches as competing methods for the graph classification task, including SORTPOOL [24], DIFFPOOL [20], TOPKPOOL [6], SAGPOOL [11] and STRUCTPOOL [22]. Meanwhile, since only one pooling method, i.e., TOPKPOOL, is practicable for node-wise tasks, hence we adopt other four GNN baselines as supplementary baselines for the node-wise tasks, including GCN [10], GraphSAGE [7], GAT [15] and GIN [18], which are typical GNN models with the message-passing mechanism. See Appendix A.3 for details of these methods.

Table 1: Results of graph classification on six different datasets, in terms of classification accuracy.

Models	NCI1	NCI109	D&D	MUTAG	Mutagenicity	PROTEINS
SORTPOOL	72.25	73.21	73.31	71.47	74.65	70.49
DIFFPOOL	76.47	76.17	76.16	73.61	76.30	71.90
TOPKPOOL	77.56	77.02	73.98	76.60	78.64	72.94
SAGPOOL	75.76	73.67	76.21	75.27	77.09	75.27
STRUCTPOOL	77.61	78.39	80.10	77.13	80.94	**78.84**
AdaHGNN	**79.77**	**79.36**	**81.51**	**80.11**	**82.04**	77.04

Table 2: Results of node-wise tasks in terms of link prediction and node classification on six different datasets. These two tasks are evaluated on classification accuracy and ROC-AUC, respectively.

	ACM		Citeseer		Cora		Emails		DBLP		Wiki	
Models	NC	LP	NC	LP	NC	LP	NC	LP	NC	LP	NC	LP
GCN	92.25	0.975	76.13	0.887	88.90	0.918	85.03	0.930	82.68	0.904	69.03	0.523
GraphSAGE	92.48	0.972	76.75	0.884	88.92	0.908	85.80	0.923	83.20	0.889	71.83	0.577
GAT	91.69	0.968	76.96	0.910	88.33	0.912	84.67	0.930	84.04	0.889	56.50	0.594
GIN	90.66	0.787	76.39	0.808	87.74	0.878	87.18	0.859	82.54	0.820	66.29	0.501
TOPKPOOL	93.42	0.890	75.59	0.918	87.68	0.932	89.16	0.936	85.27	0.934	71.33	0.734
AdaHGNN	**93.61**	**0.988**	**78.92**	**0.970**	**90.92**	**0.948**	**91.88**	**0.937**	**88.36**	**0.965**	**73.37**	**0.920**

Table 3: Comparison of AdaHGNN with different loss functions in terms of different tasks.

Models	DBLP	Wiki	ACM	Citeseer	Emails	Mutagenicity
	LP	LP	NC	NC	NC	GC
AdaHGNN+$\mathcal{L}_{task}$	0.956	0.914	91.75	76.63	87.01	79.04
AdaHGNN+$\mathcal{L}_{task}$+$\mathcal{L}_{KL}$	-	-	92.22	77.17	87.50	78.94
AdaHGNN+$\mathcal{L}_{task}$+$\mathcal{L}_{R}$	-	-	93.09	77.64	89.92	80.65
AdaHGNN (Full model)	**0.965**	**0.920**	**93.61**	**78.92**	**91.88**	**82.04**

Settings. For the graph classification task, we perform all experiments generally following the hierarchical pooling pipeline of [11] with similar parameters to ensure a fair comparison. 80% of the graphs are randomly selected as training and the rest 10% graphs are used for validation and testing, respectively. For the node-wise tasks, we follow the settings of [21] that we use two sets of 10% labelled nodes/existing links as validation and test sets, with the remaining 80% labelled nodes/existing links used as the training set. Note that, for link prediction task, an equal number of nonexistent links used as a supplementary part for every set. We present the average performance of 10 times experiments with random seeds, link prediction is evaluated by ROC-AUC and classification tasks (i.e., node and graph classification) are evaluated by classification accuracy. Detailed information about the experimental settings can be found in Appendix A.4.

4.2 Experimental Results

Performance on graph-level task. Experimental results are summarised in Table 1. Clearly, our model achieves the best performance on five of the six datasets and significantly outperforms all competing pooling techniques by up to 3.86% improvement. For the dataset PROTEINS, our result is still competitive since STRUCTPOOL only slightly outperforms our model, and our result is still much better than the other baselines. This is because our model involves adaptive pooling and unpooling operators to interactively update node and graph level information, and further allows them to enhance the representations of nodes and graphs during the training process.

