Training Sparse Graph Neural Networks via Pruning and Sprouting

Xueqi Ma* Xingjun Ma† Sarah Erfani* James Bailey*

Abstract

With the emergence of large-scale graphs and deeper graph neural networks (GNNs), sparsifying GNNs including graph connections and model parameters has attracted a lot of attention. However, most existing GNN sparsification methods apply traditional neural network pruning techniques to sparsify graphs in an iterative cycle (train-then-sparsify), which not only incurs high training costs but also limits model performance. In this paper, we propose a novel *Pruning* and *Sprouting* framework for GNN (PSGNN) that not only enhances the efficiency of inference, but also boosts the performance of GNN trained on a core subgraph beyond the original graph. Based on during-training pruning, our framework gradually sparsifies the graph connections and model weights simultaneously. More specifically, PSGNN removes edges in the original graph according to the predicted label similarity between nodes from a global view. Additionally, with our graph sprouting strategy, PSGNN can generate new edges to include important yet missing topological and feature information in the original graph, while maintaining the sparsity of the graph. Extensive experiments on node classification task across different GNN architectures and graph datasets demonstrate that our proposed PSGNN method improves the performance over existing methods while saving training and inference costs.

keywords: Graph neural networks, sparse training

1 Introduction.

Graph neural networks (GNNs) [1, 2] have shown superior performance on many graph-related tasks. However, GNNs tend to suffer from severe efficiency issues in both training and inference when trained on large-scale graphs or when the network is complex. As the analyzed graphs grow rapidly in size, it is imperative to develop more effective sparsification techniques for efficient graph representation learning.

Motivated by the success of neural network pruning [3], several works [4, 5, 6, 7] have attempted to apply network pruning techniques to GNNs. As the computational cost of GNNs comes mainly from the information propagation among edges and the huge number of model parameters, these pruning methods strive to find a subgraph from the original graph and a subnetwork from the GNN that can best maintain the GNN's final performance. However, they generally follow the lottery ticket hypothesis (LTH) [3] to sparsify the graph and GNN in an iterative manner, which leads to significant training costs. To reduce the computational cost of LTH-based GNN sparsification methods, a gradual pruning method [8] was recently proposed to sparse GNNs during training. While showing both training and inference acceleration benefits, it is still a naive application of neural network pruning techniques to graphs and pays less attention to improving the performance of sparse graphs.

For GNN sparsification, it has been found that the final performance is more sensitive to graph structure sparsification than model parameter sparsification [8]. Thus, finding accurate and important graph connections is of great significance for effective sparsification. The trainable mask-based graph pruning approach adopted by most existing methods cannot fully capture the importance of the edges to the entire graph, since it neglects the global view. On the other hand, information loss inevitably occurs in real-world graphs, e.g., missing edges between two strongly associated nodes. As such, conventional pruning that only reduces edges from the original graphs often fails to obtain a superior subgraph.

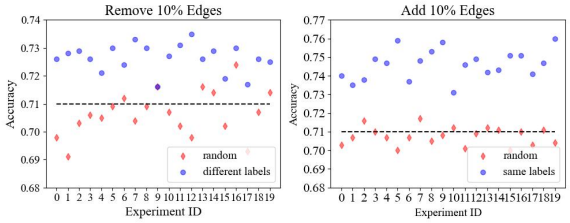

Figure 1: The performances of a two-layer GCN trained on different versions of the Citeseer dataset. *Left*: We randomly remove 10% edges or edges between two nodes with different labels in the original graph and repeat 20 times for each case. *Right*: We randomly add 10% edges or edges between two nodes with the same label. The black dashed line marks the performance obtained on the original graph, while the improved results are all above the line.

*The University of Melbourne, {xueqim@student., sarah.erfani@, baileyj@}unimelb.edu.au.

†Fudan University, xingjunma@fudan.edu.cn.

In this paper, we aim to achieve efficient inference while saving training cost in GNNs based on during-training pruning (sparse training) [9] and obtain a core sparse subgraph beyond the original graph that can lead to even better performance. Ideally, for GNNs, the message is propagated among the nodes that share the same labels. From this point of view, the edges between two nodes with different labels can be pruned, while those between the nodes that have the same labels can be added. In Fig. 1, we show how different edge modification strategies to the original graph of Citeseer [1] can impact the final performance of a GCN model. As shown in the figure, the GCN can achieve an even better performance if removing edges between nodes with different labels or adding edges between nodes with the same labels. As a comparison, randomly removing or adding edges does not have such a strong impact.

The above observation motivates us to consider the node similarity by the representation from feature and label propagation to help prune (remove) as well as sprout (generate) edges in the graphs to achieve both target sparsification and improved performance. Different from the conventional trainable mask-based pruning, our label propagation based method has two advantages: (1) it enjoys the theoretical foundation of label smoothing. Clearly, the intuition behind GNN is feature smoothing, while that behind label propagation is label smoothing. As demonstrated in [10], if the weights of edges in a graph smooth the node features with high precision, they also smooth the node labels with guaranteed upper bound on the smoothing error; (2) it considers the global structure information of the graph via the propagated labels.

