Information Sciences 626 (2023) 94–113

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

DGSLN: Differentiable graph structure learning neural network for robust graph representations

Xiaofeng Zou [a], Kenli Li [a,*], Cen Chen [a,b], Xulei Yang [b], Wei Wei [c], Keqin Li [a,d]

[a] College of Information Science and Engineering, Hunan University, Changsha, China
[b] Institute for Infocomm Research, Singapore, Singapore
[c] School of Computer Science and Engineering, Xi'an University of Technology, Xi'an, China
[d] Department of Computer Science, State University of New York, New Paltz, NY, USA

ARTICLE INFO

Article history:
Received 30 January 2022
Received in revised form 4 January 2023
Accepted 6 January 2023
Available online 11 January 2023

Keywords:
Adaptive structure learning
Graph neural networks
Graph regularization
Graph sparsification

ABSTRACT

Recently, graph neural networks (GNNs) have been widely used for graph representation learning, where the central idea is to recursively aggregate neighborhood information to update the node feature based on the graph topology. Therefore, an appropriate graph topology is crucial for effective graph representation learning in GNNs. However, most existing GNNs assume that the initial graph is complete and accurate, and utilize the fixed initial graph structure in the entire network, which may limit the learning representation capability of the model. In this work, we propose a novel differentiable graph structure learning neural network (DGSLN), which learns suitable graph structures for GNNs. Specifically, our DGSLN presents a general graph generation scheme that integrates various useful graph prior messages to generate normal structures. We describe the generation process with homophily, node degree, and sparsity as examples. Moreover, we develop a hybrid loss function to ensure the quality of learned graphs, which combines task-specific loss and graph regularization loss to optimize graph structures from both structural adaptive and task-driven aspects. Extensive experiments on graph classification and node classification have shown that our approach significantly improves performance on different benchmark datasets compared to state-of-the-art GNNs methods.

1. Introduction

Convolutional Neural Networks (CNNs) have achieved great strides in many artificial intelligence tasks, such as computer vision [1] and machine translation [2]. There is a common property behind these tasks: the underlying data can be represented as a grid-like structure. However, many real-world data are in an irregular domain and can be represented naturally as graphs, including social networks, citation networks, etc [3]. Owing to the great success of CNNs, it is quite appealing to extend the classical convolution to graph-structured data.

Recently, many studies have generalized neural networks to process graphs of arbitrary structures, called graph neural networks (GNNs) [4]. GNNs usually follow the neighborhood aggregation scheme, which recursively aggregates neighborhood information based on the given graph structure to update node features. Therefore, GNNs are strongly sensitive to

* Corresponding author.
E-mail addresses: zouxiaofeng@hnu.edu.cn (X. Zou), lkl@hnu.edu.cn (K. Li), chenc@i2r.a-star.edu.sg (C. Chen), yang_xulei@i2r.a-star.edu.sg (X. Yang), weiwei@xaut.edu.cn (W. Wei), lik@newpaltz.edu (K. Li).

https://doi.org/10.1016/j.ins.2023.01.059
0020-0255/

the quality of the given graph structure [5]. Nevertheless, most existing GNNs [6–8] assume that the initial graph structure is ground-truth information, and apply the fixed initial graph to the entire network for recursive message passing. Such an assumption may suffer from the following limitations. (i) Since the process of collecting graph data usually contains uncertainties or errors, real-world graphs commonly contain noisy edges or missing edges [3,41]. As illustrated in Fig. 1(a), some nodes in the initial graph structure on the Cora dataset do not have edge connections and suffer from missing edges. (ii) The initial graph structure reflects the topological relationship among the original node features. After multi-layer feature aggregation and update, the initial graph structure may no longer accurately represent the true interaction relationship. Recent studies [5,9] have discovered that unreliable graph structures can greatly limit the representation capability of GNNs, and thus exploring appropriate graph structures for GNNs is essential to learn robust graph representations.

To tackle these issues, some studies have been presented to conduct graph structure learning (GSL) in GNNs to improve performance of GNNs. These methods [10–12] utilize metric functions to compute node pairwise interactions to adjust the graph structure, and then use the revised graph to guide the message-passing process. Although these methods have achieved positive results, there are some limitations: (i) These methods greatly rely on the quality of the metric function design. Most existing methods [10,11] directly construct the graph structure using a simple distance metric, which would probably render the learned graph topology not well suited to the node features. In addition, some methods [8,13] employ the learnable attention mechanism to re-weight the existing edges of the given graph. However, these methods do not modify the graph structure and the model is still prone to interference from noisy data. (ii) The above metric-based methods consider only feature similarity in the process of graph structure learning, and graph structures learned from a single information source inevitably appear to lead to bias and uncertainty. In real-world applications, the graph structure may be constrained by many underlying principles, such as homophily [14], sparsity [15], degree distribution [14]. Some studies [16,17] have indicated that combining various graph properties to optimize the initial graph structure can help improve GNN performance. Nevertheless, these approaches ignore exploring the underlying graph topology in node features.

In this work, we aim to design a proper graph structure learning approach to adaptively learn better graph structures. In the meanwhile, it can be combined with GNNs to improve the message-passing process in GNNs by the learned graph structures. To achieve these objectives, we hope to design a differentiable graph structure neural network in the stack of GNNs. It is extremely challenging in technical terms, and three obstacles need to be addressed: (i) How to consider both the given graph structure and node features to learn a better graph structure in the differentiable neural network? (ii) Real-world graph structures follow many basic principles, such as homophily [14], sparsity [15]. How to take these graph properties into consideration in a differentiable neural network? (iii) A proper loss function needs to be designed to optimize differentiable neural networks.

To this end, we propose a differentiable graph structure learning neural network (DGSLN), which learns appropriate graph structures for GNNs to achieve robust graph representation learning. Specifically, we first employ the attention mechanism [18] to explore the homophily of graphs for learning the basic graph structure. Different from previous work, we directly learn the new graph topology by adaptively capturing interactions among node features through the attention mechanism. To satisfy graph sparsity, we propose a differentiable graph sparsity operation, which efficiently converts the learned dense connected graphs into sparse graphs through an explicit masking function. Besides, we develop a gated graph integration mechanism that combines the initial graph structure with the learned sparse graph. Finally, we design a hybrid loss function to co-optimize the graph representation and learned graph structure. It consists of a task loss function and a graph regularization loss, and the graph regularization loss can force the learned graph to satisfy graph properties. In this manner, the network model can be optimized from both data-driven and task-driven aspects.

More importantly, our proposed DGSLN is a general building block that can be easily integrated into each layer of various GNNs. To validate the generality and effectiveness, we plug DGSLN into the most commonly used graph pooling and convolution operations. The redesigned graph convolution layer learns the new graph topology before neighborhood aggregation, and utilizes the generated topology to guide subsequent message passing. The redesigned graph pooling layer learns a new topology for the reduced graph after pooling to ensure the integrity of the pooled graphs. Extensive experiments on graph classification and node classification have demonstrated the superiority and robustness of our approach compared to state-of-the-art GNNs methods.

Overall, our contributions are as follows:

- We propose a novel differentiable structural learning neural network (DGSLN), which utilizes the attention mechanism to dynamically learn an adaptive graph topology from node features in each layer for robust graph representation learning.
- DGSLN simultaneously considers both node features, initial graph structures, and graph properties in the graph structure learning process.
- DGSLN is a general building block that can be easily integrated into various GNNs. We also design a hybrid loss function, so that graph representation and graph structure can be learned simultaneously.
- Extensive experiments on graph classification and node classification have demonstrated the superiority and robustness of our approach compared to state-of-the-art GNNs methods and graph structure learning (GSL) methods.

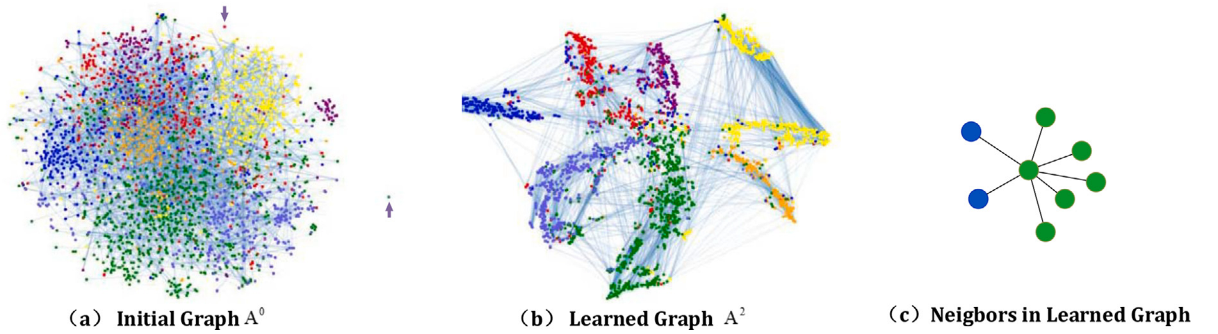


Fig. 1. Visualization of (a) the initial graph structure and (b) the learned graph structure on Cora dataset, where different colors denote different node categories. The nodes pointed by arrows in (a) reveal that some nodes in the initial graph structure are not connected by edges and there are missing edges. While our DGSLN can effectively capture more meaningful topology information, which complements the initial graph structure. (c) Neighbor change of the rightmost green node in (a).

2. Related work

In this section, we review the relevant literature in two main domains: i) graph neural networks and ii) graph structure learning.

2.1. Graph neural networks

Recently, various GNNs have been proposed for the complexity of graph data [4]. GNNs are generally viewed as a neighbor aggregation scheme, which iteratively updates node representations by gathering neighboring node features. Kipf et al. [7] explored GCN, which utilizes mean aggregation to generate new feature representations. [6] proposed GraphSAGE that uses multiple aggregation schemes to gather feature messages from neighbors, including max/mean/lstm. Veličković et al. [8] developed GAT, which employs self-attention mechanism to determine the weights of neighbors during the information aggregation process. Nevertheless, all the above methods only apply the fixed static graph to update the node representation, and cannot dynamically capture the graph topology in each layer of the network, which may result in local optimal solutions.

The pooling operations in GNNs can gradually capture hierarchical high-level features and expand the receptive field, thus achieving better generalization effects. Some studies explored advanced clustering techniques to group node features to achieve graph pooling, including spectral clustering [19], compressed Haar transform [20], etc. However, its potential drawback lies in the high computational cost of the feature decomposition of the clustering algorithm, which leads to significant scalability problems. Recently, some learnable graph pooling methods [21–23] have attracted attention because they can adaptively coarse the graphs based on the graph contents. A pioneer work [21] proposed DiffPool that learns a dense soft assignment matrix to map each node to a set of clusters, resulting in expensive calculations and poor scalability. Following the line, Gao et al. [22] devised gPool, which learns the scores of nodes by a learnable projection vector and then selects the subset of nodes with the highest scores to form a reduced graph. It effectively alleviates the issue of high complexity, but does not consider the graph topology during pooling. SAGPool [23] improved upon gPool by using GCN to aggregate neighborhood features while scoring nodes. However, SAGPool only takes the local structure into account, while ignoring the global connectivity of the graph, which may lose the integrity of the graph topology and hinder the downstream message-passing process.