Performance on node-wise task. For the node-wise tasks, we compare our model with four GNN models and one pooling based model, i.e., TOPKPOOL, since other pooling approaches do not provide an unpooling operator. Experimental results in Table 2 show that AdaHGNN can outperform the competing methods by up to 3.6% and 25.3% improvements on node classification and link prediction tasks, respectively. And our model obtains the highest average scores on both link prediction and node classification tasks. This is because these node-wise tasks depend on the quality of obtained node representations. However, existing message-passing GNN models are limited to *flat* aggregation mechanism, so it is difficult to capture the global information. The improvement of our AdaHGNN is more appealing in the link prediction task, e.g., achieving 54.88% improvement compared with the *flat* GNN models on the Wiki dataset.

Ablation study of different loss functions. The loss function of our AdaHGNN consists of three parts, i.e., $\mathcal{L}_{task}$, $\mathcal{L}_R$ and $\mathcal{L}_{KL}$. We perform an investigation on the influence of removing different

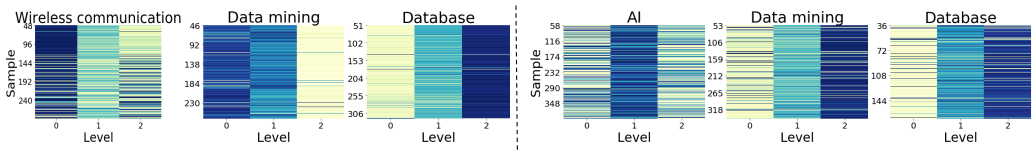

Figure 2: Visualisation of attention weight for messages from different levels. Dark colours refer to higher attention weights.

Table 4: Comparison of AdaHGNN with and without *flyback* aggregation in terms of graph classification accuracy on NCI1, NCI109 and Mutagenicity datasets.

Models	**NCI1**	**NCI109**	**Mutagenicity**
AdaHGNN without *flyback* aggregation	75.54	77.49	79.89
AdaHGNN (Full model)	**79.77**	**79.36**	**82.04**

parts of the loss function. Table 3 provides the results. For the link prediction task, we have $\mathcal{L} = \mathcal{L}_R + \gamma \mathcal{L}_{KL}$, since $\mathcal{L}_{task}$ equals to $\mathcal{L}_R$. Thus there are two comparison experiments missing in link prediction part. From the results, we can see that $\mathcal{L}_R$ can significantly improve the performance over all three tasks. This is mainly because it can eliminate the over-smoothing problem caused by the received messages from different levels. Meanwhile, $\mathcal{L}_{KL}$ can slightly improve the results as well in five of the six cases.

Understanding of messages from different levels. Compared with existing GNN models that follow a *flat* message-passing mechanism, AdaHGNN can adaptively receive messages from different levels in the learned hierarchical structure. The messages from different levels contain the meso/macro level knowledge encoded by the hyper nodes. We aim to figure out the importance of received messages from different levels. Here we consider the node classification task on the ACM and DBLP datasets as an example. ACM and DBLP are two citation datasets, and their nodes are labelled with the associated research topics. The attention scores of the received messages from different levels are plotted in Figure 2. We can find that the classifications of different paper topics come from different distributions of attention weights over various levels in the hierarchy. The relatively general topics, i.e., AI and wireless communication, receive messages from different levels with weights that are relatively indistinguishable. The topic, i.e. data mining, in these two datasets has different attention patterns. For instance, it receives messages from level-1 with the highest attention in the ACM dataset, but receives messages from level-3 with the highest attention in the DBLP dataset.

Ablation study of the *flyback* aggregator. The above analysis confirms that hierarchical semantics can improve the generated node representation in AdaHGNN. Here, we further verify whether the node representations with hierarchical semantics can improve graph representations in the next training iteration. Specifically, we aim to see how the *flyback* aggregator contributes to the performance of graph classification by removing and keeping it. The results are summarised in Table 4. It is clear that the node representations enhanced by the *flyback* aggregator can indeed improve the graph representation in the prediction task.

