Boosting Graph Pooling with Persistent Homology

Chaolong Ying

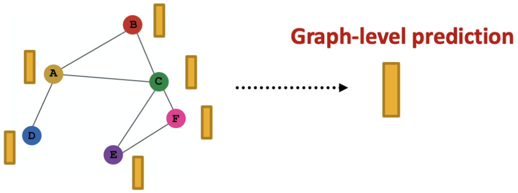
Supervisor: Prof. Tianshu Yu

March 22, 2025

Table of Contents

Graph-level Tasks

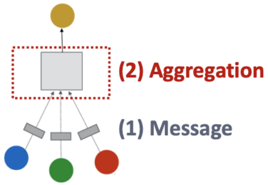
- **Graph-level classification/prediction:** Make prediction using all the node embeddings in a graph



- **Applications:**
  1. Molecular Property Prediction (Predicting whether a molecule has a specific biological activity, e.g., toxicity, solubility)
  2. Social Network Analysis (Identifying types of communities in social networks, e.g. question/answer-based community or a discussion-based community)

Global Pooling

- Global Mean/Max/Sum Pooling to aggregate the message from each node



$$y_G = \mathrm{Mean/Max/Sum}(\{h_v \in \mathbb{R}^d, \forall v \in G\})$$

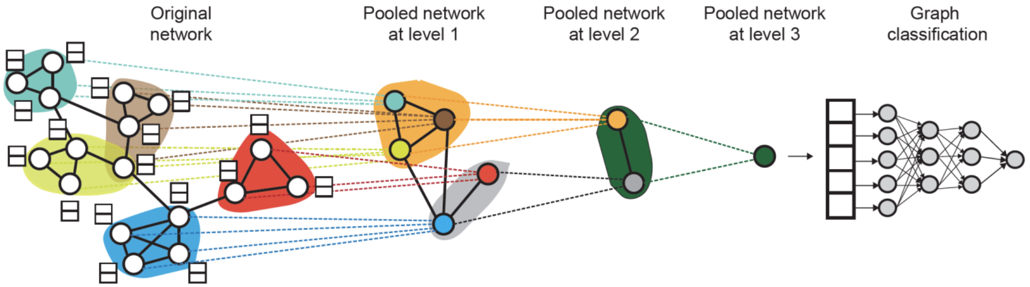
Issue of Global Pooling

- **Issue:** Global pooling over a (large) graph will lose information
- **Toy example:** we use 1-dim node embeddings
    - Node embeddings for $G_1 : \{-1, -2, 0, 1, 2\}$
    - Node embeddings for $G_2 : \{-10, -20, 0, 10, 20\}$
    - Clearly G_1 and G_2 have very different node embeddings
- If we do global sum pooling:
    - Prediction for $G_1 : y_{G_1} = \mathrm{Sum}(\{-1, -2, 0, 1, 2\}) = 0$
    - Prediction for $G_2 : y_{G_2} = \mathrm{Sum}(\{-10, -20, 0, 10, 20\}) = 0$
    - We cannot differentiate G_1 and G_2

Hierarchical Graph Pooling

- **A solution:** Aggregate all the node embeddings hierarchically
- **Toy example:** We will aggregate via $\mathrm{ReLU}(\mathrm{Sum}(\cdot))$. We first separately aggregate the first 2 nodes and last 3 nodes, and then aggregate again to make the final prediction.
- G_1 node embeddings $\{-1, -2, 0, 1, 2\}$
  - **Round 1:** $y_a = \mathrm{ReLU}(\mathrm{Sum}(\{-1, -2\})) = 0$, $y_b = \mathrm{ReLU}(\mathrm{Sum}(\{0, 1, 2\})) = 3$
  - **Round 2:** $y_G = \mathrm{ReLU}(\mathrm{Sum}(\{y_a, y_b\})) = 3$
- G_2 node embeddings $\{-10, -20, 0, 10, 20\}$
  - **Round 1:** $y_a = \mathrm{ReLU}(\mathrm{Sum}(\{-10, -20\})) = 0$, $y_b = \mathrm{ReLU}(\mathrm{Sum}(\{0, 10, 20\})) = 30$
  - **Round 2:** $y_G = \mathrm{ReLU}(\mathrm{Sum}(\{y_a, y_b\})) = 30$
- Now we can differentiate G_1 and G_2

Hierarchical Graph Pooling in Practice


Original network
Pooled network at level 1
Pooled network at level 2
Pooled network at level 3
Graph classification