2.2. Graph structure learning

Graph structure learning (GSL) is not a brand new topic, and there have been numerous works exploring various methods to learn graph structures from data. Some approaches [24,25] learned graph structure by learning probability distributions that fit graph relationships. Other methods [26,27] adopted auto-regressive models to generate graphs graph structures. However, these works deviate from graph representation learning.

Recent studies [10,11,16,17] have attempted to explore graph structure learning in GNNs for learning robust graph representations. These methods can be categorized into two categories: metric learning methods [10,11] and graph optimization methods [16,17]. Li et al. [10] designed AGCN, which constructs a graph by computing the Mahalanobis distance for each node pair, and utilizes the generated graph to guide the message-passing process. Jiang et al. [11] proposed GLCN utilizing a single-layer neural network to learn the pairwise relationship among two nodes. However, these methods only exploit feature similarity to construct graph structures, ignoring the diversity of graph principles. Besides, graph optimization methods [16,17] combined various graph properties to directly optimize the initial graph structure. For instance, Pro-GNN [17] optimized the given graph by employing sparsity, low rank, and feature smoothness regularizers. However, these methods are quite dependent on the initial graph structure and ignore the exploration of structural information implicit in the node features.

3. Methodology

3.1. Problem formulation

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E}, X)$ represents an undirected graph, where $\mathcal{V}$ denotes the node set with n nodes, and $\mathcal{E}$ denotes the set of edges between the nodes in $\mathcal{V}, X = \{x_1, x_2, \ldots x_n\} \in \mathbb{R}^{n\times d}$ is the node feature, each row denotes a node and each node contains d-dimensional features. The graph structure of $\mathcal{G}$ can be represented by an adjacent matrix for binary graphs $A \in \{0,1\}^{n\times n}$ or weighted graphs $A \in \mathbb{R}^{n\times n}$. Here, we focus on two common graph learning tasks: graph classification and node classification, which predict the class label of a graph and the class label of a node respectively. For the node classification, given a partially labeled graph $\mathcal{G}$, the goal of GNN is to learn a predictive function f_θ that maps the nodes to their true class label:

$$f_\theta(X, A)_i \rightarrow \mathcal{Y}_i, \tag{1}$$

where $\mathcal{Y}_i$ is the ground truth of node v_i, θ is the training parameters of f_θ, and $f_\theta(X, A)_i$ is the prediction of v_i, which is fully determined by the node feature X and graph topology A. Therefore, an appropriate graph topology is essential for effective graph representation learning. Nevertheless, most existing GNNs apply the initial graph topology to the entire network, which may suffer from many limitations. Because real-world graphs are often incomplete or noisy. Furthermore, the given graph topology may hinder the downstream message-passing process since the initial graph structure may not reflect the real topological relationship after multi-level feature transformations.

With the above analysis, the graph structure learning (GSL) problem can be formally defined as: Given a graph $\mathcal{G} = (A, X)$ with original graph structure A and feature matrix X, the task of graph structure learning is to simultaneously learn an optimized graph structure and the GNNs parameters to improve the representational power of GNNs.

3.2. Differentiable structural learning

In real-world applications, graph generation obeys some underlying principles, such as homophily, sparsity, and degree. Combining these prior knowledge can drive a better generative process and learn a more reasonable graph structure. However, it is challenging to consider different graph properties in the same generative mechanism. In this section, we propose a differentiable graph structure learning approach, as depicted in Fig. 2. It first introduces the basic graph generation scheme, and then extends it according to specific graph properties to integrate useful prior knowledge and learn the best graph structure. There are three main processes: (1) Adaptive structural learning: dynamically capturing pairwise node relationships via self-attention mechanism. (2) Differentiable graph sparsification: efficiently converting the dense connected graph into a more reasonable sparse graph with an explicit masking function. (3) Gated graph integration: adaptively integrating the learned graph with the original graph by a gating function. The specific process is as follows.

3.2.1. Adaptive structural learning

Homophily is a very crucial principle in graphs. Specifically, if two nodes are similar to each other, they are likely to be connected. Based on this principle, we can transform the graph structure learning problem into similarity metric learning, which constructs a basic graph structure by measuring the similarity among nodes. A good similarity metric should be learnable and expressively powerful. The attention mechanism [18] has been proven to be an efficient learnable similarity metric, which can capture the relational importance of feature space. However, the majority of GNNs [8,13] only utilize the attention mechanism to adjust the weights of edges that already exist in the initial graph for better representation. Instead, we employ the attention mechanism to explore the underlying graph structure. Given the node features $X^l \in \mathbb{R}^{n\times d}$ at the l-th layer as input, we apply the attention mechanism to learn the pairwise dependency relationship of graph nodes.

$$\alpha_{ij}^l = \mathrm{LeakyReLU}\left(\Theta^{\mathrm{T}}\mathrm{Concat}\left[Wx_i^l, Wx_j^l\right]\right), \tag{2}$$

where x_i^l and x_j^l are the features of the node v_i and v_j, respectively, α_{ij}^l means the dependency among node v_i and v_j, $\mathrm{Concat}[\cdot]$ is the concatenation operator, $\Theta^{\mathrm{T}} \in \mathbb{R}^{1\times 2d}$ is the learnable weight vector. $\mathrm{LeakyReLU}(\cdot)$ is the activation function, that can guarantee the non-negativity of the learned graph edges.

The above method is a basic graph generation scheme with a large enhancement scope. Since the attention mechanism (Eq. (2)) needs to compute the pairwise interactions for all graph nodes, which requires $O(n^2)$ complexity in time and space. Limited by high memory and computational complexity, it may lead to significant scalability issues. Moreover, computing the pairwise interactions for all nodes will introduce a lot of redundant edges. For some nodes, redundant edge connections may introduce noise, which runs counter to the original purpose of graph structure learning. Therefore, some useful prior knowledge can be integrated to improve the generation process and learn more reasonable graph structures. We take the degree as an example to propose a scalable graph generation scheme.

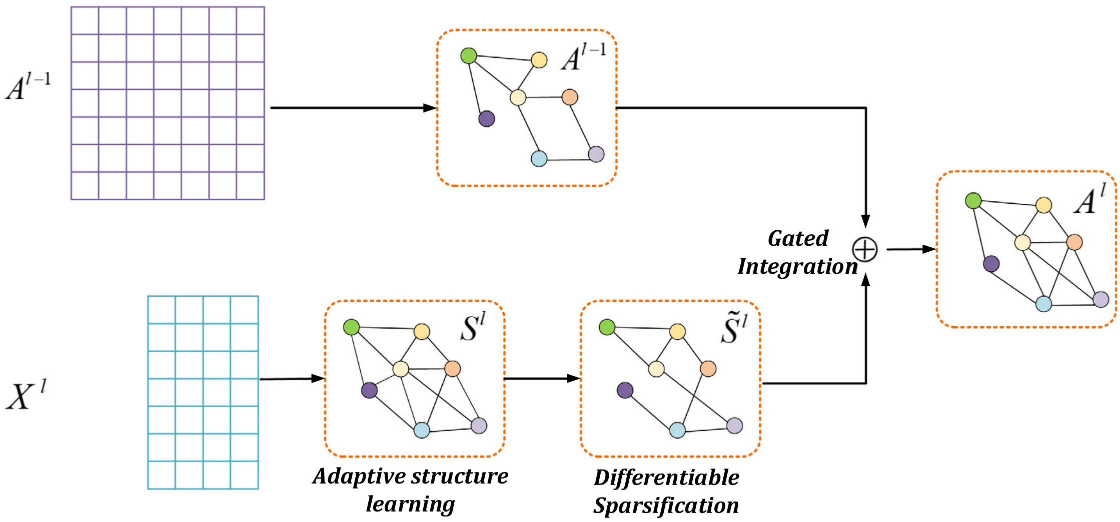


Fig. 2. Overall framework of the proposed DGSLN. It contains three main sub-procedures: adaptive structure learning, differentiable graph sparsification, and gated graph integration. Given the node features X^l in the l-th layer, we employ adaptive structure learning to dynamically learn pairwise node relationships S^l. Then, the learned dense connected graph S^l is converted into a more reasonable sparse graph $\tilde{S}^l$ by differentiable graph sparsity. Finally, we adopt gated graph integration to adaptively integrate the learned graphs $\tilde{S}^l$ with the graphs A^{l-1} of the previous layer to form the new graph structure A^l.

Scalable Graph Generation. Node degree is a common property of graphs. Theoretical analysis according to [14] revealed that the relative degree of nodes is one of the important factors affecting the quality of node feature aggregation. The low-degree nodes will become less linearly separable in message passing and are prone to be misclassified. Therefore, we pay more attention to low-degree nodes in the graph structure learning process. Based on this insight, we further propose a scalable relationship learning technique based on node sampling operation [28]. Specifically, we sort the nodes according to their in-degree, and then perform the top-k operation to sample a set of low-degree nodes $\mathcal{V}s \in \mathbb{R}^{s\times d}$ from the node set $\mathcal{V} \in \mathbb{R}^{n\times d}$. Notice that s is a fixed hyper-parameter that determines the sampling numbers. It is normally much smaller than n in large graphs. Second, we generate binary masks *mask* for each nodes $v_i \in \mathcal{V}s$, and acquire the new feature matrix $\bar{X}^l$ by multiplying the raw feature X^l with the generated masks, i.e., $\bar{X}^l = mask \cdot X^l$. Thus, Eq. (2) can be rewritten as:

$$\bar{\alpha}^l_{ij} = \mathrm{LeakyReLU}\left(\Theta^{\mathrm{T}}\mathrm{Concat}\left[W\bar{x}^l_i, Wx^l_j\right]\right), \tag{3}$$

where $\bar{x}^l_i \in \bar{X}^l$ and $x^l_j \in X^l$. In this way, we only compute pairwise interactions for low-degree nodes $\mathcal{V}s$, which require only $O(ns)$ time and space complexity. While $s \ll n$, we can significantly reduce the memory and space consumption.

Then, we utilize the softmax function to normalize $\bar{\alpha}^l_{ij}$:

$$S^l_{ij} = \mathrm{softmax}\left(\bar{\alpha}^l_{ij}\right) = \frac{\exp\left(\bar{\alpha}^l_{ij}\right)}{\sum_{j=1}^{N}\exp\left(\bar{\alpha}^l_{ij}\right)}. \tag{4}$$

3.2.2. Differentiable graph sparsification

Since the softmax function always generates non-zero values, the learned graph becomes a dense connected graph, i.e., (1) $\forall i,j \in n, S_{ij} \neq 0$, (2) $\sum_{j=1}^{n} S_{ij} = 1$. The computational cost of dense connected graphs is expensive, and it runs counter to the sparsity of real-world graph data, which may introduce lots of noise to subsequent graph representation learning. A naive idea is to directly construct the learned graph S^l as k-nearest neighbors (kNN) graph. However, this approach may cause a problem: it can distract the model from the important edge connections, especially when the attention probability distribution is flat.