Effect of the number of levels in the hierarchical structure. As it has been proved that the existing GNN models will have worse performance when the network goes deeper [13], we also examine whether our model can benefit from more levels in the hierarchical structure. We performed another group of experiments to investigate the relationship between hierarchy levels and performance, the results are summarised in Table 7 in Appendix A.5. We find that deepening the hierarchy levels of AdaHGNN can improve the performance of both tasks of link prediction and node classification.

5 Conclusion

In this paper, we have proposed AdaHGNN, a method that interactively generates both node and graph level representations, and realises the collective optimisation between them. We also designed an adaptive and efficient pooling operator with a novel ego-network selection approach to enhance the hierarchical structure construction, and a training strategy to overcome the over-smoothing problem during training. One future research direction is to extend AdaHGNN for heterogeneous networks.

6 Broader Impact

As a paper focusing on deep learning model design, the broader impact of our work would be mainly reflected in future applications. GNN models have been widely applied in various domains with impressive performance, such as medical design, physics prediction, social network analysis, recommendation systems, and software engineering. Such outstanding achievements mainly depend on the powerful ability of GNNs on mining information from structural data.

References

[1] Pablo Barceló, Egor V. Kostylev, Mikaël Monet, Jorge Pérez, Juan L. Reutter, and Juan Pablo Silva. The Logical Expressiveness of Graph Neural Networks. In *Proceedings of the 2020 International Conference on Learning Representations (ICLR)*, 2020.

[2] Deyu Bo, Xiao Wang, Chuan Shi, Meiqi Zhu, Emiao Lu, and Peng Cui. Structural Deep Clustering Network. In *Proceedings of the 2020 International Conference on World Wide Web (WWW)*, pages 1400–1410. ACM, 2020.

[3] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral Networks and Locally Connected Networks on Graphs. In *Proceedings of the 2014 International Conference on Learning Representations (ICLR)*, 2014.

[4] Catalina Cangea, Petar Velickovic, Nikola Jovanovic, Thomas Kipf, and Pietro Liò. Towards Sparse Hierarchical Graph Classifiers. *CoRR abs/1811.01287*, 2018.

[5] Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, and Xu Sun. Measuring and Relieving the Over-smoothing Problem for Graph Neural Networks from the Topological View. In *Proceedings of the 2020 AAAI Conference on Artificial Intelligence (AAAI)*. AAAI, 2020.

[6] Hongyang Gao and Shuiwang Ji. Graph U-Nets. In *Proceedings of the 2019 International Conference on Machine Learning (ICML)*. JMLR, 2019.

[7] William L. Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proceedings of the 2017 Annual Conference on Neural Information Processing Systems (NeurIPS)*, pages 1025–1035. NeurIPS, 2017.

[8] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural Collaborative Filtering. In *Proceedings of the 2017 International Conference on World Wide Web (WWW)*, pages 173–182. ACM, 2017.

[9] Thomas N. Kipf and Max Welling. Variational Graph Auto-Encoders. *CoRR abs/1611.07308*, 2016.

[10] Thomas N. Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks. In *Proceedings of the 2017 International Conference on Learning Representations (ICLR)*, 2017.

[11] Junhyun Lee, Inyeop Lee, and Jaewoo Kang. Self-Attention Graph Pooling. In *Proceedings of the 2019 International Conference on Machine Learning (ICML)*. JMLR, 2019.

[12] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, June 2014.

[13] Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper Insights Into Graph Convolutional Networks for Semi-Supervised Learning. In *Proceedings of the 2018 AAAI Conference on Artificial Intelligence (AAAI)*, pages 3538–3545. AAAI, 2018.

[14] Ekagra Ranjan, Soumya Sanyal, and Partha Pratim Talukdar. ASAP: adaptive structure aware pooling for learning hierarchical graph representations. In *Proceedings of the 2020 AAAI Conference on Artificial Intelligence (AAAI)*. AAAI, 2020.

[15] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. In *Proceedings of the 2018 International Conference on Learning Representations (ICLR)*, 2018.