- Leverage 2 independent Graph Neural Networks (GNNs) at each level.
- Use clustering assignments from GNN B to aggregate node embeddings generated by GNN A.
- Create a single new node for each cluster, maintaining edges between clusters to generated a new pooled network

Formulations of Hierarchical Graph Pooling

Graph Pooling (GP) can be formulated as $\mathcal{G} \mapsto \mathcal{G}_P = (V_P, E_P)$ such that the number of nodes $|V_P| \leq |V|$. Typical methods DiffPool [Ying et al., 2018], MinCutPool [Bianchi et al., 2020], DMoNPool [Tsitsulin et al., 2023] rely on learning a soft cluster assignment $\mathbf{S}^{(l)} \in \mathbb{R}^{n_{l-1} \times n_l}$:

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

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

$$\mathbf{A}^{(l)} = \mathbf{S}^{(l)\top}\mathbf{A}^{(l-1)}\mathbf{S}^{(l)}, \tag{2}$$

and the corresponding node representations are calculated as

$$\mathbf{X}^{(l)} = \mathbf{S}^{(l)\top}\mathrm{GNN}_B^{(l)}\left(\mathbf{A}^{(l-1)}, \mathbf{X}^{(l-1)}\right). \tag{3}$$

These approaches differ from each other in the way to produce $\mathbf{S}$, which is used to inject a bias in the formation of clusters.

Betti Numbers on Graphs

For a graph G with n nodes and m edges

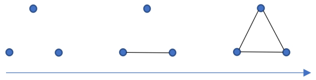
- β_0: number of connected components
- β_1: number of cycles

we have

$$\beta_1 = m + \beta_0 - n$$

Intuition

The addition of a single edge to G will change its Betti numbers, either by merging two connected components (thus decreasing β_0) or by creating an additional cycle (thus increasing β_1).

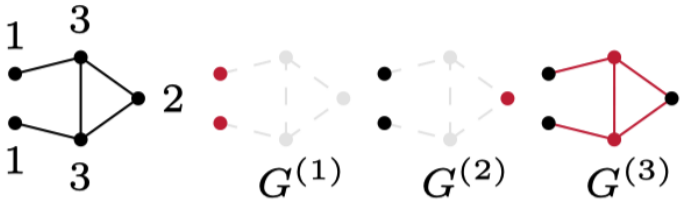


Persistent Homology

- Given a graph $G = (V, E)$, the expressivity of Betti numbers can be increased when paired with a scalar-valued **filtration** function over nodes $f : V \to \mathbb{R}$ or edges $f : E \to \mathbb{R}$.
- Since f can only attain a finite number of values $a_0, a_1, a_2, \ldots$ on the graph, this permits calculating a graph filtration $\emptyset \subseteq G_0 \subseteq G_1 \ldots \subseteq G_{k-1} \subseteq G_k = G$, where each $G_i := (V_i, E_i)$, with $V_i := \{v \in V \mid f(v) \leq a_i\}$ and $E_i := \{v, w \in E \mid max\{f(v), f(w)\} \leq a_i\}$.
- This sublevel set filtration permits tracking topological features, such as cycles, via **persistent homology**, representing each one by a creation and destruction value $\left(f^{(i)}, f^{(j)}\right) \in \mathbb{R}^2$, with $i \leq j$.

Persistent Homology

Example



0-dimensional: (1, ∞), (1, 3), (2, 3), (3, 3), (3, 3)

1-dimensional: (3, ∞)

Table of Contents

Empirical Observations

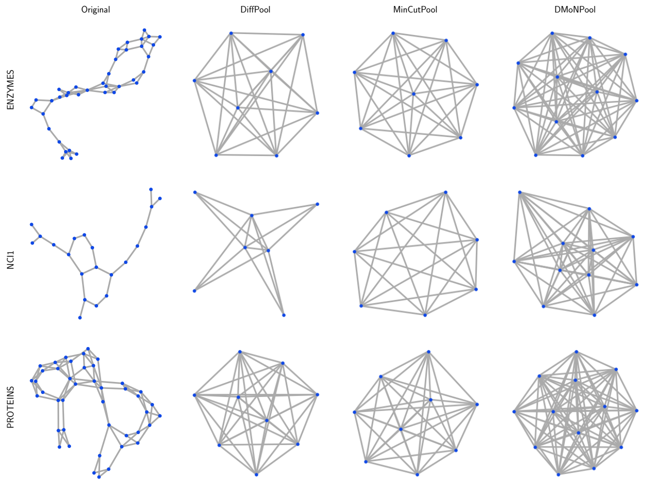
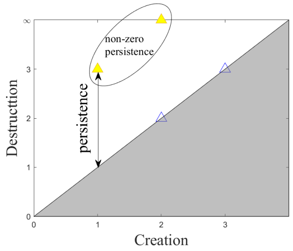


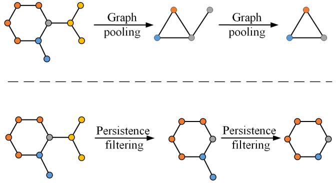
Figure: Visualization of graphs pooled with different methods.