To this end, we consider performing a sparse attention masking operation $\mathrm{M}(\cdot)$ on α^l_{ij} to extract a sparse adjacency matrix. Specifically, we select the k-th largest element of each row in α^l_{ij} as the threshold, and concatenate them to form a vector $c = [c_1, c_2, \cdots, c_n]$. Then the elements in α^l_{ij} that are lower than the threshold are filtered to form a sparse graph structure. The masking function $\mathrm{M}(\cdot)$ can be formulated as:

$$\tilde{\alpha}_{ij}^{l} = \mathrm{M}\left(\alpha_{ij}^{l}, k\right) = \begin{cases} \alpha_{ij}^{l} & \alpha_{ij}^{l} \geqslant c_i \\ -\infty & \text{otherwise}, \end{cases} \tag{5}$$

where k is a hyper-parameter, c_i is the k-th largest value of row i. As α_{ij}^{l} is smaller than c_i, the masking function assigns α_{ij}^{l} to negative infinity. In the subsequent normalization, these values below the threshold are normalized to 0, while the values above the threshold are retained. In this way, we prompt the model to focus on the edges with important contributions, and retain them to generate a sparse graph structure. The output graph structure $\tilde{S}_{ij}^{l}$ can be computed as below:

$$\tilde{S}_{ij}^{l} = \mathrm{softmax}\left(\mathrm{M}\left(\alpha_{ij}^{l}, k\right)\right). \tag{6}$$

Compared with sparsemax [29], our proposed method not only has the ability to generate sparse distribution, but also has higher efficiency. This is due to our method does not introduce too much extra computational or memory cost, detailed analysis can be found in Section 5.5.4. Below, we show the back-propagation process of graph sparse operation.

Given the masking function $\tilde{\alpha} = \mathrm{M}(\alpha, k)$ in Eq. (5). When calculating the gradient in back-propagation, we regard c_i as constants:

$$\begin{aligned} &\frac{\partial \tilde{\alpha}_{ij}}{\partial \alpha_{mn}} = 0 \quad (i \neq m, j \neq n) \\ &\frac{\partial \tilde{\alpha}_{ij}}{\partial \alpha_{ij}} = \begin{cases} 1 & \alpha_{ij}^{l} \geqslant c_i; \\ 0 & \text{otherwise}. \end{cases} \end{aligned} \tag{7}$$

The next step after masking process is normalization:

$$\begin{aligned} \frac{\partial \tilde{S}_{ij}^{l}}{\partial \alpha_{ij}} &= \sum_{p=1}^{N}\sum_{q=1}^{N} \frac{\partial \tilde{S}_{ij}^{l}}{\partial \tilde{\alpha}_{pq}} \frac{\partial \tilde{\alpha}_{pq}}{\partial \alpha_{ij}} = \frac{\partial \tilde{S}_{ij}^{l}}{\partial \tilde{\alpha}_{ij}} \frac{\partial \tilde{\alpha}_{ij}}{\partial \alpha_{ij}} \\ &= \begin{cases} \frac{\partial \tilde{S}_{ij}^{l}}{\partial \tilde{\alpha}_{ij}} & \alpha_{ij}^{l} \geqslant c_i; \\ 0 & \text{otherwise}. \end{cases} \end{aligned} \tag{8}$$

The softmax function is evidently differentiable, therefore, we can prove that the proposed graph sparse operation is differentiable.

3.2.3. Gated graph integration

The graph integration process aims to incorporate the learned graph structure into the raw graph structure. This is because the initial graph structure is constructed from specialty domain knowledge, which reveals much useful information about the graph topology. Moreover, we may be limited by the fact that the given node features may not contain adequate information to learn a good graph structure. Therefore, we need a fusion function, which can effectively fuse the given graph and the learned graph to form an optimal graph that adapts to the current layer.

Previous work [10,40] adopted the residual learning to directly fuse the graphs. However, this method does not consider that the influence degree of different graphs may be varied. We introduce the gated fusion mechanism that computes the weight factors for each graph and fuses the graphs using a weighted sum. Formally, the gated fusion is given as:

$$A^{l} = W_s \otimes \tilde{S}^{l} + W_a \otimes A^{l-1}, \tag{9}$$

where $\otimes$ denotes element-wise product. $W_s, W_a \in \mathbb{R}^n$ are the learnable parameters that determine the influence degree of $\tilde{S}^{l}$ and A^{l-1} on the new graph topology. $\tilde{S}^{l}$ means the learned graph structure in the l layer, while A^{l-1} means the graph structure of the previous layer (as the initial graph of the l layer), and A^{l} is the fused optimal graph of the current layer. For instance, when $l = 1, A^{0}$ is the initial graph (ground truth), $\tilde{S}^{1}$ is the graph learned from the initial node features X^{1}, and A^{1} is the optimal graph of the first layer formed by the fusion of A^{0} and $\tilde{S}^{1}$. In this manner, we can hierarchically learn the adaptive graph structure in different layers of the network.

State-of-the-art GNNs [4] broadly follow the neighborhood aggregation strategy, which utilizes an aggregation function to gather feature messages from neighboring nodes and an update function to generate new node representations. Our proposed method is a general building block that can be easily integrated into various GNNs, including GCN [7], GAT [8], GraphSAGE [6], GIN [30], etc. Specifically, we simply adopt DGSLN to generate new graph topology before the GNN performs neighborhood aggregation, and utilize the resulting topology to guide the subsequent message passing. For simplicity, Section 4.1 details the integration of DGSLN in GCN.

3.3. Graph optimization via hybrid loss function

In this section, we further consider optimizing the learned graph structure to ensure the quality of the learned graphs. Previous work [10,15] learns graph structures from a single information source, and directly optimized the learned graphs by minimizing the task loss function. However, these approaches do not consider constraints from the underlying properties of the graph (e.g., sparsity and connectivity [16]), which may lead to a suboptimal solution for the learned graph structure. For this reason, we design a hybrid loss function for graph optimization. Specifically, we utilize regularization techniques to guide the learning of the graph structure, driving the learned graph to satisfy more graph attributes, and integrating these constraint terms into the task prediction loss as optimization objectives. By minimizing the hybrid loss function, the graph topology and GNN parameters are jointly learned from both data-driven and task-driven aspects. Formally, the loss function is defined as:

$$\mathcal{L} = \mathcal{L}_{pred} + \sum_{l} \mathcal{L}_{reg}^{l}, \tag{10}$$

where $\mathcal{L}_{pred}$ is the task prediction function (e.g., cross-entropy), and $\mathcal{L}_{reg}^{l}$ is the regularization function for optimizing the graph structure in layer l. Notice that $\mathcal{L}_{reg}^{l}$ can be dynamically adjusted, and selecting different regularization terms can capture different prior knowledge.

3.3.1. Feature smoothness

The first regularization term is introduced to control the feature smoothness of the learned graph $\tilde{S}^l$. Some studies [31,32] have demonstrated that the nodes connected in a graph are likely to have similar features. Meanwhile, feature smoothness is a metric reflecting the similarity of node representations. Thus, we can impose feature smoothness constraints in the objective function to assure the learned graph structure adapts to the node features, i.e.,

$$\mathcal{L}_{f}^{l} = \sum_{i,j \in \mathcal{V}} \phi(x_i, x_j), \tag{11}$$

where $\phi$is a metric function that measures the difference among nodes v_i and v_j. We can adopt typical measures such as Square Error, Kullback–Leibler(KL) divergence for ϕ, and we list the corresponding function below,

$$\begin{aligned} \phi_1 &= \sum_{i,j=1}^{n} |x_i - x_j|_F^2 \tilde{S}_{ij}^l = \mathrm{tr}\left(X^T L X\right), \\ \phi_2 &= \sum_{i,j=1}^{n} \mathrm{KL}(x_i|x_j) \tilde{S}_{ij}^l = \sum_{i,j=1}^{n} \left(x_i \log \frac{x_i}{x_j}\right) \tilde{S}_{ij}^l, \end{aligned} \tag{12}$$

where $\mathrm{tr}(\cdot)$ is the trace of a matrix, $L = D - \tilde{S}^l$ is the graph Laplacian matrix, and D is the degree matrix. If v_i and v_j are connected in $\tilde{S}^l$ (i.e., $\tilde{S}_{ij}^l \neq 0$), we expect the feature smoothness $\phi(x_i, x_j)$ to be small. In other words, smaller $\phi(x_i, x_j)$ indicates that the feature of v_i and v_j are quite similar, thus the larger value of $\tilde{S}_{ij}^l$. Therefore, minimizing $\mathcal{L}_f^l$ can encourage adjacent nodes to have similar features, thereby enhancing the feature smoothness on the learned graph $\tilde{S}^l$. However, this approach may produce a trivial solution $\tilde{S}^l = 0$ [33]. Since it not only forces similar nodes to connect, but also causes other nodes to disconnect.

3.3.2. Property constraint

To prevent $\tilde{S}^l$ from falling into the trivial solution. Meanwhile, the learned graph is constrained to satisfy more graph structure properties (such as sparsity and connectivity). We impose the second regularization term on the learned graph for exploring more prior information.

$$\mathcal{L}_p^l = -\lambda_1 \mathbf{1}^{\mathrm{T}} \log\left(\tilde{S}^l \mathbf{1}\right) + \frac{\lambda_2}{2} \left|\tilde{S}^l\right|_F^2, \tag{13}$$

where $\mathbf{1}^{\mathrm{T}} = [1, \ldots, 1]^{\mathrm{T}}$, $|\cdot|_F$ is the Frobenius norm, $\tilde{S}^l \mathbf{1}$ is the node degree vector. The first term applies the logarithmic barrier on $\tilde{S}^l \mathbf{1}$, which can force the degrees to be positive, but does not prevent the edges from being 0. In this way, the graph connectivity can be improved without affecting the sparsity. Nevertheless, adding only logarithmic term causes the graph to be quite sparse, thus we add the second term to control sparsity by penalizing large degrees.

In summary, we define the total regularization loss as follows:

$$\mathcal{L}_{reg}^l = \lambda_0 \mathcal{L}_f^l + \mathcal{L}_p^l, \tag{14}$$

where λ_0, λ_1, and λ_2 are predefined hyper-parameters that control the contribution of the regularization term to the overall task.

3.4. Complexity analysis

In this section, we analyze the model complexity. For DGSLN, the cost of learning the graph structure is $O(n^2 d)$, while computing graph sparse operation costs $O(n)$, computing gated fusion cost $O(n)$. Most existing GNNs [7,8,6] compute the node embedding costs $O(ndd' + |\mathcal{E}|d)$. GLCN [11] is a special case whose time complexity is $O(n^2d + ndd' + n^2d)$. Since GLCN learns the dense graphs to guide neighborhood aggregation. If DGSLN is integrated into a general GNN, we obtain the total time complexity $O(n^2d + 2n + ndd' + |\mathcal{E}|d)$. Among them, $n^2 \gg |\mathcal{E}| \gg n$ and $d \approx d'$. Thus, the computational complexity of our DGSLN is higher than the general GNNs, but much smaller than GLCN.