While pruning edges from the original graph can be easily done, sprouting (generating) new edges can be more challenging in achieving efficient edge generation during training without extra costs. Besides predicted node similarity, we propose to first consider candidate edges, specifically, a hypergraph is constructed to encode the high-order and similarity correlations of the nodes to obtain a *candidate edge set*. Based on the candidate edge set, we can then apply the label propagation based sprouting method to generate the new edges. By pruning and sprouting the original graph, we can obtain a core sparse subgraph. Correspondingly, a magnitude-based gradual pruning approach and momentum-based sprouting strategy are applied to obtain a sparse GNN network. Experimental results show that our proposed PSGNN can improve the performance of GNNs on node classification with different GNN architectures and graph datasets in a sparse manner while achieving more efficient training and inference than LTH-based GNN sparsification methods. To summarize, our main contributions are:

- We propose a novel Pruning and Sprouting (PSGNN) framework for GNN sparsification. PSGNN not only prunes existing edges but also sprouts new edges in the graph to obtain a core sparse graph that can lead to an even better performance beyond the original graph.
- We propose to use feature and label propagation to obtain feature and label information and then prune or sprout the edges based on the similarity between its two associated nodes. This presents a more graph-oriented strategy for GNN sparsification.
- We conduct extensive experiments on both small- and large-scale graph datasets to show the effectiveness of our proposed PSGNN on node classification tasks. PSGNN improves upon GCN by a margin of 2.4% and 1.4% on Citeseer and Pubmed, and a margin of over 30% on heterophilic datasets. Compared to standard LTH-based methods UGS, PSGNN saves up to 45× training time.

2 Related Work.

2.1 Graph Neural Networks. Graph neural networks have been developed as powerful models for graph-related tasks [1, 11]. The popular GNN models are developed to update the node embedding with messages from its neighboring nodes. As shallow GNNs have limited expressive power on large graphs, deeper GNN architectures are designed to achieve more powerful graph representation. For example, ResGCN [12] introduced residual connections and dilated convolutions to build a 56-layer GCN model. However, deep GNN models applied on large-scale graphs usually suffer from high computational costs due to the increased scale of the graph as well as the model parameters.

2.2 GNN Sparsification. The lottery ticket hypothesis (LTH) [3] indicates that a dense randomly-initialized neural network contains a sparse subnetwork that—when trained in isolation—can achieve comparable test accuracy to the original network with the same number of iterations. Based on this hypothesis, Chen et al. [4] proposed a unified GNN sparsification framework (UGS) to obtain a sparse graph and a sparse GNN, showing the existence of graph lottery tickets. Following this, several works [5, 6, 7] based on LTH were proposed to identify tickets. However, these LTH-based methods often need to train the dense models fully and iteratively for up to 20 times, to obtain a sparse graph and GNN, significantly increasing the computational

cost. To boost both training and inference efficiency, Liu et al. [8] proposed a novel gradual pruning framework CGP to prune the graph structure and network parameters simultaneously during training. However, CGP also directly applied the traditional neural network pruning technique to graphs without considering the unique properties and challenges of graph structure sparsifying. All these sparsification methods pay less attention to improving the performance of the sparse graph and GNN. In this paper, we develop a graph-oriented sparsification technique to achieve sparsification and performance boosts at the same time.

3 Methodology.

Overview. Our proposed GNN Pruning and Sprouting (PSGNN) framework is illustrated in Fig. 2. Via gradual pruning and sprouting applied on both the graph structure and model weights, we obtain a core sparse subgraph and subnetwork to 1) speed up model inference, and 2) improve the final performance of GNNs in a sparse manner. In PSGNN, we pay more attention to graph structure sparsification which has been proven to be more critical for graph learning. Specifically, we remove the unimportant edges of the original graph structure using a label propagation based pruning method. Meanwhile, considering the given graph may have missing but important edges, we generate new edges via a label propagation based sprouting method while maintaining the target sparsity. The pruning and sprouting steps are gradually and iteratively applied during training to obtain an optimal subgraph. For model weight sparsification, we simply use the traditional magnitude based strategy for pruning while introducing a momentum-based sprouting strategy to help recover the pruned yet critical connections to refine the pruning process. Next, we will introduce the preliminary definitions, the two important components of PSGNN: graph structure sparsification and model weight sparsification, and the overall sparsification process.

3.1 Preliminaries.

Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with N nodes in $\mathcal{V}$ and edges $(v_i, v_j) \in \mathcal{E}$. The graph topology can be described by the adjacency matrix $\mathbf{A}$, where $\mathbf{A}_{ij} = 1$ if there exists an edge (v_i, v_j), $\mathbf{A}_{ij} = 0$ otherwise. The feature space $\mathbf{X} \in \mathbb{R}^{N \times d}$ with each node $i \in \mathcal{V}$ has a d-dimensional feature vector. For a semi-supervised node classification task with a subset of node labels $\mathbf{Y}_L \in \mathbb{R}^{m \times C}$ with m and C representing the number of labeled samples and classes, the objective is to learn an embedding function f that can predict the labels of the remaining nodes.

Modern GNNs usually follow the message-passing scheme, where node representation is iteratively updated by aggregating representations of its neighbors. Taking GCN as an example, a two-layer GCN model can be formulated as:

$$\boldsymbol{H} = \text{softmax}\left(\hat{\mathbf{A}}\,\text{ReLU}(\hat{\mathbf{A}}\mathbf{X}\mathbf{W}^{(0)})\mathbf{W}^{(1)}\right), \tag{3.1}$$

where, $\boldsymbol{H}$ is the GCN's output predictions, $\hat{\mathbf{A}} = \hat{\mathbf{D}}^{-\frac{1}{2}}(\mathbf{A} + \mathbf{I}_n)\hat{\mathbf{D}}^{-\frac{1}{2}}$ is the normalized $\mathbf{A} + \mathbf{I}_n$ matrix by the degree matrix $\hat{\mathbf{D}}$, and $\mathbf{W} = (\mathbf{W}^{(0)}, \mathbf{W}^{(1)})$ are the weights of the two-layer GCN model. For semi-supervised learning, the objective function $\mathcal{L}$ is:

$$\mathcal{L} = -\frac{1}{\mathcal{Y}_L}\sum_{v_i \in \mathcal{Y}_L} \boldsymbol{y}_i \log(\boldsymbol{h}_i), \tag{3.2}$$

where $\mathcal{Y}_L$ is the label set, and $\boldsymbol{y}_i$ and $\boldsymbol{h}_i$ are the label and prediction of node v_i.