[16] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A Comprehensive Survey on Graph Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[17] Junyuan Xie, Ross B. Girshick, and Ali Farhadi. Unsupervised Deep Embedding for Clustering Analysis. In *Proceedings of the 2016 International Conference on Machine Learning (ICML)*. JMLR, 2016.

[18] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How Powerful are Graph Neural Networks? In *Proceedings of the 2019 International Conference on Machine Learning (ICML)*. JMLR, 2019.

[19] Cheng Yang, Zhiyuan Liu, Deli Zhao, Maosong Sun, and Edward Y. Chang. Network Representation Learning with Rich Text Information. In *Proceedings of the 2015 International Joint Conferences on Artifical Intelligence (IJCAI)*, pages 2111–2117. IJCAI, 2015.

[20] Rex Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L. Hamilton, and Jure Leskovec. Hierarchical Graph Representation Learning with Differentiable Pooling. In *Proceedings of the 2018 Annual Conference on Neural Information Processing Systems (NeurIPS)*, pages 4805–4815. NeurIPS, 2018.

[21] Jiaxuan You, Rex Ying, and Jure Leskovec. Position-aware Graph Neural Networks. In *Proceedings of the 2019 International Conference on Machine Learning (ICML)*. JMLR, 2019.

[22] Hao Yuan and Shuiwang Ji. StructPool: Structured Graph Pooling via Conditional Random Fields. In *Proceedings of the 2020 International Conference on Learning Representations (ICLR)*, 2020.

[23] Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. In *Proceedings of the 2018 Annual Conference on Neural Information Processing Systems (NeurIPS)*, pages 5171–5181. NeurIPS, 2018.

[24] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. In *Proceedings of the 2018 AAAI Conference on Artificial Intelligence (AAAI)*. AAAI, 2018.

[25] Xiaotong Zhang, Han Liu, Qimai Li, and Xiao-Ming Wu. Attributed Graph Clustering via Adaptive Graph Convolution. In *Proceedings of the 2019 International Joint Conferences on Artifical Intelligence (IJCAI)*, pages 4327–4333. IJCAI, 2019.

[26] Ziwei Zhang, Peng Cui, and Wenwu Zhu. Deep Learning on Graphs: A Survey. *IEEE Transactions on Knowledge and Data Engineering*, 2020.

A Appendix

A.1 Influence of hyper-parameter in Top-k Selection

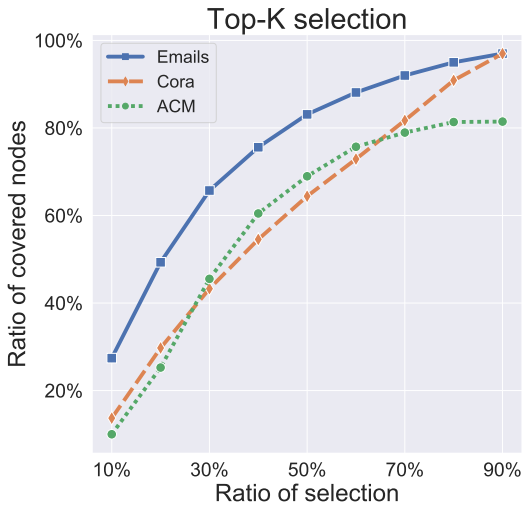

Figure 3: Ratio of covered nodes in the graph with different ratio of selection.

To address the efficiency limitation of pooling approaches, recently proposed methods follow a Top-k selection strategy, which uses a pre-defined pooling ratio and each level only selects clusters with the top k fitness scores. However, we argue that this selection strategy introduced one additional hyper-parameter which is crucial for the final performance [6], thus reduces their convenience in applications. In addition, as shown in Figure 3, different k will significantly affect the number of covered nodes in the graph, which means the important node features could get lost during the trivial pooling operation. Therefore, in this paper, we proposed a novel selection strategy, which does not introduce any hyper-parameters, realises a true adaptive graph pooling method.

A.2 Dataset Description

Table 5: Statistics of the datasets for node-wise tasks. N.A. means a dataset does not contain node features.

Dataset	#Nodes	#Edges	#Features	#Classes
ACM	3,025	13,128	1,870	3
Citeseer	3,327	4,552	3,703	6
Cora	2,708	5,278	1,433	7
Emails	799	10,182	N.A.	18
DBLP	4,057	3,528	334	4
Wiki	2,405	1,2178.5	4973	17

Table 6: Statistics summary of datasets for graph classification.