For the scalable version, the cost of learning the graph structure is $O(nsd)$. If it is integrated into the existing GNNs, we obtain the total time complexity $O(nsd + 2n + ndd' + |\mathcal{E}|d)$.

4. Model architecture

The proposed differentiable structure learning is a generic building block that can be easily integrated into various GNNs. To validate the generality and effectiveness, we redesign the graph convolution and graph pooling operations based on structure learning, and extended the redesigned operations to node classification and graph classification tasks.

4.1. Graph convolution with structure learning

Based on the proposed DGSLN, we redesign a new graph convolutional layer, termed as GSLConv. Different from [7,8], the new convolutional layer first learns the optimal graph adapted to the current node features, and then uses the learned graph to guide the feature propagation to generate new node representations.

Suppose the current feature matrix X^l and the optimal graph A^{l-1} of the previous layer are inputs. Based on Section 3.2, in the new convolutional layer, we first perform our proposed DGSLN to learn potential structural information $\widetilde{S}^l$ from X^l. Then, the optimal graph A^l of the current layer is obtained by Eq. (9), i.e., the previous graph structure A^{l-1} is added to the learned graph $\widetilde{S}^l$ through gated fusion. Finally, we utilize the optimal graph A^l to gather the neighborhood information for generating a new node representation. The layer-wise propagation process is defined as:

$$\begin{aligned} A^l &= \mathrm{DGSLN}\left(X^l, A^{l-1}\right), \\ X^{l+1} &= \sigma\left(\widehat{D}^{-\frac{1}{2}} A^l \widehat{D}^{-\frac{1}{2}} X^l W^l\right), \end{aligned} \tag{15}$$

where $W^l \in \mathbb{R}^{d \times d'}$ is the trainable weight, $X^{l+1} \in \mathbb{R}^{n \times d'}$ denotes the updated node features.

4.2. Graph pooling with structure learning

Some work [22,23] have achieved the graph coarsening by calculating the importance scores of the nodes in the graph and adaptively retaining several nodes with higher scores. However, these methods only retain a subset of graph nodes, it is inevitable to lose the integrity of the graph structure information. To solve the problem, we redesign a new graph pooling layer (GSLPool), which can reduce the size of graph and learn the topology information of the reduced graph to ensure the integrity of the graph structure.

Similar to the previous work [22], we first remove some task-irrelevant nodes to approximate the graph information. Specifically, we utilize the attention mechanism to evaluate the relative importance of each node, and perform topk selection to retain several critical nodes to form a new reduced graph A' and node features X^{l+1}. Since the unselected nodes are directly filtered during the selection process, which causes a massive loss of graph structure information. We employ our proposed DGSLN to learn the hidden edge connections from the reduced node features X^{l+1}, and combine them with the reduced graph A' to generate the output graph structure A^{l+1}. In this way, we can supplement the structure information lost in the pooling process and ensure the integrity of the reduced graph structure. The process can be formulated as:

$$\begin{aligned} p &= \mathrm{softmax}\left(\sigma\left(W^T X^l\right)\right), \\ \mathrm{idx} &= \mathrm{topk}(p, \lceil r \cdot N \rceil), \\ A' &= A^l_{(\mathrm{idx},\mathrm{idx})}, \\ X^{l+1} &= X^l_{(\mathrm{idx},:)}, \\ A^{l+1} &= \mathrm{DGSLN}\left(X^{l+1}, A'\right), \end{aligned} \tag{16}$$

where $p \in \mathbb{R}^{n \times 1}$ is the node importance scores, r is the pooling ratio, idx is the indices of the selected node, and $A^l_{(\mathrm{idx},\mathrm{idx})}$ and $X^l_{(\mathrm{idx},:)}$ perform the row/column extraction.

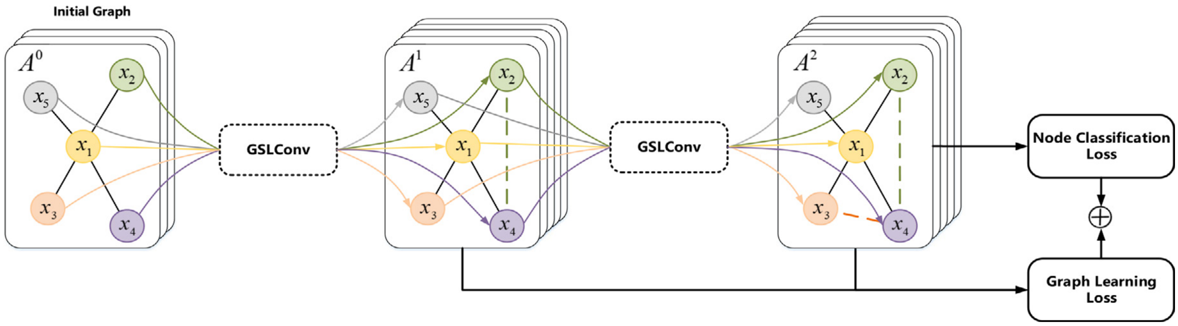


Fig. 3. Overview of Node Classification Architecture DGSLN$_n$. GSLConv represents the redesigned convolution operation.

4.3. Node classification

For a fair comparison, we adopt the network architectures similar to GCN [7], referred to as DGSLN$_n$. As shown in Fig. 3, we stack two GSLConv layers. In each GSLConv layer, we dynamically capture the optimal graph adapted to the current features, and then uses the learned graph to guide feature aggregation to generate new node representations. We apply the softmax function to output features to predict the label of each node, and the loss function is defined as the cross-entropy of all labeled nodes:

$$\mathcal{L}_{node} = -\sum_{i=1}^{n}\sum_{j=1}^{c} Y_{ij} \log \hat{Y}_{ij}, \tag{17}$$

where n is the set of labeled instances, c is the number of categories, $\hat{Y}_{ij}$ means the predicted probability, and Y_{ij} denotes the ground truth. Finally, we integrate the graph structure learning loss (Eq. (14))) into the task prediction loss, and optimize the network model by minimizing the hybrid loss function. For node classification tasks, the total loss function is defined as:

$$\mathcal{L} = \mathcal{L}_{node} + \sum_{l} \mathcal{L}_{reg}^{l}. \tag{18}$$

4.4. Graph classification

In this setting, we implement a hierarchical graph classification model recently proposed by [34], referred to as DGSLN$_h$. As illustrated in Fig. 4, the architecture consists of several blocks, each of which is stacked with a GSLConv layer followed by a GSLPool layer. GSLConv layer is responsible for aggregating neighborhood information to generate a hidden graph representation, while GSLPool reduces the graph size to extract higher-order features. The proposed model learns the graph representation in a hierarchical manner, thus we would observe that each block has a different size of the graph representation. The readout function [34] is utilized to aggregate the node features in the subgraph for generating a fixed-size global representation.

$$r = \frac{1}{n}\sum_{i=1}^{n} x_i || \max_{i=1}^{n} x_i, \tag{19}$$

where $||$is the concatenation operator.

Finally, the output of each block is added to generate a graph level representation, and the graph level representation is input to an MLP with softmax function for graph classification tasks. Similar to the node classification task, its loss function can be defined as:

$$\mathcal{L} = \mathcal{L}_{graph} + \sum_{l} \mathcal{L}_{reg}^{(l)}. \tag{20}$$

5. Experimental results

5.1. Datasets

For graph classification task, we carry out comprehensive experiments on five commonly used public benchmark datasets: D&D, PROTEINS, NCI109, NCI1, COLLAB, and Mutagenicity [23].

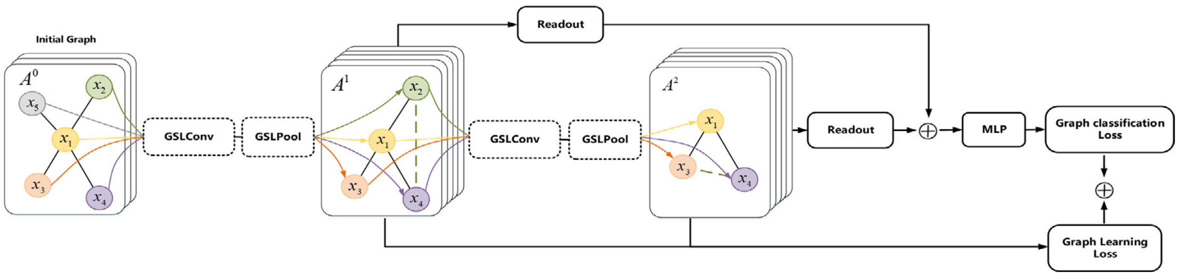


Fig. 4. Overview of Garph Classification Architecture DGSLN$_h$. GSLConv represents the redesigned convolution operation, GSLPool represents the redesigned pooling operation.

For node classification task, we employ three popular citation network benchmark datasets: Citeseer, Cora, and Pubmed [3]. To further validate the scalability of our proposed model, we utilize a large graph dataset: CoraFull, which is an extended version of Cora. The statistics of these datasets are depicted in Table 1.

5.2. Baselines

5.2.1. Graph classification task

We compare our method with some representative and state-of-the-art GNNs and graph pooling methods. The classic GNNs include: GCN [7], GraphSAGE with mean aggregator [6], GAT [8], and GIN [30]. Since the group of methods does not include any pooling layer, we directly input the learned graph representation into the readout function for graph classification.

Graph pooling methods can be grouped into two main categories: (1) Global pooling methods, which contain multiple graph convolutional layers but only one pooling layer. The global pooling methods include: SortPool [35], and SAGPool$_g$ [23] that adopts the global pooling architecture for graph classification. (2) Hierarchical pooling methods, which contain multiple graph convolutional layers and pooling layers. Six hierarchical pooling methods are used as baselines: DiffPool [21], gPool [22], SAGPool$_h$ that adopts the hierarchical pooling architecture for graph classification [23], EigenPool [36], HaarPool [20], HGP-SL [15] which introduces a structure learning method in graph pooling operations to ensure graph integrity.

5.2.2. Node classification task

We first compare our model against the baseline of GCN [7], which is the most relevant model to our model. We then against some other graph neural networks, including GAT [8], SGC [37], GMNN [38], GLCN [11], GRCN [12], and GDC[39]. Among them, GLCN and GRCN are graph structure learning methods, which learn an optimal graph structure by combining graph convolution and graph structure learning in a unified network structure. GDC is a diffusion model-based approach for improving graph structure. To facilitate a fair comparison, all baselines adopt the identical setting.

5.2.3. DGSLN variants

Some variants are constructed to further validate the effectiveness of the proposed DGSLN: DGSLN$_n$ that removes the pooling layer and the structure learning is only considered in the convolutional layer. DGSLN$_g$ adopts the global pooling architecture for graph classification. **DGSLN-Fast** leverages the proposed scalable relationship learning technique in Section 3.2.1.

