Research and Analysis on Graph Pooling

Bowen Pei

Pooling layer in CNN

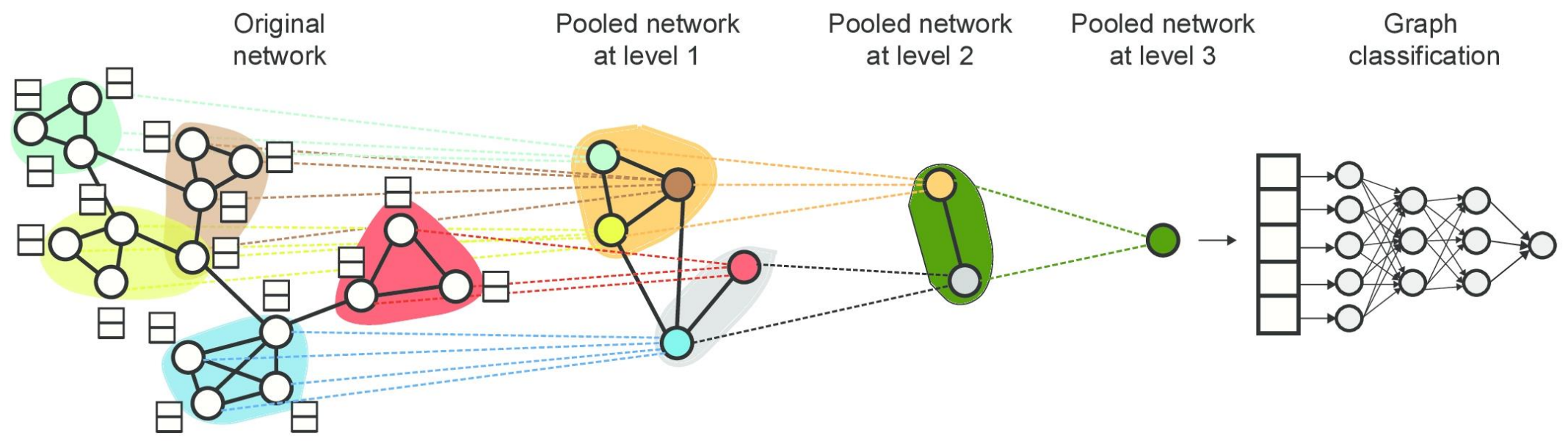
- Function:
- (1) Retaining the main features while reducing parameters (lowering latitude, similar to PCA) and calculations to prevent overfitting
- (2) Invariance, this invariance includes translation, rotation, scale
- ——> Preserve the graph structure

降维

feature integration.

Pooling

Schematic diagram



Notations:

- $G = \{E, V\}$,
- $V = \{v_1, \ldots, v_N\}$ is the set of N nodes
- E is the set of edges.
- Adjacency matrix $A \in R^{N \times N}$.
- Node feature matrix $X \in R^{N \times d}$, d is the dimension of features.
- Node embedding matrix $Z \in R^{N \times d}$

DiffPool

- Assignment matrix $S^{(l)} \in \mathbb{R}^{n_l \times n_{l+1}}$

$$Z^{(l)} = \mathrm{GNN}_{l,\mathrm{embed}}(A^{(l)}, X^{(l)}),$$

$$S^{(l)} = \mathrm{softmax}\left(\mathrm{GNN}_{l,\mathrm{pool}}(A^{(l)}, X^{(l)})\right),$$

$$X^{(l+1)} = S^{(l)^T} Z^{(l)} \in \mathbb{R}^{n_{l+1} \times d},$$

$$A^{(l+1)} = S^{(l)^T} A^{(l)} S^{(l)} \in \mathbb{R}^{n_{l+1} \times n_{l+1}}.$$

- Note that these two GNNs consume the same input data but have distinct parameterizations and play separate roles.
- Computational performance:

  Storage complexity: O (k|V|^2)

  Time complexity: O (k|V|^3) where V and k denote vertices and pooling ratio

DiffPool

- Pros:

  (a) It can generate hierarchical representations of graphs.

  (b) It can be combined with various graph neural network architectures in an end-to-end fashion.

- Cons:

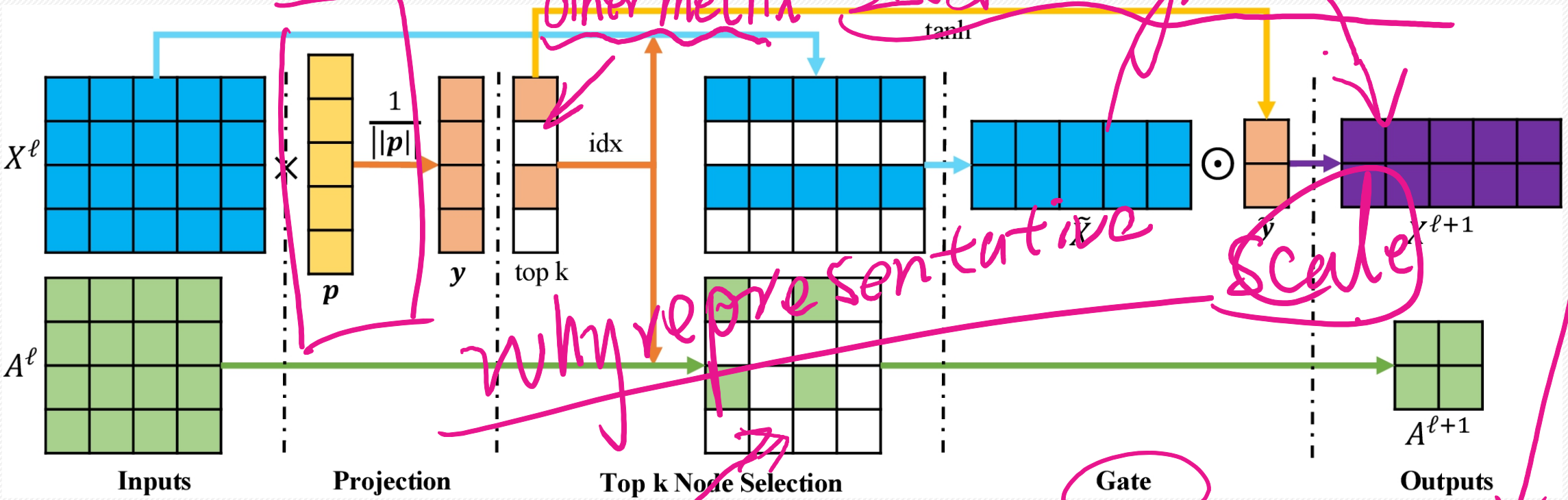
  (a) The number of clusters has to be chosen in advance, which might cause performance issues when used on datasets with different graph sizes.

  (b) Since cluster assignment is based only on node features, nodes are assigned to the same cluster based on their features, ignoring distances.

  (c) The cluster assignment matrix is dense and the complexity is large.

TopKPool

- Schematic diagram



TopKPool

- Pros:

(a) Sparse, the computational complexity is reduced to O(|E|). Storage complexity: O(|V|+|E|) where V , E, and k denote vertices, edges, and pooling ratio, respectively

(b) Variable in graph size.

- Cons:

(a) Adding nodes to a graph can change the pooling result of the whole graph. (locally dependent)

(b) Whole areas of a graph might see no node chosen, which causes loss of information.

SAGPool

- A variant of TopKPool, SAGPool no longer uses only node features to compute node scores but uses graph convolutions to take neighbouring node features into account.
- The self-attention score $Z \in R^{N\times 1}$ is calculated as follows:

$$Z = \sigma(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}X\Theta_{att})$$

1、2种方法的区别

- The top k*N nodes are selected based on the value of Z.

$$\text{idx} = \text{top-rank}(Z, \lceil kN \rceil), \quad Z_{mask} = Z_{\text{idx}}$$

SAGPool

- Pros:

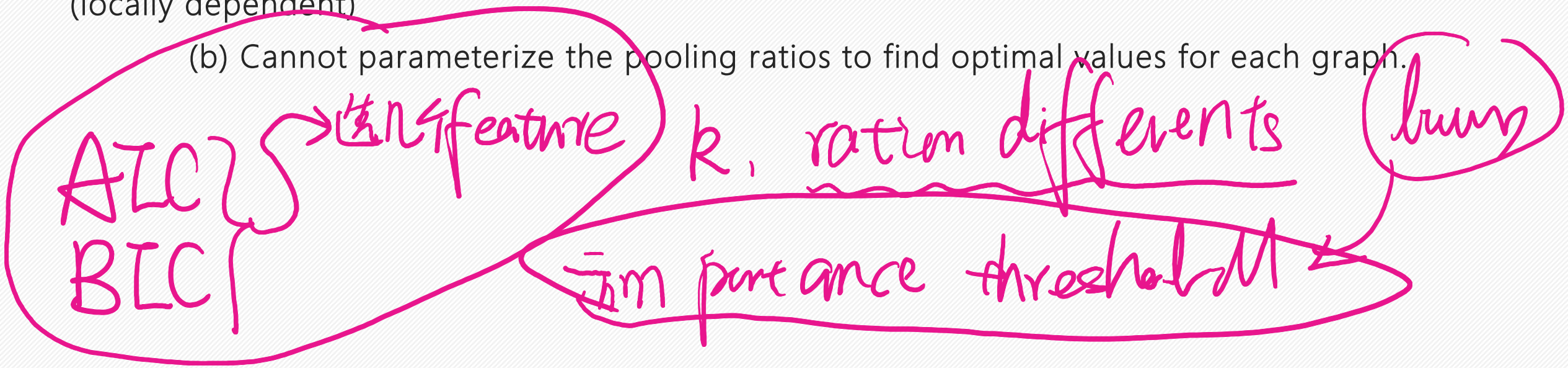
  (a) Sparse, the same complexity

  (b) It can be both H architecture and G architecture. Therefore, it can be used to learn both many vertexes and few vertexes graph.