Dataset	#Graphs	#Nodes (avg)	#Edges (avg)	#Features	#Classes
NCI1	4,110	29.87	32.3	37	2
NCI109	4,127	29.68	32.13	38	2
D&D	1,178	284.32	715.66	89	2
MUTAG	188	17.93	19.79	7	2
Mutagenicity	4,337	30.32	30.77	14	2
PROTEINS	1,113	39.06	72.82	32	2

For the node-wise tasks, we use six datasets to perform experiments on link prediction and node classification tasks:

ACM. This is a paper network from ACM dataset. An edge exists between two papers if they are written by common authors, and paper features are the bag-of-words of the paper's keywords. Here,

we select papers published in KDD, SIGMODm SIGCOMM and MobiCOMM, and divide the papers into three classes (database, wireless communication, and data mining) by their research areas.

Citeseer. This is a citation network of $3,327$ papers, which contains sparse bag-of-words feature vectors for each document and a list of citation links between these documents. These paper belong to six research areas: agents, artificial intelligence, database, information retrieval, machine language and human-computer interaction.

Cora. This is a citation network consists of $2,708$ scientific publications and $5,429$ links. Each publication is described by a $1,433$ dimension word vector as a node feature.

DBLP. This is an author network from the DBLP dataset with $4,057$ authors. There is an edge between two authors if they have a co-author relationship in the dataset. And the authors are divided into four research areas: database, data mining, artificial intelligence, and computer vision. We label each author's research area depending on the conferences they submitted. Author features are the elements of a bag-of-words represented by keywords.

Emails. 7 real-world email communication graphs from SNAP without node features. Each graph has 6 communities and each node is labelled with the community it belongs to.

Wiki. This is a webpage network with $2,405$ pages, where nodes are webpages and are connected if one links the other. It is associated with TF-IDF weighted word vector.

For the graph level task, we evaluate our methods on six large graph datasets selected from common benchmarks used in graph classification tasks, they can be found at `https://chrsmrrs.github.io/datasets/`. **D&D** and **PROTEINS** are dataset containing proteins as graphs. **NCI1** and **NCI109** are datasets involves anticancer activity graphs. The **MUTAG** and **Mutagenicity** datasets consist of chemical compounds divided into two classes according to their mutagenic effect on a bacterium.

A.3 Competing Methods Description

For node-wise tasks, we adopt 6 competing methods which include five GNN models with flat message-passing mechanism, and one state-of-the-art method that contains a hierarchical structure:

GCN [10]: It is the first deep learning model which generalises the convolutional operation on graph data and it introduces the semi-supervised paradigm for train GNN models.

GraphSAGE [7]: It extends the pooling operation to mean/ max/ LSTM poolings and introduces an unsupervised way to train the GNN model. Besides, it discusses the possibility of applying GNN on large-scale graphs and inductive learning settings. We adopt an implementation with mean pooling.

GAT [15]: It employs trainable attention weight during message aggregation from neighbours. We set the number of multi-head-attentions as 1.

GIN [18]: It summarises previous existing GNN layers as two components, AGGREGATE and COMBINE, and models injective multiset functions for the neighbour aggregation.

TOPKPOOL [6]: It generalises the U-nets architecture of convolutional neural networks for graph data to get better node embedding. We sample 2000 nodes in the gPool layers if there are enough nodes, otherwise, we sample 200 nodes.

For the graph classification task, we adopt five competing methods that involve the state-of-the-art models:

SORTPOOL: It applies a GNN architecture and then performs a single layer of soft pooling followed by 1D convolution on sorted node representations.

DIFFPOOL: It proposes a differentiable pooling operator that learns a soft assignment matrix mapping each node to a set of clusters in the hyper-graph.

TOPKPOOL: To address the efficiency limitation of DIFFPOOL, they propose a scalar projection score for each node and selects the top k nodes to form the hyper-graph.

SAGPOOL: It is a Top-k selection based architecture that further propose leverage self-attention network to learn the node scores.