5.3. Implementation details

5.3.1. Graph classification settings

Our experimental setup is consistent with previous work [22,23] and adopts the same data partition as in [23]. We implement our methods utilizing PyTorch[1], and the Adam optimizer is utilized to optimize the network. For all methods and datasets, the dimension of the node embedding is set to 128. We set the learning rate to 0.0005, the weight decay to 0.0001, the pooling ratio r to 0.5, the layers to 3, the hyper-parameter $k \in [2,4]$, the trade-off parameter λ_0 to 0.0005, λ_1 to 0.1, and λ_2 to 0.01. For DGSLN-Fast, we set $s = 0.4$. The MLP consists of three fully connected layers, the hidden sizes set to $256, 128, 64$, respectively. The early stop strategy is adopted during training, i.e., if the validation loss does not improve within 50 epochs or in the iteration number exceeds a maximum of 1000 epochs, the training is stopped. For a fair comparison, the pooling rate, learning rate, and weight decay are set as the same in all the hierarchical pooling methods (including gPool, SAGPool$_h$ and HGP-SL).

[1] The source code can be provided on request and will be released to the public after this paper review to facilitate more research.

Table 1
Dataset statistics.

Dataset Type	Dataset	Graphs	Avg. nodes	Avg. Edges	Classes
Graph classification	D&D	1,178	284.32	715.66	2
	PROTEINS	1,113	39.06	72.82	2
	NCI109	4,127	29.68	32.13	2
	NCI1	4,110	29.87	32.30	2
	COLLAB	5,000	74.49	2457.78	3
	Mutagenicity	4,337	30.32	30.77	2
Dataset Type	Dataset	Nodes	Edges	Feature	Classes
Node classification	Cora	2708	5,429	1433	7
	Citeseer	3327	4,732	3703	6
	Pubmed	19717	44,338	500	3
	CoraFull	19793	65,311	8710	70

5.3.2. Node classification settings

For a fair comparison, we adopt the network architecture consistent with [7]. We set the network depth to 2, and the dimension of feature vector in each layer to 16. We apply dropout of $p = 0.5$ for the input in each layer, and adopt LeakyReLU as the activation function. We utilize Adam optimization to train proposed models, the maximum of training iterations are 200 epochs, the learning rate is 0.1, and the weight decay is 0.0005.

5.4. Performance comparison

5.4.1. Graph classification

Table 2 demonstrates the average accuracy of our proposed DGSLN compared with other baselines. From this table, we can obtain several insights:

First, we can easily observe that the proposed DGSLN performs significantly better than other baseline methods in 5 out of 6 benchmarks. The results demonstrate that the proposed method can adaptively learn useful structural information and effectively improve the ability of representation learning.

Second, our approach is significantly better than all methods that do not consider structure learning, including classic GNNs and graph pooling methods. Particularly, our approach surpasses the recently proposed EigenPool and HaarPool among all datasets. This effectively validates the necessity of learning appropriate graph structures in GNNs.

Third, HGP-SL performs better than previous methods that do not apply structure learning. One possible reason is that HGP-SL applies structure learning in the pooling operations to learn the appropriate graph structure and obtain a better graph representation. Compared with HGP-SL, our method achieves better performance. This is because DGSLN is able to learn the optimal graph adapted to node features from both task-driven and data-driven aspects through the designed hybrid loss function.

Fourth, the performance of $DGSLN_n$ is better than GCN, which proves that the redesigned convolutional layer can learn the appropriate graph structure and generate a better node representation. Furthermore, the performance of $DGSLN_g$ and $DGSLN_h$ is better than $DGSLN_n$, which indicates that the redesigned pooling layer can ensure the integrity of the reduced graph after pooling while encoding advanced features. This proves that our DGSLN is a general building block that can hierarchically learn the adaptive graph structure in different layers of the network.

Finally, $DGSLN_h$-Fast can achieve comparable or even better performance than $DGSLN_h$. This indicates that $DGSLN_h$-Fast can efficiently learn the appropriate graph topology while significantly reducing the computational complexity, and increasing the scalability of DGSLN.

5.4.2. Node classification

Table 3 demonstrates the comparison results of the benchmark datasets. Our $DGSLN_n$ performs better on all datasets than the baselines that do not consider graph structure learning, including GCN, GAT, SGC, and the recently proposed GMNN. It effectively validates the importance of learning appropriate graph structure in GNN. Moreover, our approach outperforms GLCN, GRCN, and GDC, which demonstrates the superiority of our proposed structural learning algorithm. Different from GLCN which learns a dense fully connected graph and applies it to the entire network, DGSLN can dynamically learn the adaptive sparse graph structure of different layers, thus obtaining a better node representation. Compared with GRCN and GDC, DGSLN considers more graph properties and learns a more reasonable graph structure. Finally, we can find that $DGSLN_n$-Fast performs nearly as well as $DGSLN_n$. On the large dataset CoraFull, $DGSLN_n$ fails due to memory limitations, while $DGSLN_n$-Fast achieves the best performance. It demonstrates that $DGSLN_n$-Fast can significantly reduce the memory and space consumption, and effectively address the scalability problem of DGSLN.

Table 2
Overview of graph classification results, and the symbol '-' means the results are unavailable.

Categories	Baselines	PROTEINS	D&D	NCI1	NCI109	COLLAB	Mutagenicity
GNNs	GCN [7]	74.17 ± 1.63	75.26 ± 2.46	72.49 ± 1.79	70.70 ± 1.84	80.60 ± 2.10	78.01 ± 1.58
	GraphSAGE [6]	74.60 ± 4.27	76.58 ± 3.91	73.23 ± 1.34	70.37 ± 2.89	79.70 ± 1.70	78.55 ± 1.18
	GAT [8]	74.72 ± 4.01	77.30 ± 3.68	73.90 ± 1.72	75.81 ± 2.68	-	78.89 ± 2.05
	GIN [30]	76.20 ± 2.80	77.78 ± 1.81	$\mathbf{80.14 \pm 1.40}$	78.64 ± 1.40	79.30 ± 2.70	79.93 ± 1.25
Global	SortPool [35]	73.20 ± 0.44	77.76 ± 1.21	72.98 ± 2.19	72.23 ± 1.31	71.38 ± 0.98	78.78 ± 1.02
	SAGPool_g [23]	70.04 ± 1.47	76.19 ± 0.94	74.18 ± 1.20	74.06 ± 0.78	74.48 ± 0.78	77.35 ± 0.67
Hierarchical	DiffPool [21]	75.54 ± 2.02	78.05 ± 2.41	75.32 ± 1.90	74.18 ± 1.98	69.18 ± 1.71	71.34 ± 0.89
	gPool [22]	71.10 ± 0.90	75.01 ± 0.86	67.02 ± 2.25	66.12 ± 1.60	71.26 ± 1.39	72.12 ± 1.37
	SAGPool_h [23]	71.86 ± 0.94	75.45 ± 0.97	67.45 ± 1.11	67.86 ± 1.41	72.76 ± 1.74	77.02 ± 1.23
	EigenPool [36]	77.12 ± 1.25	76.77 ± 1.65	76.65 ± 1.12	74.82 ± 1.35	77.25 ± 1.21	79.53 ± 1.04
	HaarPool [20]	80.4 ± 1.8	-	78.6 ± 0.5	75.6 ± 1.2	-	80.9 ± 1.5
	HGP-SL [15]	83.92 ± 1.65	77.12 ± 1.25	77.13 ± 1.12	76.33 ± 1.35	-	78.85 ± 1.04
Ours	DGSLN_n	82.76 ± 0.85	76.95 ± 0.56	75.22 ± 0.92	76.94 ± 2.19	81.82 ± 1.27	80.15 ± 0.76
	DGSLN_g	83.37 ± 1.17	78.55 ± 1.03	79.76 ± 1.28	$\mathbf{80.34 \pm 0.59}$	79.86 ± 0.85	$\mathbf{81.92 \pm 0.99}$
	DGSLN_h	$\mathbf{86.84 \pm 0.76}$	$\mathbf{79.97 \pm 0.75}$	76.92 ± 1.07	77.20 ± 0.84	$\mathbf{82.67 \pm 0.97}$	80.48 ± 0.84
	DGSLN_h-Fast	85.98 ± 0.54	79.07 ± 0.66	76.56 ± 0.71	77.26 ± 0.69	82.06 ± 0.81	80.15 ± 1.12

Table 3
Overview of node classification results. "oom" means out of memory.

Baselines	Citeseer	Cora	Pubmed	CoraFull
GCN [7]	70.3 ±0.7	81.5 ±0.6	78.4 ±0.6	60.3 ±0.7
GAT [8]	72.5 ±0.7	83.0 ±0.7	79.0 ±0.3	59.9 ±0.6
SGC [37]	71.9 ±0.1	81.0 ±0.0	78.9 ±0.0	59.1 ±0.7
GMNN[38]	73.1	83.7	79.4	-
GLCN [11]	72.4 ±0.4	83.4 ±0.5	79.1 ±0.4	59.1 ±0.7
GRCN [12]	72.6 ±1.3	83.7 ±1.7	77.9 ±3.2	60.2 ±0.5
GDC [39]	73.2 ±0.3	83.6 ±0.2	78.7 ±0.4	59.5 ±0.4
DGSLN_n	**73.4** ±0.8	**84.2** ±0.5	**79.4** ±0.2	oom
DGSLN_n-Fast	72.6 ±0.5	83.9 ±0.5	79.1 ±0.2	**60.7** ±**0.4**

5.5. Ablation study

5.5.1. Robustness analysis

To verify the robustness of DGSLN, we randomly remove edges on Cora dataset to generate a synthetic dataset. Specifically, the edge deletion rates are $20\%, 40\%, 60\%, 80\%$. Fig. 5 shows that the classification accuracy of GCN decreases as the edge deletion rate increases, which means that GNN is very sensitive to the quality of the original graph structure. Compared with GCN and GRCN, DGSLN achieves better results. It indicates that the proposed DGSLN can effectively capture meaningful graph topology without relying on the initial graph structure. In particular, our method shows the more significant performance improvement when the edge deletion rate becomes larger. These results all demonstrate that the proposed DGSLN can learn the more robust graph representation.

5.5.2. Impact of different metric functions

To explore the impact of different metric functions on pairwise relationship learning, we design three variants: DGSLN_h-ED and DGSLN_h-CS represent adopting the euclidean distance (ED) and cosine similarity (CS) to obtain the pairwise relationship. DGSLN_h-MLP means directly employing the single-layer MLP proposed by GLCN [11] for graph learning.

From Table 4, we can observe that DGSLN_h-MLP is superior to DGSLN_h-ED and DGSLN_h-CS, which indicates that a good metric function should be learnable. The proposed DGSLN_h and DGSLN_h-Fast further improve the performance. This is because the attention mechanism can explicitly capture the relationship importance of the graph, driving the model to focus on more important edges.

5.5.3. Effects of hierarchical graph learning

To investigate the effectiveness of hierarchical graph learning, we design two variants: DGSLN_g-fixed and DGSLN_h-fixed, which remove the hierarchical graph learning component (i.e., only learn a fixed graph from the input node feature). Table 5 illustrates the comparison results on three graph classification datasets. It is obvious that GSLNN_g and DGSLN_h perform better than GSLNN_g-fixed and DGSLN_h-fixed significantly. It demonstrates that hierarchical graph learning can dynamically capture the graph structure adapted to node features and improve the representational power of the model.