3.2 Graph Structure Sparsification.

For graph learning in which the message propagates and aggregates along the edges, the ideal scenario is that the edge connects two nodes with the same label. With this in mind, we introduce our label propagation based pruning and sprouting techniques as follows.

3.2.1 Label Propagation.

For semi-supervised learning, we assume the initial label matrix $\mathbf{Y}^{l(0)} = [y_1^{l(0)}, y_2^{l(0)}, \cdots, y_N^{l(0)}]$ consists of one-hot label indicator vectors for labeled nodes $i = 1, \cdots, m$ while zero vectors for unlabeled nodes. Propagating the labels with the normalized adjacency $\mathbf{D}^{-1}\mathbf{A}$, the k^{th} iteration of label propagation is formulated as:

$$\mathbf{Y}^{l(k)} = \mathbf{D}^{-1}\mathbf{A}\mathbf{Y}^{l(k-1)}. \tag{3.3}$$

It then resets the labeled samples to their initial labels via $y_i^{l(k)} = y_i^{l(0)}, \forall i \leq m$. This is to maintain the label information of the labeled nodes so that the unlabeled nodes do not overpower the labeled ones, as the initial labels would otherwise fade away.

After K-order label propagation, we can obtain the label matrix $\mathbf{Y}$, and get the labels of nodes as following:

$$\mathbf{Z} = \text{softmax}(\mathbf{Y} + \alpha\boldsymbol{H}), \tag{3.4}$$

where, $\mathbf{Y}$ is the representation obtained from label propagation, $\boldsymbol{H}$ is the prediction from feature propagation, $\mathbf{Z} \in \mathbb{R}^{N \times C}$, and α is a learnable attention parameter. The above equation combines label propagation and feature propagation with the attention parameter α balancing the two. In GNN sparsification, the prediction $\boldsymbol{H}$ is obtained with the sparse GNN. After obtaining

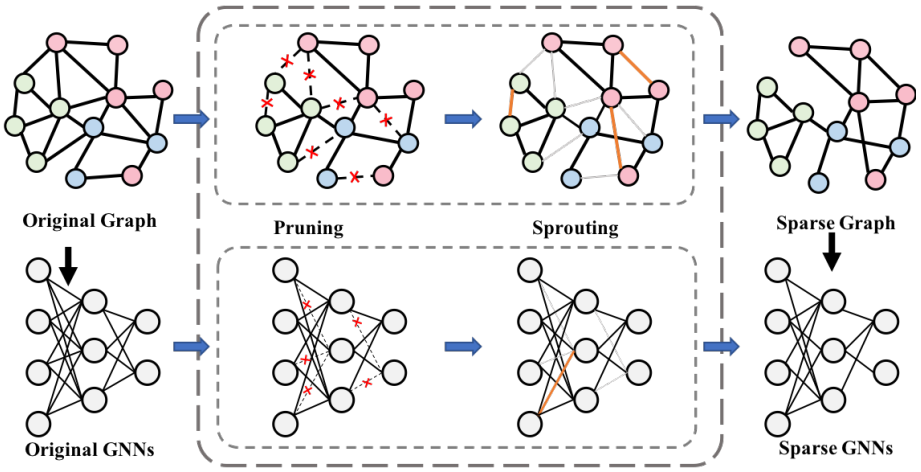

Figure 2: The GNN Pruning and Sprouting (PSGNN) framework. During the training process of GNNs, PSGNN gradually prunes and sprouts the graph structure and model weights to obtain a sparse graph and sparse GNNs.

predicted outputs, the supervised objective of the graph node classification task is defined as

$$\mathcal{L} = -\frac{1}{\mathcal{Y}_L} \sum_{v_i \in \mathcal{Y}_L} \boldsymbol{y}_i \log(\boldsymbol{z}_i), \tag{3.5}$$

where $\mathcal{Y}_L$ is the label set, and $\boldsymbol{y}_i$ and $\boldsymbol{z}_i$ are the label prediction of node v_i.

3.2.2 Graph Pruning. After obtaining the label prediction of all nodes, we use the similarity between the two nodes to assess the importance of the existing edge between them. Assuming there is an edge between nodes v_i and v_j, we compute the edge score $s_{ij} = ||\mathbf{Z}_i - \mathbf{Z}_j||_2$. As a smaller score s_{ij} means nodes v_i and v_j have a stronger relationship, we thus sort these edges in an ascending order to obtain a sorted edge set $\mathcal{E}^r$, where the front edges are the critical connections that we want to preserve.

We next prune the graph structure via an adjacency mask $\boldsymbol{m}_a \in \mathbb{R}^M$ which is gradually sparsified during the pruning process. Note that $M > \|\mathbf{A}\|_0$ ($\|\mathbf{A}\|_0$ is the total number of edges in the original graph), because of the sprouting process afterward. The elements of mask $\boldsymbol{m}_a$ are initialized to 1 if there exists an edge in the original graph and 0 otherwise. Based on the candidate removing edge list $\mathcal{E}^r$, the graph structure is updated by:

$$\begin{aligned} \boldsymbol{m}_a &= \text{select-p}\left(\boldsymbol{m}_a, \text{TopK}\left(\{\mathcal{E}^r\}, \lceil (1-p_a)\|\mathbf{A}\|_0 \rceil\right)\right) \\ \mathbf{A} &= \boldsymbol{m}_a \odot \mathbf{A}, \end{aligned} \tag{3.6}$$

where p_a is the sparsity level, $p_a\|\mathbf{A}\|_0$ is the number of pruned edges, $\text{TopK}(v, k)$ returns the top k edge indexes from $\mathcal{E}^r$, $\lceil \cdot \rceil$ is the rounding up operation. The function select-p(v, k) updates the mask by keeping the returned edge indexes unchanged and setting all others to 0. The adjacency relationship used in the next training iteration will be updated by dot multiplying the new mask.