Motivation

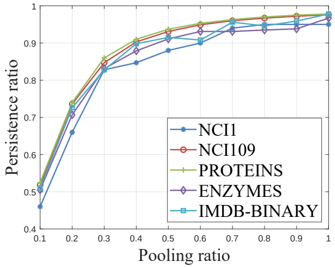
Motivated by Topological autoencoders [Moor et al., 2020], we propose to preserve topological structures in the pooling process.



(a) Persistence diagrams



(b) Hierarchical view of GP and PH



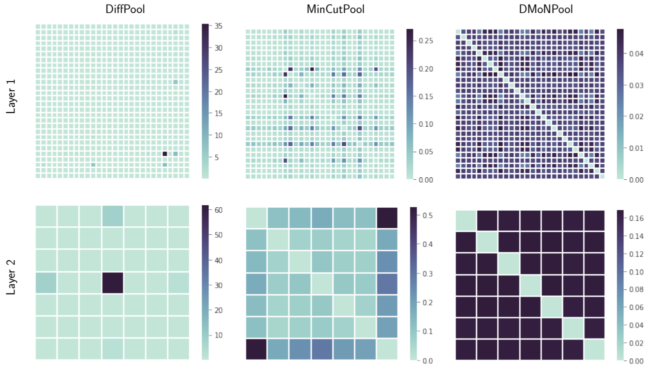
(c) Alignment of GP and PH

Figure: Illustration of Graph Pooling (GP) and Persistent Homology (PH). (a) Illustration of persistence diagrams. (b) GP and PH share a similar hierarchical fashion by coarsening a graph. (c) As a motivating experiment, we gradually change pooling ratio and count how persistence ratio (ratio of non-zero persistence) changes with it.

Table of Contents

Empirical Observations

Issues in existing graph pooling methods: In DiffPool, the edge weights may span a wide range due to the involvement of multiple multiplications in their generation, which hinders the stability and generalization power of the subsequent message passing layers in GNNs. In MinCutPool and DMoNPool, the edge weights are normalized by degree to mitigate numerical explosion. However, this normalization leads to the edge weights becoming excessively smooth and lacking sparsity



Resampling

To solve the issue, we resample the coarsened adjacency $\mathbf{A}^{(l)}$ obtained from a normal graph pooling layer (Eq. 2) as:

$$\mathbf{A}'^{(l)} = \mathrm{resample}\left(\frac{\mathbf{A}^{(l)} - \min(\mathbf{A}^{(l)})}{\max(\mathbf{A}^{(l)}) - \min(\mathbf{A}^{(l)})}\right), \tag{4}$$

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

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:

$$\begin{aligned} \tilde{\mathcal{D}}_1 &= \mathrm{ph}(\mathbf{A}'^{(l)}, \mathrm{sigmoid}(\Phi(\mathbf{X}^{(l)})))[1] \\ \mathbf{A}^{(l)} &= \mathbf{A}'^{(l)} \odot \mathrm{to\_dense}(\tilde{\mathcal{D}}_1[1] - \tilde{\mathcal{D}}_1[0]), \end{aligned} \tag{5}$$

where $\odot$ is the Hadamard product, $\mathrm{to\_dense}()$ means transforming sparse representations in terms of edges to dense matrix representations, $\tilde{\mathcal{D}}_1$ is the augmented 1-dimensional persistence diagrams, ph is the calculation of persistent homology. **Persistence injection can actually be regarded as a reweighting process.**