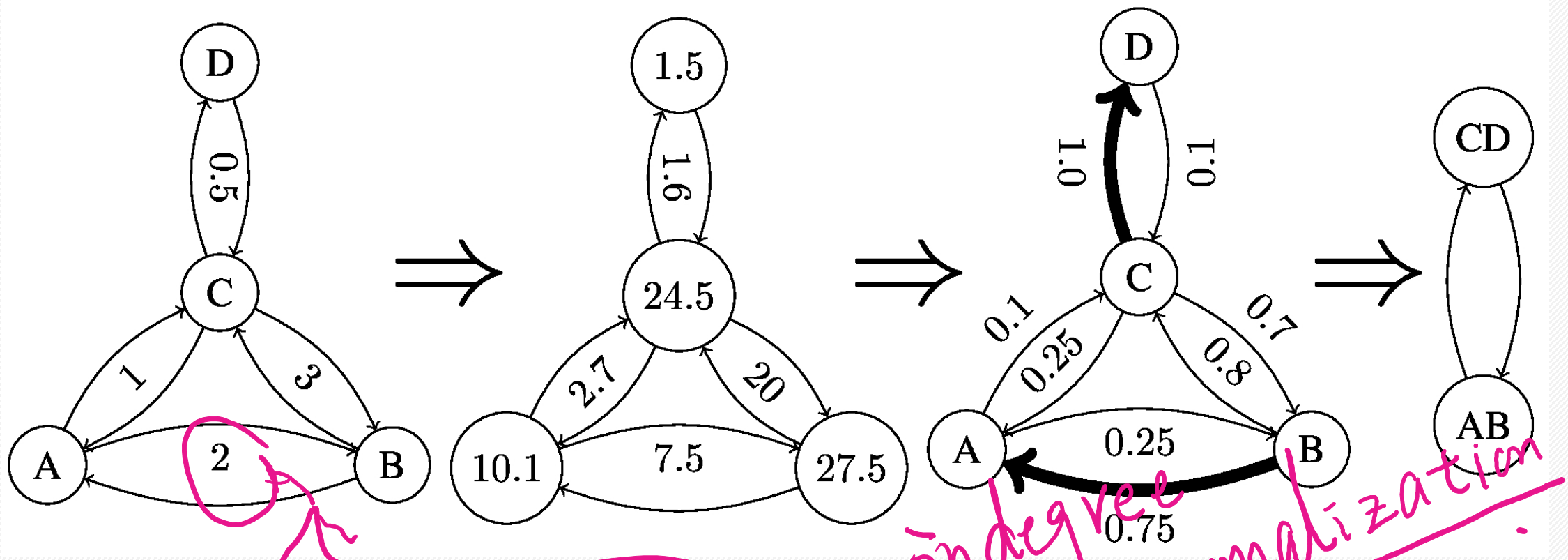
- Cons:

  (a) Adding nodes to a graph can change the pooling result of the whole graph. (locally dependent)

  (b) Cannot parameterize the pooling ratios to find optimal values for each graph.



EdgePool


D
0.5
C
1
3
A
2
B
1.5
1.6
24.5
2.7
20
10.1
7.5
27.5
D
1.0
0.1
C
0.1
0.25
0.7
0.8
A
0.25
B
0.75
CD
AB
indegree
normalization

EdgePool

- Based on edge contraction
- 1) Choosing edges. For an edge from node i to node j, we compute the raw score r as

$$r(e_{ij}) = W \cdot (n_i \| n_j) + b,$$

- Where n_i and n_j are the node features and W and b are learned parameters.

- 2) Computing new node features:

$$\hat{n}_{ij} = s_{ij}(n_i + n_j)$$

- 3) Integrating edge features:

$$r(e_{ij}) = W \cdot (n_i \| n_j \| f_{ij}) + b.$$

EdgePool

- Pros:

  (a) Both runtime and memory scales linearly in the number of edges.

  (b) Locally independent: As long as the node scores of two nodes n_i and n_j and of their neighbours do not change (by changing nodes within the receptive fields), the choice of edge e_{ij} will not change.

- Cons:

  (a) Fixed pooling ratio. (50%)

EigenPool

- 1) Graph Coarsening: we adopt spectral clustering to obtain the subgraphs.

- 2) Transform the original graph signal information into the graph signal defined on the coarsened graph:
- Based on local graph Fourier transform
- Define the smoothness:

$$s(\mathbf{x}) = \mathbf{x}^T \mathbf{L}\mathbf{x} = \frac{1}{2}\sum_{i,j}^{N} \mathbf{A}[i,j](\mathbf{x}[i] - \mathbf{x}[j])^2.$$

.....

- Too much math, so omit

EigenPool

- Pros:

  (a) Can extract subgraph information utilizing both node features and structure of the subgraph.

- Cons:

  (a) Code not available.

  (b) Locally dependent.

THANKS!