3.2.3 Graph Sprouting. To obtain an optimal subgraph, it is important to explore candidate important yet missing edges from the original graph structure. However, generating new edges can be extremely challenging for two reasons. First, as the adjacency matrix is often sparse, there are an enormous number of candidate edges to consider, which brings large memory and computational costs. Second, it is difficult to evaluate the importance of a non-existing edge relative to existing edges. To tackle these challenges, we first define a candidate edge set of high potential connections to restrict the search space, and then propose a label propagation based and computationally cheap sprouting method.

Intuitively, the missing important edges can be discovered from the nodes that are close in the feature space or the topological space. Viewing node v_i as a center, its neighbors in the feature space can be connected by an edge (Fig. 3(a)). Under the assumption that two vertices that are close in topology are more likely to share the same label, we also need to consider the higher-order correlations based on the original graph structure (Fig. 3(b)).

Hypergraphs [13] have been widely adopted to present the high-order relationship among the nodes via hyperedges. Here, we introduce hypergraphs to encode the high-order relationship missing in the origi-

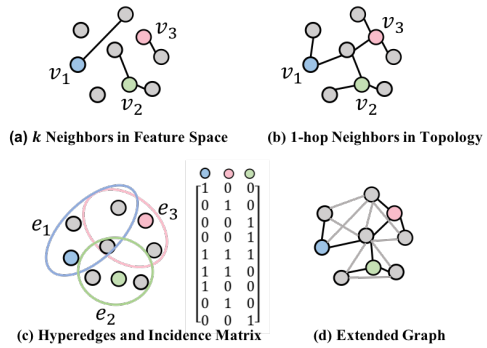

Figure 3: Candidate edges from feature and topology space. Viewing each node as a centroid, (A) finding its k neighbors in the feature space; (b) finding its 1-hop neighbors in the topology space; (c) generating a hyperedge for each node with its neighbors in the feature and topology space; (d) obtaining the candidate edges (grey lines) based on clique expansion of the hypergraph.

nal graph structure. Viewing each node as a centroid, we generate a hyperedge for each node with its 1-hop neighbors in the original graph structure and its k-nn neighbors in the feature space (see in Fig. 3(c)). Consequently, the generated hyperedges contain structural information and feature similarity. The correlations of nodes and hyperedges can be represented in a $|V| \times |E|$ incidence matrix $\mathbf{H}$ with entries $h(v,e) = \begin{cases} 1, & \text{if } v \in e \\ 0, & \text{if } v \notin e. \end{cases}$. Then, we get the extended correlations: $\mathbf{\Theta} = \mathbf{D}_v^{-1/2}\mathbf{H}\mathbf{D}_e^{-1}\mathbf{H}^\top\mathbf{D}_v^{-1/2}$, with the diagonal matrices $\mathbf{D}_v$ and $\mathbf{D}_e$ denote the vertex and hyperedge degrees, respectively. All edges are the clique expansion based on the constructed hypergraph. We can now obtain the candidate edges that are not in the original graph by selecting strong-link edges from $\mathbf{\Theta}$ with respect to a threshold θ.

Let r denote the ratio of the number of regenerated connections to the total number of connections. To achieve and maintain the target graph sparsity, we first remove r proportion of edges by:

$$\boldsymbol{m}_a = \text{select-p}\left(\boldsymbol{m}_a, \text{TopK}\left(\{\mathcal{E}^r\}, \lceil (1 - s_a - r)\|\mathbf{A}\|_0 \rceil\right)\right), \tag{3.7}$$

where s_a is the current sparsity. Immediately after this, we generate r proportion of new connections. We first calculate the similarity scores of the candidate edges and all removed edges from the original graph, i.e., the edges with zero values in mask $\boldsymbol{m}_a$. Then, similar to the sorting strategy of the pruning process, we sort edges in ascending order into a sorted edge set $\mathcal{E}^a$. The front edges in $\mathcal{E}^a$ are edges we want to sprout. After obtaining the sprouting edge list, we update the mask by:

$$\begin{aligned} \boldsymbol{m}_a &= \text{select-s}\left(\boldsymbol{m}_a, \text{TopK}\left(\{\mathcal{E}^a\}, \lceil r\|\mathbf{A}\|_0 \rceil\right)\right) \\ \mathbf{A} &= \boldsymbol{m}_a \odot \mathbf{A}, \end{aligned} \tag{3.8}$$

where, function select-s(v,k) updates the mask by setting the returned indexes from TopK(v,k) to 1. The adjacency matrix is then updated by mask $\boldsymbol{m}_a$.

3.3 Model Weight Sparsification. Model weight sparsification also consists of two steps: weight pruning and sprouting. A conventional magnitude-based weight pruning strategy is adopted for weight pruning, while for weight sprouting, a momentum-based strategy is used.