Topological Loss Function

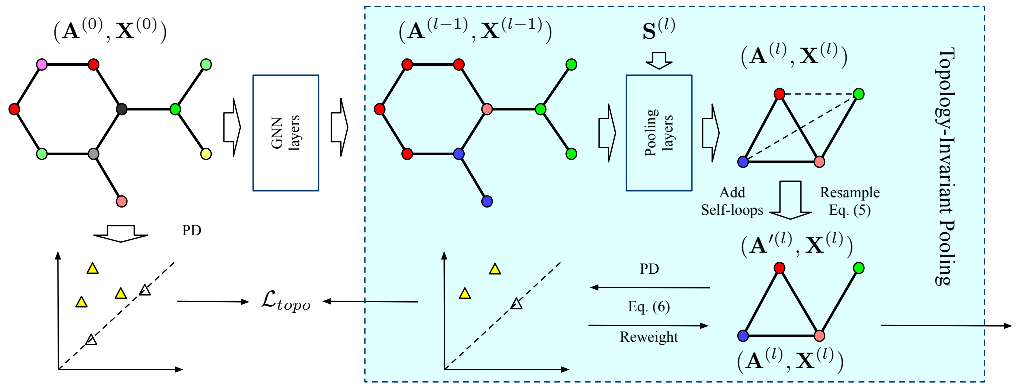
- We propose an additional loss function to implicitly guide the graph pooling process.
- We use several transformations (denoted as $\mathrm{transform}(\cdot)$) to convert the tuples in persistence diagrams into vector $\mathbf{h}_t$ ($t \in [1, m]$). We calculate the mean vector μ as well as the second-order statistics as the standard deviation vector σ as:

$$\mathbf{h}_t = \mathrm{transform}(\tilde{\mathcal{D}}_1)$$
$$\mu = \frac{1}{m}\sum_{t=1}^{m} \mathbf{h}_t, \qquad \sigma = \sqrt{\frac{1}{m}\sum_{t=1}^{m} \mathbf{h}_t \odot \mathbf{h}_t - \mu \odot \mu} \tag{6}$$

- To regularize the topological difference between layers, Topological Loss Function is defined as

$$\mathcal{L}_{topo} = \frac{1}{Ld}\sum_{l=1}^{L}\sum_{i=1}^{d}\left(\left(\mu_i^{(l)} \| \sigma_i^{(l)}\right) - \left(\mu_i^{(0)} \| \sigma_i^{(0)}\right)\right)^2 \tag{7}$$

Overview of Our Method



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

Figure: Overview of our method. The shaded part is one layer of our proposed Topology-Invariant Pooling (TIP) [Ying et al., 2024].

Table of Contents

Graph Classification Results

Table: Test accuracy (↑) of graph classification on benchmark datasets. A **bold** value indicates the overall winner. Gray background indicates that TIP outperforms the base graph pooling methods.

Methods	**Datasets**							
	NCI1	NCI109	ENZYMES	PROTEINS	DD	IMDB-BINARY	IMDB-MULTI	OGBG-MOLHIV
GCN	77.81 ± 1.50	74.90 ± 1.85	32.51 ± 3.35	76.65 ± 3.14	78.66 ± 2.36	74.20 ± 2.40	53.23 ± 3.04	75.04 ± 0.84
GIN	80.30 ± 1.70	79.66 ± 1.55	42.83 ± 3.66	77.18 ± 3.35	78.05 ± 3.60	72.65 ± 3.04	53.28 ± 3.16	76.03 ± 0.84
GraphSAGE	80.85 ± 1.25	79.16 ± 1.28	39.17 ± 3.28	76.67 ± 3.05	78.83 ± 3.07	76.60 ± 2.37	53.46 ± 2.39	76.18 ± 1.27
TOGL	80.53 ± 2.29	78.27 ± 1.39	46.09 ± 3.72	78.17 ± 2.80	76.10 ± 2.24	76.65 ± 2.75	53.87 ± 2.67	77.21 ± 1.33
GSN	83.50 ± 2.00	79.45 ± 1.88	49.50 ± 6.54	74.59 ± 5.00	73.17 ± 4.17	**76.80 ± 2.00**	52.60 ± 3.60	76.06 ± 1.74
Graclus	80.82 ± 1.27	79.13 ± 1.79	41.44 ± 3.46	75.69 ± 2.62	74.67 ± 2.45	74.45 ± 3.29	54.72 ± 2.79	76.81 ± 0.70
TopK	79.43 ± 3.50	77.96 ± 1.58	38.35 ± 4.83	76.03 ± 2.94	76.97 ± 3.94	72.60 ± 4.24	53.66 ± 2.93	76.28 ± 0.67
DiffPool	77.64 ± 1.86	76.50 ± 2.32	48.34 ± 5.14	78.81 ± 3.12	80.27 ± 2.51	73.15 ± 3.30	54.32 ± 2.99	76.60 ± 1.04
DiffPool-TIP	**83.75 ± 1.31**	**81.09 ± 1.65**	**65.05 ± 4.24**	**79.86 ± 3.12**	**82.12 ± 2.53**	76.40 ± 3.13	**55.53 ± 2.92**	**77.75 ± 1.18**
MinCutPool	77.92 ± 1.67	75.88 ± 2.06	39.83 ± 2.63	78.25 ± 3.84	79.15 ± 3.51	73.80 ± 3.54	53.87 ± 2.95	75.60 ± 0.54
MinCutPool-TIP	80.17 ± 1.29	79.48 ± 1.37	46.34 ± 3.85	79.73 ± 3.27	80.87 ± 2.47	75.20 ± 2.67	54.47 ± 2.27	77.18 ± 0.83
DMoNPool	78.03 ± 1.64	76.62 ± 1.94	40.82 ±3.68	78.63 ± 3.89	79.16 ± 3.61	73.50 ± 3.01	54.07 ± 3.08	76.30 ± 1.34
DMoNPool-TIP	79.68 ± 1.38	78.46 ± 1.50	45.84 ± 5.32	79.73 ± 3.66	81.46 ± 2.96	74.25 ± 2.93	54.23 ± 2.64	76.70 ± 0.62