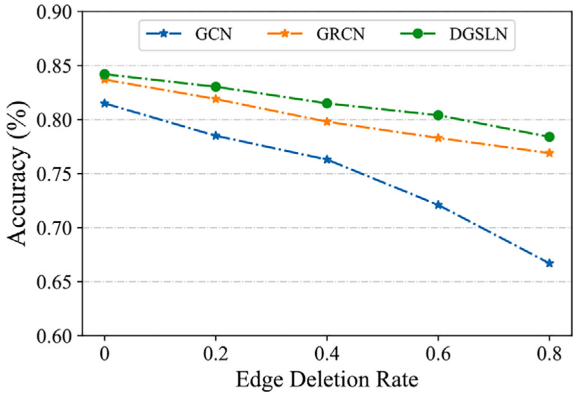


Fig. 5. Test accuracy on the synthetic dataset.

5.5.4. Effects of differentiable graph sparsification

Five variants are explored to evaluate the effectiveness of graph sparse operation. (1) HGP-SL-Den means adopting softmax to learn dense graph structure. (2) HGP-SL-SO means removing sparsemax in HGP-SL, and adopting the proposed graph sparse method. (3) DGSLN$_h$-Den means discarding sparse operations and learning dense graph structure with softmax. (4) DGSLN$_h$-SM stands for utilizing sparsemax in DGSLN$_h$ to obtain a sparse structure. (5) DGSLN$_h$-kNN stands for utilizing the settings of kNN algorithm to obtain a sparse structure.

Table 6 presents the experimental results on two graph classification datasets. We can observe that HGP-SL-Den and DGSLN$_h$-Den achieve the worst performance, and DGSLN$_h$-Den is unavailable on D&D dataset due to out of memory. It demonstrates that learning dense graph structure not only affects scalability, but also might introduce additional noise information and cause performance degradation. Whether it is the baseline HGP-SL or DGSLN, our proposed sparse operation can achieve better performance than sparsemax. More importantly, we can observe from the table that its training speed is 3 or 4 times faster than sparsemax. It indicates that our method can efficiently generate sparse distributions and learn more reasonable graph structures without introducing too much extra computational overhead. Moreover, DGSLN$_h$ is also superior to DGSLN$_h$-kNN, which further verifies the effectiveness of the proposed sparse operation.

5.5.5. Effects of gated integration

To evaluate the proposed gated fusion mechanism, we explore some variants of DGSLN$_h$: (1) DGSLN$_h$-NIG denotes discarding the original graph structure and only utilizing the graph structure learned by DGSLN. (2) DGSLN$_h$-Avg means replacing the gated fusion with the simple addition, i.e., the new graph topology is obtained by an average aggregation.

We present the experimental results in Table 7. The results demonstrate that DGSLN$_h$ performs better than DGSLN$_h$ and DGSLN$_h$-Avg. This is because the given graph structure contains rich and useful graph topology information, and discarding the original graph structure will cause performance degradation. Furthermore, our proposed gating Integration mechanism can effectively fuse the raw graph structure and the learned graph by learning adaptive weights, thereby forming a fused graph structure that adapts to the node features.

5.5.6. Effects of hybrid loss function

In this section, some variants are designed to validate the effectiveness of the hybrid loss function: (1) DGSLN$_h$-NHL means removing the graph regularization loss in DGSLN$_h$. (2) DGSLN$_h$-FS denotes only adopting the feature smoothness constrain. (3) DGSLN$_h$-P means only adopting the property constraint. (4) DGSLN$_h$-GL denotes adopting the graph learning loss proposed by GLCN [11] to replace the proposed hybrid loss.

Table 8 depicts the performance of the four variants and our proposed DGSLN$_h$. From the results, we can observe that the variant DGSLN$_h$-NHL without hybrid loss function gets the worst performance. It shows that it is difficult to learn the adap-

Table 4
Impact of Different Metric Functions in the Pairwise Relationship Learning.

Methods	PROTEINS	D&D	NCI1
DGSLN$_h$-ED	82.14 ± 1.14	76.33 ± 0.76	74.93 ± 1.03
DGSLN$_h$-CS	83.92 ± 0.81	76.42 ± 1.05	75.13 ± 1.79
DGSLN$_h$-MLP	85.71 ± 0.82	78.45 ± 1.16	75.45 ± 0.91
DGSLN$_h$	$\mathbf{86.84 \pm 0.76}$	$\mathbf{79.97 \pm 0.75}$	$\mathbf{76.92 \pm 1.07}$
DGSLN$_h$-**Fast**	85.98 ± 0.54	79.07 ± 0.66	76.56 ± 0.71

Table 5
Effectiveness of the Hierarchical Graph Learning.

Methods	PROTEINS	D&D	NCI1
DGSLN$_g$-fixed	80.92 $\pm$ 1.72	77.12 $\pm$ 1.05	77.13 $\pm$ 1.79
DGSLN$_g$	83.37 $\pm$ 1.17	78.55 $\pm$ 1.03	**79.76** $\pm$ **1.28**
DGSLN$_h$-fixed	83.92 $\pm$ 1.14	77.66 $\pm$ 1.03	74.13 $\pm$ 0.73
DGSLN$_h$	**86.84** $\pm$ **0.76**	**79.97** $\pm$ **0.75**	76.92 $\pm$ 1.07

Table 6
Effectiveness of the Graph Sparsification.

Methods	D&D		PROTEINS	
	Accuracy (%)	Training time (s)	Accuracy (%)	Training time (s)
HGP-SL-Den	76.02 $\pm$ 1.09	7.5157	83.12 $\pm$ 1.37	1.5635
HGP-SL	77.12 $\pm$ 1.25	13.1136	83.92 $\pm$ 1.65	3.9464
HGP-SL-SO	77.12 $\pm$ 0.92	2.823	84.82 $\pm$ 1.06	1.2396
DGSLN$_h$-Den	oom	-	84.66 $\pm$ 1.08	1.7992
DGSLN$_h$-SM	78.47 $\pm$ 1.65	16.425	85.62 $\pm$ 1.26	3.6464
DGSLN$_h$-kNN	77.99 $\pm$ 1.23	4.9223	85.24 $\pm$ 0.86	1.2678
DGSLN$_h$	**79.97** $\pm$ **0.75**	4.7774	**86.84** $\pm$ **0.76**	1.0947

Table 7
Effectiveness of the Gated Fusion Mechanism.

Methods	PROTEINS	D&D	NCI1
DGSLN$_h$-NIG	83.92 $\pm$ 1.72	77.12 $\pm$ 1.05	74.13 $\pm$ 1.79
DGSLN$_h$-Avg	85.71 $\pm$ 0.82	78.45 $\pm$ 1.16	75.45 $\pm$ 0.91
DGSLN$_h$	**86.84** $\pm$ **0.76**	**79.97** $\pm$ **0.75**	**76.92** $\pm$ **1.07**

Table 8
Effectiveness of Hybrid Loss Function.

Methods	PROTEINS	D&D	NCI1
DGSLN$_h$-NHL	84.51 $\pm$ 1.67	77.45 $\pm$ 1.10	74.45 $\pm$ 1.42
DGSLN$_h$-FS	85.32 $\pm$ 1.23	78.86 $\pm$ 1.49	75.47 $\pm$ 1.06
DGSLN$_h$-P	85.08 $\pm$ 1.65	78.56 $\pm$ 1.25	75.34 $\pm$ 1.12
DGSLN$_h$-GL	85.58 $\pm$ 0.74	78.99 $\pm$ 0.64	75.66 $\pm$ 0.43
DGSLN$_h$	**86.84** $\pm$ **0.76**	**79.97** $\pm$ **0.75**	**76.92** $\pm$ **1.07**

tive graph structure only by minimizing the task loss function during the training phase. DGSLN$_h$-FS and DGSLN$_h$-P outperform DGSLN$_h$-NHL, which shows that the proposed feature smoothness and property constraint can effectively control the feature smoothness, sparsity and connectivity of the learned graph. DGSLN$_h$ combines two regularization terms to obtain the optimal performance. It reveals the effectiveness of the proposed hybrid loss function, which can force the learned graph to satisfy more graph properties.

5.5.7. Generality of DGSLN

As mentioned in Section 3.2.3, our proposed DGSLN can be integrated into various GNNs architecture. In order to verify the generality of DGSLN, we integrate it into the three most widely used graph convolution operations: GCN, GraphSAGE with mean aggregator, and GAT, and then integrate them into DGSLN$_h$ as a basic building block. Table 9 shows that the performance of the three variants on graph classification tasks. These results indicate that the three variants have achieved good performance. The model performance depends on the selected dataset and the GNN type, thus verifying the effectiveness and flexibility of the proposed structure learning.

5.6. Visualization

5.6.1. Visualization of graph structure changes

To observe the graph structure changes brought by DGSLN intuitively, we visualize it on Cora dataset and PROTEINS dataset. Fig. 1 shows the original graph structure and the learned graph structure on Cora dataset. From Fig. 1(a), we can observe

Table 9
Results of $DGSLN_h$ and its variants on graph classification task.

Methods	PROTEINS	D&D	NCI1
$DGSLN_h$	**86.84** $\pm$ **0.76**	**79.97** $\pm$ **0.75**	**76.92** $\pm$ **1.07**
$DGSLN_h$-GAT	85.62 ± 1.26	78.47 ± 1.65	75.47 ± 1.12
$DGSLN_h$-SAGE	83.51 ± 1.67	77.45 ± 1.10	74.45 ± 1.42

that the original graph structure contains missing edges and some nodes do not have edge connections. While our proposed DGSLN can effectively capture more meaningful topology information, which complements the original graph structure. Fig. 1(c) further zooms in on the local details of the learned graph structure, and illustrates the neighborhood changes of the rightmost green node in Fig. 1(a). Fig. 6 illustrates the examples of graph structure changes for $SAGPool_h$ and $DGSLN_h$ on PROTEINS dataset. The results show that $SAGPool_h$ cannot retain meaningful global topology and graph connectivity is severely disrupted. In contrast, $DGSLN_h$ preserves the relatively reasonable topology of the protein graph by learning the graph structure and obtains superior performance.

5.6.2. T-SNE visualization

Fig. 7 presents t-SNE visualization of graph representations learned by $DGSLN_h$, HGP-SL and $SAGPool_h$ on NCI and PROTEINS. In $SAGPool_h$, the samples are not well clustered, and the boundaries between graph classes are not clear. In contrast, $DGSLN_h$ shows a good clustering for graph samples, which means that $DGSLN_h$ can learn more meaningful representations by capturing the adaptive graph structure.

5.7. Hyper-parameter analysis

The selection of hyper-parameters may directly affect the performance of the proposed model. In this section, we alter the values of hyper-parameters $s, k, r, \lambda_0, \lambda_1$ and λ_2 to search for the optimal parameter values.

5.7.1. Analysis on hyper-parameter s

In the proposed scalable version, s directly affects the complexity of graph structure learning. A larger s results in higher model complexity. Fig. 8(a) presents the results of $DGSLN_h$ on different s. From this figure, increasing the value of s is helpful to improve the classification accuracy, but when s is greater than 0.4, the performance improvement begins to slow down. The weak performance gains are negligible for the added computational complexity, especially on large graphs. Thus, we set $s = 0.4$ to carry out the trade-off between accuracy and complexity.