STRUCTPOOL: As a recently proposed method, it is a Top-k selection based architecture that consider graph pooling as a node clustering problem and employ conditional random fields to build relationships between the assignments of different nodes.

A.4 Experiment Settings

The detailed settings for the experiments in this work include:

- **Hyper-parameters**. For the competing methods with a hierarchical structure, we set up experiments with the same hyper-parameters as they described in the original paper if it is available. Otherwise, we let them have the same hyper-parameters as AdaHGNN. For a fair comparison, the default embedding dimension d of all models is set to 64, and all methods adopt the same input node features, learning rate, number of iterations as AdaHGNN. In terms of the number of levels of AdaHGNN, we select the one with better performance, between $2-5$. Specifically, for link prediction: Emails (4 levels), Wiki (5 levels), ACM (5 levels), DBLP (5 levels), Cora (4 levels) Citeseer (4 levels); for node classification: Emails (3 levels), Wiki (4 levels), ACM (4 levels), DBLP (4 levels), Cora (3 levels) Citeseer (5 levels); and for graph classification: D&D (), PROTEINS (4 levels), NCI1 (4 levels), NCI109 (4 levels), MUTAG (4 levels), Mutagenicity (3 levels). For the hyper-parameters of loss function $\mathcal{L} = \mathcal{L}_{task} + \gamma\mathcal{L}_{KL} + \delta\mathcal{L}_R$, we set $\gamma = 0.1$ and $\delta = 0.01$ for all the experiments to let all loss values lie in a reasonable range, i.e., $(0, 10)$.
- **Software & Hardware**. We employ Pytorch[1] and PyTorch Geometric[2] to implement all models that mentioned in this paper, and further conduct it on a server with GPU (NVIDIA Tesla V100) machines. Code and data are available at: `https://www.dropbox.com/sh/evel6yvy7n6ts75/AACTxsUaeiwdCzFb0PWyz_5oa?dl=0`.

A.5 Hierarchy Level Study of AdaHGNN

Table 7: Comparison of AdaHGNN with different number of hierarchy levels in terms of different tasks on multiple datasets.

# Levels	DBLP	Wiki	ACM	Citeseer	Emails	Mutagenicity
	LP	LP	NC	NC	NC	GC
2	0.951	0.912	92.60	77.68	86.83	78.16
3	0.958	0.913	93.38	74.67	**91.88**	**82.04**
4	0.959	0.917	**93.61**	76.15	90.61	81.58
5	**0.965**	**0.920**	90.84	**78.92**	-	81.01

Since the number of hierarchy levels of AdaHGNN is an important hyper-parameter, we conduct experiments to investigate the relationship between hierarchy levels and AdaHGNN's performance in terms of multiple tasks, e.g., link prediction, node classification, and graph classification. The results are summarised in Table 7. We can observe from the results that the best performance of different tasks comes with a different number of levels. For the link prediction, more levels will lead to better performance. Then, for other tasks, the best number of levels depends on the size of the dataset.

A.6 Proof of Proposition 1

Proof. For $G = (V, E, X)$ with n nodes. n ego-networks can be generated by following the procedures, $c_\lambda(v_i) = \{j \mid \forall j \in \mathcal{N}_i^\lambda\}$, and each ego-network will be given a fitness score ϕ_i as follows:

$$\phi_i = \frac{1}{|\mathcal{N}_i^\lambda|} \sum_{v_j \in \mathcal{N}_i^\lambda} f_\phi(v_i, v_j), \tag{4}$$

where $f_\phi(v_i, v_j) = \mathrm{Softmax}(\overrightarrow{a}^{\mathrm{T}}\, \sigma(Wh_j||Wh_i)) * \mathrm{Sigmoid}(h_j^{\mathrm{T}} \cdot h_i)$. We assume that these cluster fitness scores are not all the same, thus there exists at least one maximum ϕ_{max}. Hence, the clusters with fitness score ϕ_{max} satisfy the requirements of ego-network selection requirement that:

$$\phi_{max} > \phi_j, \ \ \forall v_j \in \mathcal{N}_{max}^1, \tag{5}$$

[1] https://pytorch.org/

[2] https://pytorch-geometric.readthedocs.io/en/latest/

where $\mathcal{N}^1_{max} = \{v_j \mid if\ d(v_i, v_j) = 1\}$. Therefore, for any connected G with n nodes, there exists at least one cluster satisfies the requirements of our ego-network selection approach. □