Visualization

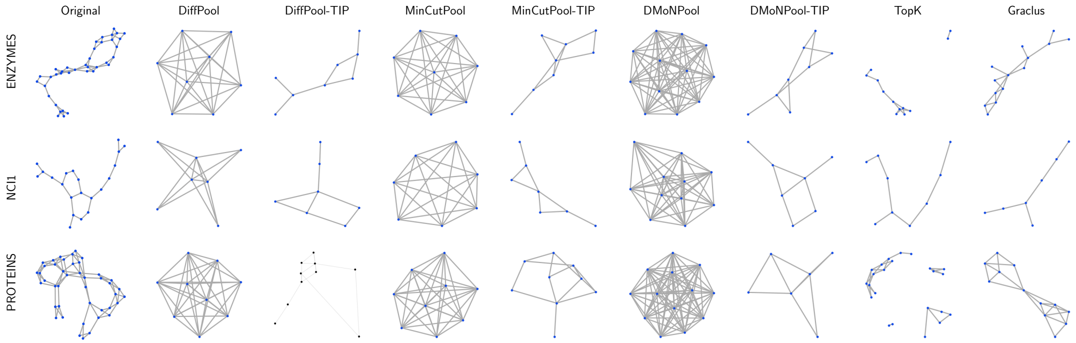


Figure: Graphs pooled with different methods in graph classification experiment.

Training Curves

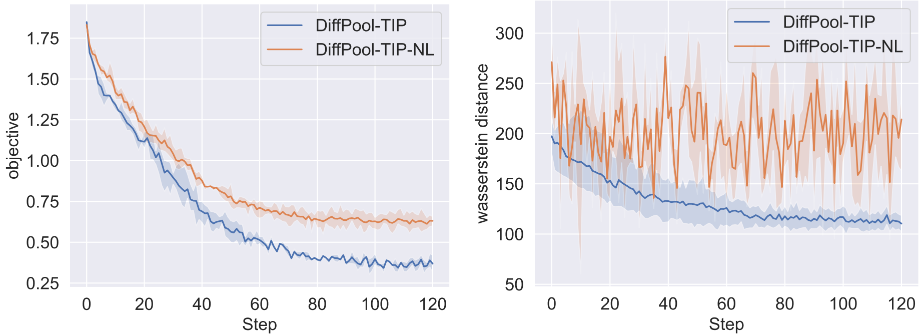


Figure: The training curves of DiffPool-TIP and DiffPool-TIP-NL on ENZYMES dataset. We show the average values and min-max range of objective and Wasserstein distance for multiple runs.

References

Bianchi, F. M., Grattarola, D., and Alippi, C. (2020).
Spectral clustering with graph neural networks for graph pooling.
In *International conference on machine learning*, pages 874–883. PMLR.

Moor, M., Horn, M., Rieck, B., and Borgwardt, K. (2020).
Topological autoencoders.
In *International conference on machine learning*, pages 7045–7054. PMLR.

Tsitsulin, A., Palowitch, J., Perozzi, B., and Müller, E. (2023).
Graph clustering with graph neural networks.
Journal of Machine Learning Research, 24(127):1–21.

Ying, C., Zhao, X., and Yu, T. (2024).
Boosting graph pooling with persistent homology.
In *Advances in Neural Information Processing Systems*, volume 37, pages 19087–19113.

Ying, Z., You, J., Morris, C., Ren, X., Hamilton, W., and Leskovec, J. (2018).
Hierarchical graph representation learning with differentiable pooling.
In *Advances in neural information processing systems*, volume 31.

The End