5.7.2. Analysis on hyper-parameter k

In the proposed graph sparse operation, the degree of graph sparseness is affected by the value of k. Specifically, a larger k results in a dense graph and causes high computational overhead, while a smaller k may lose a lot of the learned structural information. Therefore, it is critical to choose an appropriate value of k. In Fig. 8(b), we study the effect of different k values on model performance. The results show that the optimum result differs for different datasets. The larger k value is required for D&D dataset that contains many nodes and edges. We consistently find the k value of around $[2, 4]$ to perform best. Since different graphs have different structures, the k should be adjusted for the dataset under study.

5.7.3. Analysis on hyper-parameter r

We provide an ablation study on the pooling ratio r. Fig. 8(c) displays the classification accuracy under different r values. The results demonstrate that the optimal r has different values for different datasets. Therefore, the appropriate r value needs to be selected according to the graph data and application. Note that r cannot be too small, otherwise rich structural message will be lost during the pooling process, resulting in performance degradation.

5.7.4. Choices of ϕ

We adopt two metric functions to measure feature smoothness in Eq. (12). For facilitate selection, Fig. 8(d) reports the classification accuracy of using Square error and KL divergence on three datasets. We can observe that the two metrics have similar performance, but KL divergence has better stability. Therefore, we give priority to KL divergence when expanding to new datasets.

5.7.5. Analysis on hyper-parameter $\lambda_0, \lambda_1, \lambda_2$

In the hybrid loss function (Eq. (13)), the weight parameters $\lambda_0, \lambda_1, \lambda_2$ are the key parameters that affect the model performance. In particular, $\lambda_0, \lambda_1, \lambda_2$ can scale the loss values to the same scale level and assign different weights to them during optimization. Here, we evaluate the contribution of the hybrid loss function with different $\lambda_0, \lambda_1, \lambda_2$. As illustrated in Fig. 9(a)-9(c), $DGSLN_h$ achieves the optimal accuracy on PROTEIN dataset when $\lambda_0 = 0.0005, \lambda_1 = 0.1$ and $\lambda_2 = 0.01$.

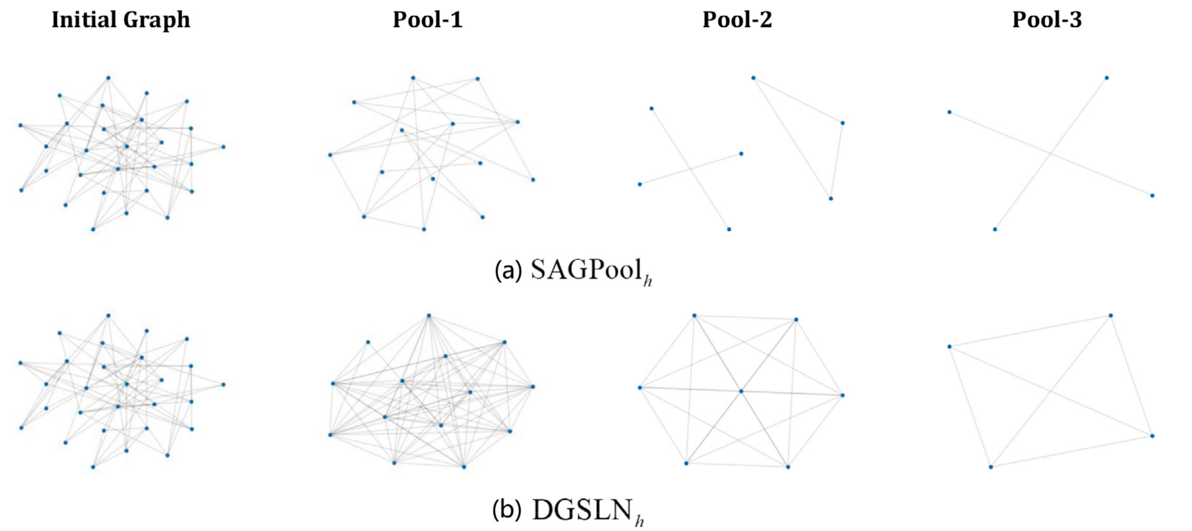


Fig. 6. Illustrative examples of graph structure changes for SAGPool$_\mathrm{h}$ and DGSLN$_h$ on PROTEINS dataset.

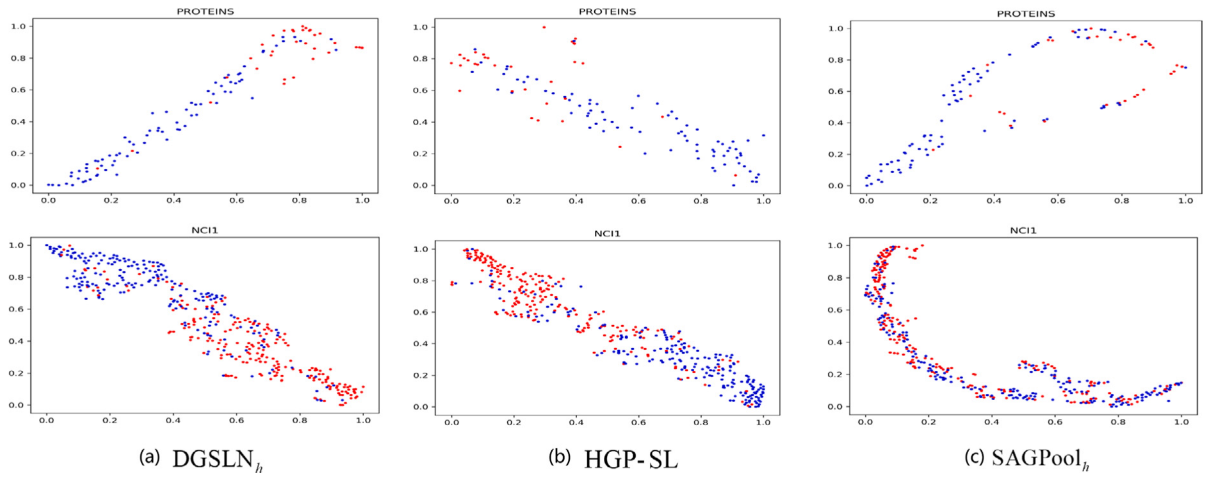


Fig. 7. T-SNE visualization of graph representations learned by SAGPool$_\mathrm{h}$, HGP-SL and our proposed DGSLN$_h$. Each color represents a category.

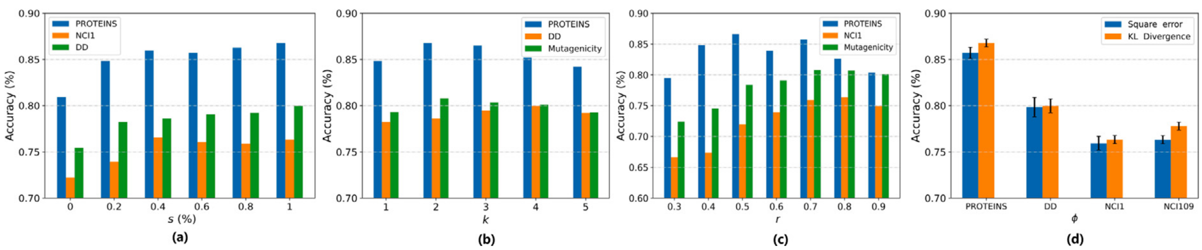


Fig. 8. Hyper-parameter sensitivity analysis.

5.7.6. Influence of network depth

Network depth is a key parameter, which directly affects the quality of feature learning. From Fig. 9(d), we can find that when the network depth is 3, the performance model is better than the rest of the models, and when the network depth deviates from 3, the performance of the model usually decreases. One possible reason is that the shallower network is not enough to learn the effective representation, and the deeper network will lead to over-fitting problems.

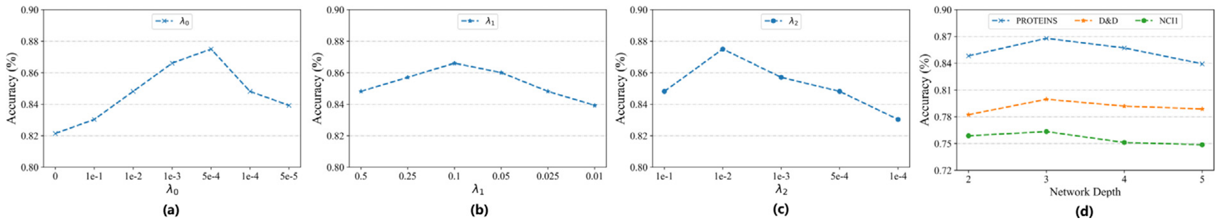


Fig. 9. (a-c) Experiments on model performance with different $\lambda_0, \lambda_1, \lambda_2$ values in the hybrid loss function. (d) Influence of the network depth.

Table 10
Model sizes, training time (training for one epoch), testing time comparison (in seconds).

Methods	D&D			PROTEINS			NCI1		
	Model size (KB)	Training time	Testing time	Model size (KB)	Training time	Testing time	Model size (KB)	Training time	Testing time
SAGPool_g	596	0.7681	0.0767	552	0.4669	0.0536	572	2.2725	0.2842
SAGPool_h	340	2.9658	0.1004	300	0.8716	0.0505	316	3.2751	0.2175
EigenPool	1928	39.0185	4.6332	1928	1.6221	0.3392	1928	5.3354	0.6860
HGP-SL	1928	13.1136	1.6396	303	3.9464	0.4558	249	18.2076	2.2706
DGSLN_g	598	0.8775	0.0869	554	0.8643	0.0621	572	2.7114	0.1758
DGSLN_g**-Fast**	598	0.6234	0.0671	554	0.6143	0.0509	572	1.6142	0.1152
DGSLN_h	343	4.7774	0.3188	300	1.0947	0.0902	317	4.0055	0.3374
DGSLN_h**-Fast**	343	2.2747	0.2162	300	0.5247	0.0623	317	1.9047	0.2144

5.8. Model storage and running time comparison

In this section, we study the model storage and running time of our method compared to other baselines. Table 10 presents the model sizes, training time (training for one epoch), testing time of DGSLN_g, DGSLN_g-Fast, DGSLN_h, DGSLN_h-Fast, and other baselines. To facilitate fair calculation, only one NVIDIA P100 GPU is used for all the methods. We can discover that the model size and running time of DGSLN is bigger than SAGPool in both global architecture and hierarchical architecture. While the running time of DGSLN-Fast is similar to SAGPool. Moreover, EigenPool and HGP-SL have the largest model sizes and running time especially on D&D which has many nodes and edges. These results show that our DGSLN can simply and efficiently obtain better representations.