A.7 Algorithm

We have presented the idea of AdaHGNN and the design details of each component in Section 3. Here, we further generally summarise the entire model as Algorithm 1 to provide a general view of our model. Specifically, given a graph G, we firstly apply a GNN layer to generate the primary node embedding (line 3) as described in Section 3.1. After, we construct a hierarchical structure with k levels (line 5-15) with adaptive pooling operator as described in Section 3.2. Meanwhile, we also propose a method to define the initial features of pooled hyper-nodes (line 16-21). The graph connectivity of the pooled graph is maintained as (line 22). Next, we perform the graph convolution operation on the pooled graph to summarise the relationship between different hyper-nodes (line 23). The explored hierarchical semantics will be further distributed to the original graph follows an unpooling operator (line 24) as described in Section 3.3. At last, the *flyback* aggregator could summarise the meso/macro level knowledge from different levels as the node representation of G (line 26) as described in Section 3.4 and additional READOUT operators could summarise the node representations as to the graph representation (line 27).

Algorithm 1: *Adaptive Hierarchical Graph Neural Network*

Input: graph $G = (V, E, X)$.
Output: node representations $\mathbf{h}_v$, graph representations $\mathbf{h}_g$.

1 $H_0 = \mathrm{ReLU}(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{\frac{1}{2}}XW^0)$, where $\hat{A} = A + I$, $\hat{D} = \sum_j \hat{A}_{ij}$ and $W^0 \in \mathbb{R}^d$;
2 **for** $k \leftarrow \{1, 2, \ldots, K\}$ **do**
3 **for** $v_i \leftarrow \{v_1, v_2, \ldots, v_n\}$ **do**
4 **for** $v_j \in \mathcal{N}_i^{\lambda}$ **do**
5 $\phi_{ij} = f_\phi(v_i, v_j) = \mathrm{Softmax}(\overrightarrow{a}^{\mathrm{T}}\sigma(Wh_j||Wh_i)) \times \mathrm{Sigmoid}(h_j^{\mathrm{T}} \cdot h_i)$;
6 **end**
7 $\phi_i = \frac{1}{|\mathcal{N}_i^{\lambda}|}\sum_{v_j \in \mathcal{N}_i^{\lambda}} \phi_{ij}$;
8 **end**
9 **for** $v_i \leftarrow \{v_1, v_1, \ldots, v_n\}$ **do**
10 $\hat{N}_p = \{v_i \mid \phi_i > \phi_j, \forall v_j \in \mathcal{N}_i^1\}$;
11 **end**
12 $\hat{N}_r = \{v_j \mid v_j \notin c_\lambda(v_i), \forall v_i \in \hat{N}_c\}$;
13 Generate the hyper-node formation matrix: $S_k \in \mathbb{R}^{n \times (|\hat{N}_p| + |\hat{N}_r|)}$;
14 **for** $v_i \in \hat{N}_r$ **do**
15 $X_k(i,:) = H_{k-1}(i,:)$;
16 **end**
17 **for** $v_i \in \hat{N}_p$ **do**
18 $X_k(i,:) = H_{k-1}(i,:) + \sum_{v_j}^{c_\lambda(v_i)\setminus v_i} \alpha_{ij} H_{k-1}(j,:)$;
19 **end**
20 $A_k = S_k^{\mathrm{T}}\hat{A}^{k-1}S_k$;
21 $H_k = \mathrm{ReLU}(\hat{D}^{-\frac{1}{2}}\hat{A}_k\hat{D}^{\frac{1}{2}}X_kW^k)$;
22 $\hat{H}_k = (S_1 \ldots (S_{k-1}(S_kH_k)))$;
23 **end**
24 $H = H_0 + \sum_i^k \beta_i \hat{H}_i$;
25 $h_g = READOUT(\{H, \hat{H}_1, \ldots, \hat{H}_k\})$;
26 $\mathbf{h}_i \in \mathbb{R}^d, \forall v_i \leftarrow \{v_1, v_2, \ldots, v_n\}$;
27 $\mathbf{h}_g \in \mathbb{R}^d$;