Boosting Graph Pooling with Persistent Homology

Zeyang Ding

Contents

Graph pooling

Graph pooling has been used in various applications, which can reduce the graph size while preserving its structural information. DiffPool,MinCutPool and DMoNPool methods are examples of dense pooling due to the space complexity they incur. Despite their effectiveness, dense pooling methods have been criticized for high memory cost and complexity. Therefore, various sparse pooling methods have been proposed, such as Top-K, ASAPool, and SAGPool. These methods coarsen graphs by selecting a subset of nodes based on a ranking score. As they drop some nodes in the pooling process, these methods are criticized for their limited capacity to retain essential information, with potential effects on the expressiveness of preceding GNN layers.

Persistent homology in GNNs

PH is a technique to calculate topological features of structured data, and many approaches have been proposed to use PH in graph machine learning due to the high expressiveness of topological features on graphs. This encourages further exploration on equipping GNNs with topological features. However, existing methods tend to view PH merely as a tool for providing supplementary information to GNNs, resulting in only marginal improvements and limited interpretability.

Contents

Graph

Let $\mathcal{G} = (V, E)$ be an undirected graph with n nodes and m edges, where V and E are the node and the edge sets, respectively. Nodes in attributed graphs are associated with features, and we denote by $V = \{(v, \mathbf{x}_v)\}_{v \in 1:n}$ the set of nodes v with d dimensional attribute $\mathbf{x}_v$. It is also practical to represent the graph with an adjacency matrix $\mathbf{A} \in \{0, 1\}^{n \times n}$ and the node feature matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$

GNN framework

The k-th layer of GNNs can be expressed as:

$$\mathbf{X}^{(k)} = \mathrm{M}\left(\mathbf{A}, \mathbf{X}^{(k-1)}; \theta^{(k)}\right) \tag{1}$$

where $\theta^{(k)}$ is the trainable parameter, and M is the message propagation function. Numbers of M have been proposed in previous research (Kipf & Welling, 2016; Hamilton et al., 2017). A complete GNN is typically instantiated by stacking multiple layers of 1. Hereafter we denote by $\mathrm{GNN}(\cdot)$ an arbitrary such multi-layer GNN for brevity.

Dense Graph Pooling

GP in GNNs is a special layer designated to produce a coarsened or sparsified sub-graph. Formally, GP can be formulated as $\mathcal{G} \mapsto \mathcal{G}_P = (V_P, E_P)$ such that the number of nodes $|V_P| \leq n$. GP layers can be placed into GNNs in a hierarchical fashion to persistently coarsen the graph. Typical GP approaches (Ying et al., 2018; Bianchi et al., 2020; Müller, 2023) rely on learning a soft cluster assignment matrix $\mathbf{S}^{(l)} \in \mathbb{R}^{n_{l-1} \times n_l}$:

$$\mathbf{S}^{(l)} = \mathrm{softmax}\left(\mathrm{GNN}^{(l)}\left(\mathbf{A}^{(l-1)}, \mathbf{X}^{(l-1)}\right)\right). \qquad (2)$$

Dense Graph Pooling

Subsequently, the coarsened adjacency matrix at the l-th pooling layer is calculated as

$$\mathbf{A}^{(l)} = \mathbf{S}^{(l)\top} \mathbf{A}^{(l-1)} \mathbf{S}^{(l)} \tag{3}$$

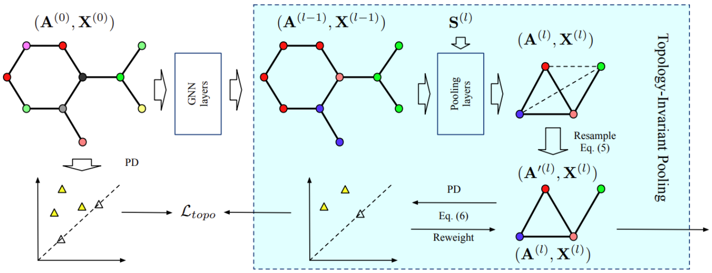
and the corresponding node representations are calculated as

$$\mathbf{X}^{(l)} = \mathbf{S}^{(l)\top} \, \mathrm{GNN}^{(l)} \left(\mathbf{A}^{(l-1)}, \mathbf{X}^{(l-1)} \right). \tag{4}$$

Dense graph pooling above differ from each other in the way to produce $\mathbf{S}$.