6. Conclusion and future work

In this work, we explore graph structure learning in GNNs to learn robust graph structures for guiding message passing. Specifically, we propose a differentiable graph structure learning neural network (DGSLN), which utilizes the attention mechanism to learn an adaptive graph topology from node features. To learn more reasonable structures, we further propose a differentiable graph sparse operation, which can transform dense fully connected graphs into sparse graphs. We further design a hybrid loss function, so that graph representation and graph structure can be co-optimized. DGSLN is a flexible building block that can be integrated into any GNNs framework. Extensive experiments have demonstrated that our DGSLN can effectively improve performance compared to state-of-the-art GNNs methods.

In future work, we aim to extend the proposed structure learning method to unstructured point cloud learning.

CRediT authorship contribution statement

Xiaofeng Zou: Conceptualization, Methodology. **Kenli Li:** Supervision, Writing – review & editing, Funding acquisition. **Cen Chen:** Data curation, Validation. **Xulei Yang:** Visualization. **Wei Wei:** Investigation. **Keqin Li:** Writing – review & editing.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

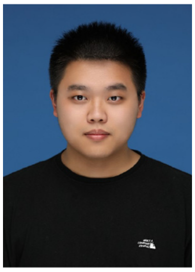
This work is supported by the National Natural Science Foundation of China under Grant 62172151 and 61860206011, the Cultivation of Shenzhen Excellent Technological and Innovative Talents (Ph.D. Basic Research Started)

RCBS20200714114943014, and by the Basic research of Shenzhen Science and technology Plan under Grant JCYJ20210324123802006, and the Postgraduate Scientific Research Innovation Project of Hunan Province under Grant CX20210407.

References

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 770–778, 2016.
[2] Yuebing Zhang, Zhifei Zhang, Duoqian Miao, Jiaqi Wang, Three-way enhanced convolutional neural networks for sentence-level sentiment classification, Inf. Sci. 477 (2019) 55–64.
[3] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, Tina Eliassi-Rad, Collective classification in network data, AI magazine 29 (3) (2008) 93.
[4] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S Yu. A comprehensive survey on graph neural networks. arXiv preprint arXiv:1901.00596, 2019.
[5] Dongsheng Luo, Wei Cheng, Wenchao Yu, Bo Zong, Jingchao Ni, Haifeng Chen, and Xiang Zhang. Learning to drop: Robust graph neural network via topological denoising. In Proceedings of the 14th ACM International Conference on Web Search and Data Mining, pages 779–787, 2021.
[6] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In Advances in neural information processing systems, pages 1024–1034, 2017.
[7] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In International Conference on Learning Representations, 2017.
[8] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. In International Conference on Learning Representations, 2018.
[9] Luca Franceschi, Mathias Niepert, Massimiliano Pontil, and Xiao He. Learning discrete structures for graph neural networks. In International Conference on Machine Learning, pages 1972–1982, 2019.
[10] Ruoyu Li, Sheng Wang, Feiyun Zhu, and Junzhou Huang. Adaptive graph convolutional neural networks. In Proceedings of the AAAI conference on artificial intelligence, volume 32, 2018.
[11] Bo Jiang, Ziyan Zhang, Doudou Lin, Jin Tang, and Bin Luo. Semi-supervised learning with graph learning-convolutional networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 11313–11320, 2019.
[12] Donghan Yu, Ruohong Zhang, Zhengbao Jiang, Yuexin Wu, Yiming Yang, Graph-revised convolutional network, in: Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2020, Springer, 2020.
[13] Guangtao Wang, Rex Ying, Jing Huang, and Jure Leskovec. Improving graph attention networks with large margin-based constraints. arXiv preprint arXiv:1910.11945, 2019.
[14] Yujun Yan, Milad Hashemi, Kevin Swersky, Yaoqing Yang, and Danai Koutra. Two sides of the same coin: Heterophily and oversmoothing in graph convolutional neural networks. arXiv preprint arXiv:2102.06462, 2021.
[15] Zhen Zhang, Jiajun Bu, Martin Ester, Jianfeng Zhang, Chengwei Yao, Zhi Yu, and Can Wang. Hierarchical graph pooling with structure learning. arXiv preprint arXiv:1911.05954, 2019.
[16] Xiang Gao, Wei Hu, and Zongming Guo. Exploring structure-adaptive graph learning for robust semi-supervised classification. In IEEE International Conference on Multimedia and Expo, 2020.
[17] Wei Jin, Yao Ma, Xiaorui Liu, Xianfeng Tang, Suhang Wang, and Jiliang Tang. Graph structure learning for robust graph neural networks. In Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pages 66–74, 2020.
[18] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Advances in neural information processing systems, pages 5998–6008, 2017.
[19] Filippo Maria Bianchi, Daniele Grattarola, Cesare Alippi, Spectral clustering with graph neural networks for graph pooling, in: International Conference on Machine Learning, PMLR, 2020, pp. 874–883.
[20] Yu Guang Wang, Ming Li, Zheng Ma, Guido Montufar, Xiaosheng Zhuang, Yanan Fan, Haar graph pooling, in: International Conference on Machine Learning, PMLR, 2020, pp. 9952–9962.
[21] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L. Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In Advances in Neural Information Processing Systems, pages 4800–4810, 2018.
[22] Hongyang Gao and Shuiwang Ji. Graph u-nets. In International Conference on Machine Learning, pages 2083–2092, 2019.
[23] Junhyun Lee, Inyeop Lee, and Jaewoo Kang. Self-attention graph pooling. In International Conference on Machine Learning, pages 3734–3743, 2019.
[24] Kihyuk Sohn, Honglak Lee, and Xinchen Yan. Learning structured output representation using deep conditional generative models. In Advances in neural information processing systems, pages 3483–3491, 2015.
[25] Aditya Grover, Aaron Zweig, Stefano Ermon, Graphite: Iterative generative modeling of graphs, in: International conference on machine learning, PMLR, 2019, pp. 2434–2444.
[26] Jiaxuan You, Rex Ying, Xiang Ren, William L. Hamilton, and Jure Leskovec. Graphrnn: Generating realistic graphs with deep auto-regressive models. In International Conference on Machine Learning, 2018.
[27] Renjie Liao, Yujia Li, Yang Song, Shenlong Wang, Will Hamilton, David K Duvenaud, Raquel Urtasun, and Richard Zemel. Efficient graph generation with graph recurrent attention networks. In Advances in Neural Information Processing Systems, pages 4257–4267, 2019.
[28] Wenzheng Feng, Jie Zhang, Yuxiao Dong Yu, Huanbo Luan Han, Qian Xu, Qiang Yang, Evgeny Kharlamov, Jie Tang, Graph random neural networks for semi-supervised learning on graphs, Advances in Neural Information Processing Systems 33 (2020) 22092–22103.
[29] Andre Martins and Ramon Astudillo. From softmax to sparsemax: A sparse model of attention and multi-label classification. In International Conference on Machine Learning, pages 1614–1623, 2016.
[30] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In International Conference on Learning Representations, 2019.
[31] David I. Shuman, Sunil K. Narang, Pascal Frossard, Antonio Ortega, Pierre Vandergheynst, The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains, IEEE Signal Process. Mag. 30 (3) (2013) 83–98.
[32] Vassilis Kalofolias and Nathanaël Perraudin. Large scale graph learning from smooth signals. arXiv preprint arXiv:1710.05654, 2017.
[33] Feiping Nie, Xiaoqian Wang, and Heng Huang. Clustering and projected clustering with adaptive neighbors. In Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining, pages 977–986, 2014.
[34] Cătălina Cangea, Petar Veličković, Nikola Jovanović, Thomas Kipf, and Pietro Liò. Towards sparse hierarchical graph classifiers. arXiv preprint arXiv:1811.01287, 2018.
[35] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. In Proceedings of the AAAI conference on artificial intelligence, volume 32, 2018.
[36] Yao Ma, Suhang Wang, Charu C Aggarwal, and Jiliang Tang. Graph convolutional networks with eigenpooling. In Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pages 723–731, 2019.
[37] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, Kilian Weinberger, Simplifying graph convolutional networks, in: International conference on machine learning, PMLR, 2019, pp. 6861–6871.

[38] Qu. Meng, Yoshua Bengio, Jian Tang, Gmnn: Graph markov neural networks, in: International conference on machine learning, PMLR, 2019, pp. 5241–5250.
[39] Johannes Klicpera, Stefan Weißenberger, and Stephan Günnemann. Diffusion improves graph learning. In Proceedings of the 33rd International Conference on Neural Information Processing Systems, pages 13366–13378, 2019.
[40] Cen Chen, Kenli Li, Sin G Teo, Xiaofeng Zou, Keqin Li, and Zeng Zeng. Citywide traffic flow prediction based on multiple gated spatio-temporal convolutional neural networks. ACM Transactions on Knowledge Discovery from Data (TKDD), 14(4):1–23, 20.
[41] Weng Tongfeng, Xu. Zhou, Kenli Li, Peng Peng, Li Keqin, Efficient distributed approaches to core maintenance on large dynamic graphs, IEEE Transactions on Parallel and Distributed Systems 33 (1) (2021) 129–143.



Xiaofeng Zou is currently working toward the PhD degree in computer science and technology, Hunan University, China. His research interests include data mining, machine learning and deep learning.

Kenli Li received the PhD degree in computer science from Huazhong University of Science and Technology, China, in 2003. His major research areas include high performance computing, parallel computing, grid and cloud computing. He has published more than 200 research papers in international conferences and journals such as IEEE-TKDE, IEEE-TC, IEEE-TPDS, and DAC.

Cen Chen received his PhD degree in Computer Science, Hunan University, China. Currently, he works as a Scientist II in Institute for Infocomm Research (I2R), A*STAR, Singapore. His research interest includes deep learning, achine learning, and parallel and distributed computing. He has published several research articles in international conference and journals of machine learning algorithms and parallel computing, such as IEEE-TC, IEEETPDS, IEEE-TCYB, ACM-TKDD, AAAI, ICDM, ICPP, and many more.

Xulei Yang works as a Senior Research Scientist in Institute for Infocomm Research (I2R), *STAR, Singapore. His current research interests focus on deep learning for recommendation and image/signal analysis. He is the Principal Investigator for various projects involved in providing deep learning solutions for computer vision and healthcare. He has published more than 60 scientific papers and international patents.

Wei Wei is an associate professor in the School of Computer Science and Engineering at the Xi'an University of Technology in China. He has received his Ph.D from the Xi'an Jiaotong University in 2010. His research interests include wireless networks, sensor networks, image processing, Internet of Things. He has over 100 papers published or accepted by international conferences and journals.

Keqin Li is a SUNY Distinguished Professor of computer science. He is also a Distinguished Professor at Hunan University, China. His current research interests include cloud computing, fog computing and mobile edge computing, energy-efficient computing and communication, big data computing, high-performance computing, CPU-GPU hybrid and cooperative computing, computer architectures and systems, machine learning, intelligent and soft computing.
He has authored or coauthored more than 770 journal articles, book chapters, and refereed conference papers, and has received several best paper awards. He is currently an associate editor of the ACM Computing Surveys and the CCF Transactions on High-Performance Computing.
He has served on the editorial boards of the IEEE-TPDS, IEEE-TC, IEEE-TCC, and IEEE-TSC. He is an IEEE Fellow.