3.3.1 Weight Pruning. Arguably, weight pruning is the most typical approach for model simplification. Here, we prune a certain proportion of the weights that have the smallest magnitude, a standard pruning strategy for neural network pruning. Specifically, we create a binary mask $\boldsymbol{m}_w$ that is of the same size as the model weights $\mathbf{W}$ and initialize its elements to one at the time of model initialization. During the pruning process, the mask matrix is updated by removing p_w proportion of the weights that have the smallest magnitude as follows:

$$\begin{aligned} \boldsymbol{m}_w &= \text{select-p}\left(\boldsymbol{m}_w, \text{TopK}\left(|\mathbf{W}|, \lceil (1 - p_w)\|\mathbf{W}\|_0 \rceil\right)\right) \\ \mathbf{W} &= \boldsymbol{m}_w \odot \mathbf{W}, \end{aligned} \tag{3.9}$$

where $\|\mathbf{W}\|_0$ is the total number of the weights, TopK(v,k) returns the top k elements. The select-p function keeps the values in $\boldsymbol{m}_w$ with indices from TopK and sets all others to 0. The model weights are updated by dot multiplying with mask $\boldsymbol{m}_w$. Note that we adopt a global pruning strategy, that is, pruning the weights of different layers together.

3.3.2 Weight Sprouting. During the weight pruning progress, especially in the early iterations, premature pruning may occur. To correct the "mistaken" pruning, we also introduce a sprouting scheme for model weight. Specifically, we adopt the momentum-based [14] strategy for sprouting the model weights, specifically, leveraging exponentially smoothed gradients (momentum) at different layers to distribute the pruning budget and also grow weights according to the momentum magnitude of zero-valued weights. Whilst sparse momentum based sprouting works reasonably well in our experiments, it can potentially be replaced by other strategies such as random sprouting or gradient-based sprouting. In order to maintain the network sparsity before and after sprouting, we first remove r proportion of the elements in the weight mask $\boldsymbol{m}_w$ and then perform

sprouting. This process can be formulated as follows:

(3.10)
$$\boldsymbol{m}_w = \text{select-p}\left(\boldsymbol{m}_w, \text{TopK}\left(|\mathbf{W}|, \lceil (1 - s_w - r)\|\mathbf{W}\|_0 \rceil\right)\right)$$
$$\boldsymbol{m}_w = \text{select-s}\left(\boldsymbol{m}_w, \text{TopK}\left(\|\boldsymbol{M}_{i \notin \mathbf{W}}\|, \lceil (r)\|\mathbf{W}\|_0 \rceil\right)\right)$$
$$\mathbf{W} = \boldsymbol{m}_w \odot \mathbf{W},$$

where $\boldsymbol{M}_{i \notin \mathbf{W}}$ are the momentum magnitudes of the zero weights. The last step in the above updates the model weights by the mask.

3.4 Overall Sparse Training Procedure. We now present the sparse training procedure of our proposed PSGNN framework. The detailed procedure is described in Algorithm 1. Given a graph $\mathcal{G} = (\mathbf{A}, \mathbf{X}, \mathbf{Y}^{l(0)})$, GNN model $f(\mathcal{G}, \mathbf{W})$, masks $\boldsymbol{m_a}$ and $\boldsymbol{m_w}$, we **gradually** sparsify [15] the dense GNN to the target sparsity level over n pruning iterations under the objective $f(\boldsymbol{m}_a \odot \mathbf{A}, \mathbf{X}, \boldsymbol{m}_w \odot \mathbf{W})$. Let us define s_{a_i} and s_{w_i} as the initial graph sparsity and weight sparsity, s_{a_f} and s_{w_f} are the target graph sparsity and weight sparsity, t_0 is the starting epoch of gradual pruning, t_f is the ending epoch of gradual pruning, and Δt is the pruning frequency. The pruning rates for graph s_{a_t} and weights s_{w_t} in each pruning iteration are dynamically set to be as follows:

$$\begin{aligned} s_{a_t} &= s_{a_f} + \left(s_{a_i} - s_{a_f}\right)\left(1 - \frac{t - t_0}{n\Delta t}\right)^3, \\ s_{w_t} &= s_{w_f} + \left(s_{w_i} - s_{w_f}\right)\left(1 - \frac{t - t_0}{n\Delta t}\right)^3, \end{aligned} \tag{3.11}$$

where, $t \in \{t_0, t_0 + \Delta t, \ldots, t_0 + n\Delta t\}$. This sparsity function can prune the networks rapidly in the initial phase when the redundant connections are abundant and gradually reduce the number of weights or edges being pruned each time as there are fewer and fewer remaining in the networks. During the training stage, at every Δt training step, the pruning rate of edges and weights are calculated by Eq. (3.11), and the specific pruning operations are presented in Eq. (3.6) and Eq. (3.9). After the pruning operation, we apply the sprouting operation to generate edges by Eq. (3.7) and Eq. (3.8), and "correct weight" by Eq. (3.10) before the next round's training.

Algorithm 1 GNN Pruning and Sprouting (PSGNN)

Require: Graph $\mathcal{G} = (\mathbf{A}, \mathbf{X}, \mathbf{Y}^{l(0)})$, GNN $f = (\mathcal{G}, \mathbf{W})$, trainable parameter α, initial masks $\boldsymbol{m}_a$, $\boldsymbol{m}_w$, target sparsity s_{a_f} and s_{w_f}, gradual pruning starting point t_0, gradual pruning endpoint t_f, gradual pruning frequency Δt.