Contents

Overview Figure



$(\mathbf{A}^{(0)}, \mathbf{X}^{(0)})$
GNN layers
PD
$\mathcal{L}_{topo}$
$(\mathbf{A}^{(l-1)}, \mathbf{X}^{(l-1)})$
$\mathbf{S}^{(l)}$
Pooling layers
$(\mathbf{A}^{(l)}, \mathbf{X}^{(l)})$
Resample Eq. (5)
$(\mathbf{A}'^{(l)}, \mathbf{X}^{(l)})$
PD
Eq. (6)
Reweight
$(\mathbf{A}^{(l)}, \mathbf{X}^{(l)})$
Topology-Invariant Pooling

Figure: Overview of the method, where the shaded part corresponds to one layer of Topology-Invariant Pooling.

LF

To efficiently integrate PH into GP, the paper proposed using learnable filtration functions that depend on node features and graph topology. These functions are designed to be flexible and computationally efficient, allowing for the dynamic adaptation of filtration criteria as the graph is coarsened during training. One major limitation of utilizing LF is that the computation process is unconscious of edge weights, i.e. edges with non-negative weights will be treated equally, so PH cannot directly extract meaningful topological features from $\mathbf{A}^{(l)}$

Resampling

Therefore, The paper resampled the coarsened adjacency $\mathbf{A}^{(l)}$ obtained from a normal GP layer 3 as:

$$\mathbf{A}'^{(l)} = \text{resample}\left(\frac{\mathbf{A}^{(l)} - \min\left(\mathbf{A}^{(l)}\right)}{\max\left(\mathbf{A}^{(l)}\right) - \min\left(\mathbf{A}^{(l)}\right)}\right) \tag{5}$$

where $\mathbf{A}^{(l)}$ is first normalized in the range of $[0, 1]$, and resample $(\cdot)$ is performed independently for each matrix entry using the Gumbel-softmax trick (Jang et al., 2016).

Persistence Injection

Now $\mathbf{A}'^{(l)} \in \{0,1\}^{n_l \times n_l}$ is a sparse matrix without edge features so we can easily inject topological information into it. For a resampled graph with $\mathbf{A}'^{(l)}$ and $\mathbf{X}^{(l)}$, we formulate the persistence injection as:

$$\mathcal{D}_1 = \mathrm{ph}\left(\mathbf{A}'^{(l)}, \mathrm{sigmoid}\left(\Phi\left(\mathbf{X}^{(l)}\right)\right)\right)$$
$$\mathbf{A}^{(l)} = \mathbf{A}'^{(l)} \odot \text{ dense } \left(\mathcal{D}_1[1] - \mathcal{D}_1[0]\right)$$

where $\odot$ is the Hadamard product, dense() means transforming sparse representations in terms of edges to dense matrix representations, $\mathcal{D}_1[i]$ is the ith value in each tuple of $\mathcal{D}_1$. Since the filtration values are within $[0,1]$, $\mathbf{A}^{(l)}$ after persistence injection is guaranteed to have edge weights in the range of $[0,1]$ and is passed to the next pooling layer.

Topological Loss Function

$$\mathbf{h}_t = \text{transform}\left(\mathcal{D}_1\right)$$

$$\mu = \frac{1}{m}\sum_{t=1}^{m}\mathbf{h}_t, \quad \sigma = \sqrt{\frac{1}{m}\sum_{t=1}^{m}\mathbf{h}_t \odot \mathbf{h}_t - \mu \odot \mu}$$

$$\mathcal{L}_{\text{topo}} = \frac{1}{Ld}\sum_{l=1}^{L}\sum_{i=1}^{d}\left(\left(\mu_i^{(l)} \| \sigma_i^{(l)}\right) - \left(\mu_i^{(0)} \| \sigma_i^{(0)}\right)\right)^2$$