Ensure: $f(\boldsymbol{m}_a \odot \mathbf{A}, \mathbf{X}, \boldsymbol{m}_w \odot \mathbf{W})$

for each training step t **do**
  Forward $f(\{\boldsymbol{m}_a \odot \mathbf{A}, \mathbf{X}\}, \boldsymbol{m}_w \odot \mathbf{W})$
  Label prediction by Eq. (3.3), Eq. (3.4)
  Backpropagate to update $\mathbf{W}$ and α
  if $t_0 \le t \le t_f$ and $(t \bmod \Delta t) == 0$ **then**
    Pruning $\mathbf{A}$ by Eq. (3.6) with dynamic pruning rate produced by Eq. (3.11)
    Pruning $\mathbf{W}$ by Eq. (3.9) with dynamic pruning rate produced by Eq. (3.11)
    Sprouting $\mathbf{A}$ by Eq. (3.7) and Eq. (3.8)
    Sprouting $\mathbf{W}$ by Eq. (3.10)
  end if
end for

4 Experiments.

We conduct extensive experiments to evaluate the effectiveness of our PSGNN framework for node classification on diverse graph datasets and GNN models.

4.1 Experimental Setting. We consider 10 graph datasets of different scales including small-scale graphs and large-scale graphs from Open Graph Benchmark (OGB) [16]. These datasets cover both homophilic graphs and heterophilic graphs. We use their original train-val-test splits for our experiments.

Classification Baseline Methods. We consider three representative baseline models including GCN [1], GAT [2], and APPNP [17]. For large-scale datasets, we consider deeper GNNs, i.e., 28-layer deep ResGCNs [12]. While our focus is to improve the performance of existing GNNs by sparsifying the graph structure and model weight, we also compare the performance of our PSGNN method with state-of-the-art standard (non-sparse) GNN training methods, including DropEdge [18] and GPRGNN [19], to show its effectiveness.

Sparsification Baseline Methods. We compare our method with 4 state-of-the-art GNN sparsification methods including UGS [4], GEBT [5], ICPG [6], and one during-training pruning based GNN sparsification method, Comprehensive Graph Pruning (CGP) [8]. CGP is the closest method to our PSGNN.

Our method needs to generate the candidate edges in the sprouting phase of graph structure sparsification. For homophilic datasets, we construct the hypergraph by considering the high-order connections in topology and feature spaces. For heterophilic datasets, the hypergraphs for these heterophilic datasets are constructed based on the nearest k neighbors in the feature space. In label propagation, we control the number of propagation layers K in $\{8, 10, 12\}$. In the sprouting, we construct a hypergraph, and introduce hyper-parameter k with $k$nn algorithm. We choose k from $\{3, 5, 8\}$ and set θ with 0.15 to restrict the candidate edge set. The other

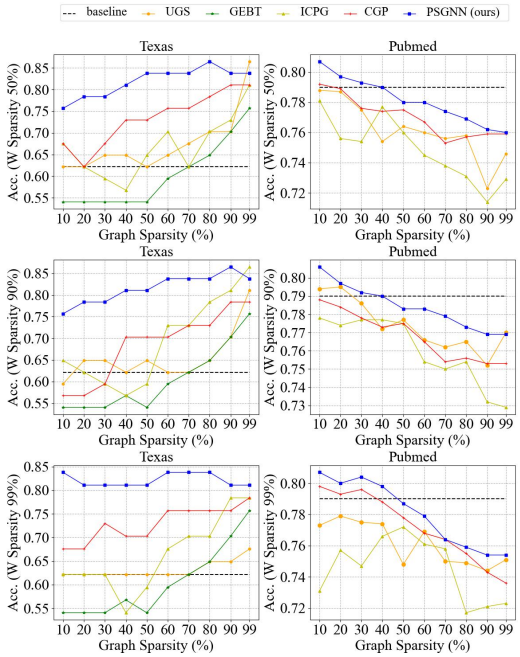

Figure 4: Comparison with SOTA GNN sparsification methods on different datasets at different weight sparsities and graph sparsities. The baseline model is a standard GCN. The dashed lines mark the baseline (unsparsified) performance, above which are improved results. (GEBT OOM on Pubmed.)

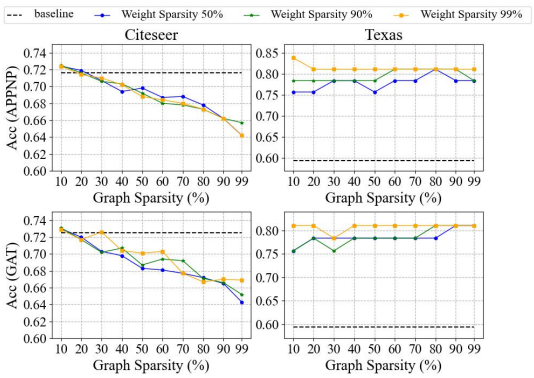

Figure 5: Performance of PSGNN at different weight sparsities (curves) and graph sparsities (the x-axis) on different datasets (columns), with different baseline models APPNP (top row) and GAT (bottom row).

experimental settings are similar to CGP[8].

4.2 Main Results. We conduct experiments to evaluate the effectiveness of our PSGNN in node classification and compare its performance with the baseline models and GNN sparsification methods. Based on the results, we summarize the key observations as follows.

Obs.1. PSGNN can improve the performance of the baseline GNNs (in a sparse manner). As shown in Fig. 4, PSGNN can improve the performance of the original GCN model (the dashed lines) significantly on different datasets, with a sparse graph and sparse network. On Pubmed, at 90% weight sparsity and 10% graph sparsity, PSGNN improves upon GCN by a margin of 1.7%. PSGNN can achieve comparable performance to the dense baseline with 40% graph sparsity on Pubmed. The superiority of PSGNN can also be observed on other GNN models including GAT and APPNP in Fig. 5.

On heterophilic datasets, i.e., Texas, PSGNN can improve the performance consistently at different sparsity levels. Particularly, as shown in Table 1, PSGNN brings up to 30% improvement in classification accuracy compared to the corresponding baseline GCN, APPNP, and GAT. In a low homophilic graph, the connected nodes may have different class labels and dissimilar features with high probability. Therefore, most edges should be pruned while a lot of missing edges should be added. By pruning and sprouting, PSGNN can quickly correct the graph connections and achieve better performance at a low graph sparsity. This indicates the importance of pruning, and more importantly, sprouting in graph sparsification.

Obs.2. PSGNN outperforms GNN sparsification methods. The improvement of our PSGNN over other GNN sparsification methods is shown in Fig. 4, where it surpasses the baseline sparsification methods in most cases across different datasets. Specifically, on Pubmed, the baseline sparsification methods perform poorly without comprehensively considering significant graph connections. On Texas, the advantage of PSGNN over existing methods is more pronounced for sparsity less than 90%.

Obs.3. PSGNN achieves comparable or even better performance than SOTA dense GNN models. As can be observed in Table 1, PSGNN achieves comparable or higher accuracy than some standard GNN models (dense) consistently across different datasets. DropEdge [18] can be viewed as a graph sparsification method as it removes edges from the original graph structure. The GNN models trained by PSGNN are noticeably better than those trained by DropEdge on homophilic datasets as shown in Table 1 (except Cora which is sensitive to pruning [4]). This is also the case on heterophilic datasets. PSGNN achieves comparable or even better performance than SOTA GNN models (H2GCN [21], GPRGNN [19], and FAGCN [22]) which

Table 1: Classification accuracy (%) on different datasets. The results of PSGNN are reported over 10 runs. For heterophilic datasets, the results of PSGNN are reported with over 80% graph sparsity.

Methods	Cora	Citeseer	Pubmed	Cornell	Texas	Wisconsin	Actor
MLP	54.9±0.5	53.7±0.6	70.2±0.5	71.6±5.6	78.0±5.2	82.2±6.9	33.3±0.9
DropEdge [18]	82.8	72.3	79.6	-	-	-	-
SGC [20]	81.0±0.0	71.9±0.1	78.9±0.0	43.0±5.5	60.3±5.1	53.1±4.8	27.0±1.4
H_2GCN [21]	75.3±0.6	67.9±0.7	76.0±0.7	75.4±4.1	79.7±3.3	77.6±4.1	36.2±0.5
GPRGNN [19]	83.5±0.7	71.4±0.8	79.2±0.7	76.7±2.2	81.1±4.4	82.7±5.6	35.3±0.8
FAGCN [22]	83.3±0.4	72.6±0.6	79.4±0.2	67.6±5.26	75.7±4.7	75.3±3.1	32.1±1.3
GCN [1]	81.5	70.3	79.0	45.7±7.9	60.8±8.0	52.6±4.3	28.1±1.2
GAT [2]	83.0±0.7	72.5±0.7	79.0±0.3	47.0±7.6	62.2±4.5	57.5±3.5	28.3±1.1
APPNP [17]	**83.8**±0.3	71.6±0.5	79.7±0.3	41.4±7.2	61.6±5.4	55.3±3.9	29.4±0.8
PSGNN (GCN)	81.4±0.3	72.7±0.3	80.4±0.3	**83.8**±0.0	**83.8**±0.0	**88.2**±0.0	36.9±0.0
PSGNN (GAT)	81.4±0.3	**73.1**±0.4	**80.9**±0.2	81.1±0.0	81.1±0.0	86.3±0.0	36.5±0.0
PSGNN (APPNP)	83.5±0.4	72.5±0.3	80.5±0.2	**83.8**±0.0	81.1±0.0	**88.2**±0.0	**38.4**±0.0

designed specifically for heterophilic datasets. This confirms the benefit of improving the graph topology with our proposed pruning and sprouting.

Obs.4. PSGNN achieves comparable or better inference efficiency than the baselines. In Fig. 6, we see that, when the inference FLOPs (floating point operations, 1 FLOPs ≈ 2 MACs) reduces (from left to right) to about only 500M, PSGNN saves up to 90% FLOPs/MACs of the baseline models while achieving a similar performance as the original graph and model. For less inference speed up, e.g., 10% FLOPs saving, the performance is considerably improved beyond the original graph and network.

Obs.5. PSGNN is more training efficient than LTH-based sparsification methods. In Fig. 7, we plot the training time of different GNN sparsification methods relative to standard training (1×) on Citeseer and Cornell datasets. For a fair comparison, we set the same target sparsification rate (90% sparsity of graph and 99% sparsity of model weights) for all the compared methods. It is evident that UGS is the least training-efficient method which takes about 100 times more than the baseline. The sparse training based methods CGP and our PSGNN demonstrate the best training efficiency, while the LTH-based methods (UGS, GEBT, and ICPG) are far less efficient. This highlights the training acceleration advantage of during-training pruning methods.

4.3 Scaling to Deeper GNNs on Large-Scale Datasets.

We conduct experiments with a deeper GNN model ResGCN on 3 large-scale datasets (ogbn-arxiv, ogbn-products, and ogbn-proteins). As shown in Fig. 8, on ogbn-arxiv and ogbn-proteins datasets, PSGNN is able to maintain the original performance of ResGCN (dashed lines) while reducing 50%, 25% of the FLOPs, respectively. The speedup is more significant on ogbn-products dataset where it improves the baseline performance by a considerable margin while saving up to 80% FLOPs. Our PSGNN has a similar training time to the dense ResGCN, while the lottery ticket hypothesis based method UGS needs nearly 50× more training time.

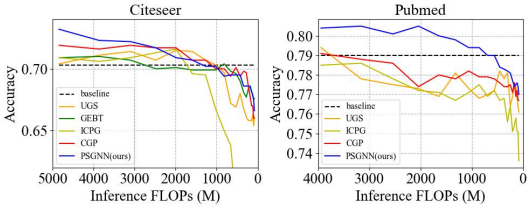

Figure 6: Comparison inference efficiency (FLOPs) of different sparsification methods based on GCN. Note that FLOPS are computed based on the same GNN sparsity level for different GNN sparsification methods.

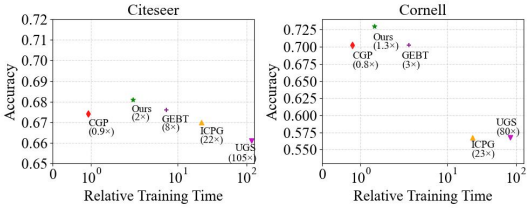

Figure 7: Relative training time (x-axis) vs. performance (y-axis) of different sparsification methods based on GCN. The training time of dense GCN is 1×.

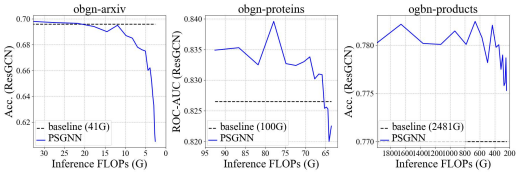

Figure 8: The performance of 28-layer ResGCNs on large-scale graph datasets. The inference FLOPs is computed based on different graphs and weight sparsity.

5 Conclusion.

In this paper, we proposed a novel GNN Pruning and Sprouting framework (PSGNN) to gradually and simultaneously sparsify the graph structure and model weights during training. Besides pruning, a sprouting process is also introduced in PSGNN to generate new and important edges and model weights to help maintain or even boost the final performance. The effectiveness of PSGNN has been verified in node classification. Our method can serve as a strong baseline for GNN sparsification. Note that our proposed method is a transductive setting and thus cannot be applied for graph classification. In future work, we aim to efficiently identify the core subgraph by considering more graph structural information.

Acknowledgements: This work is partially supported by Australian Research Council (ARC) Discovery Project DP230101534, Australian Research Council (ARC) Discovery Early Career Researcher Award (DECRA) DE220100680.

References

[1] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *ICLR*, 2017.

[2] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *ICLR*, 2018.

[3] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," in *ICLR*, 2019.

[4] T. Chen, Y. Sui, X. Chen, A. Zhang, and Z. Wang, "A unified lottery ticket hypothesis for graph neural networks," in *ICML*, 2021.

[5] H. You, Z. Lu, Z. Zhou, Y. Fu, and Y. Lin, "Early-bird gcns: Graph-network co-optimization towards more efficient GCN training and inference via drawing early-bird lottery tickets," in *AAAI*, pp. 8910–8918, 2022.

[6] Y. Sui, X. Wang, T. Chen, X. He, and T.-S. Chua, "Inductive lottery ticket learning for graph neural networks," 2021.

[7] B. Hui, D. Yan, X. Ma, and W. Ku, "Rethinking graph lottery tickets: Graph sparsity matters," in *ICLR*, 2023.

[8] C. Liu, X. Ma, Y. Zhan, L. Ding, D. Tao, B. Du, W. Hu, and D. P. Mandic, "Comprehensive graph gradual pruning for sparse training in graph neural networks," *CoRR*, vol. abs/2207.08629, 2022.

[9] T. Lin, S. U. Stich, L. Barba, D. Dmitriev, and M. Jaggi, "Dynamic model pruning with feedback," in *ICLR*, 2020.

[10] H. Wang and J. Leskovec, "Unifying graph convolutional neural networks and label propagation," *arXiv preprint arXiv:2002.06755*, 2020.

[11] M. Zhang and Y. Chen, "Link prediction based on graph neural networks," in *NeurIPS*, 2018.

[12] G. Li, M. Müller, A. K. Thabet, and B. Ghanem, "Deepgcns: Can gcns go as deep as cnns?," in *ICCV*, pp. 9266–9275, IEEE, 2019.

[13] D. Zhou, J. Huang, and B. Schölkopf, "Learning with hypergraphs: Clustering, classification, and embedding," in *NeurIPS*, 2006.

[14] T. Dettmers and L. Zettlemoyer, "Sparse networks from scratch: Faster training without losing performance," *CoRR*, vol. abs/1907.04840, 2019.

[15] M. Zhu and S. Gupta, "To prune, or not to prune: Exploring the efficacy of pruning for model compression," in *ICLR*, 2018.

[16] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec, "Open graph benchmark: Datasets for machine learning on graphs," in *NeurIPS*, 2020.

[17] J. Klicpera, A. Bojchevski, and S. Günnemann, "Predict then propagate: Graph neural networks meet personalized pagerank," in *ICLR*, 2019.

[18] Y. Rong, W. Huang, T. Xu, and J. Huang, "Dropedge: Towards deep graph convolutional networks on node classification," in *ICLR*, 2020.

[19] E. Chien, J. Peng, P. Li, and O. Milenkovic, "Adaptive universal generalized pagerank graph neural network," in *ICLR*, 2021.

[20] F. Wu, A. H. S. Jr., T. Zhang, C. Fifty, T. Yu, and K. Q. Weinberger, "Simplifying graph convolutional networks," in *ICML*, 2019.

[21] J. Zhu, Y. Yan, L. Zhao, M. Heimann, L. Akoglu, and D. Koutra, "Beyond homophily in graph neural networks: Current limitations and effective designs," in *NeurIPS*, 2020.

[22] D. Bo, X. Wang, C. Shi, and H. Shen, "Beyond low-frequency information in graph convolutional networks," in *AAAI*, pp. 3950–3957, 2021.