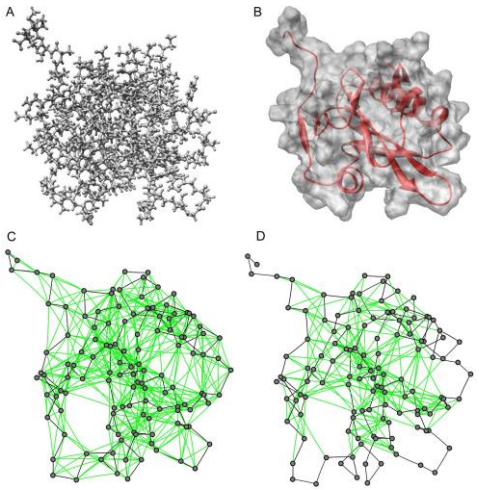
Bottom-Up and Top-Down Graph Pooling

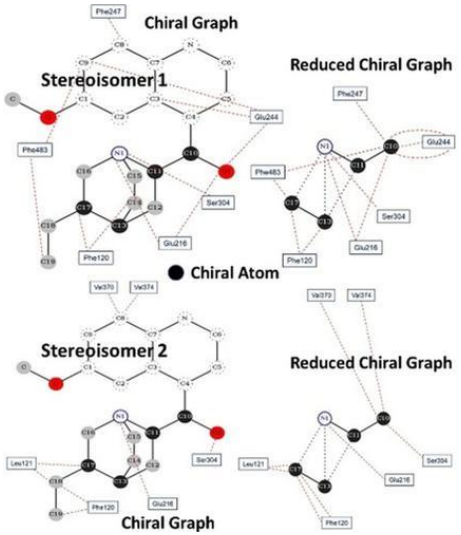
Jia-Qi Yang[1], De-Chuan Zhan[1]✉, and Xin-Chun Li[1]

National Key Laboratory for Novel Software Technology
Nanjing University, Nanjing 210023, China
{yangjq,lixc}@lamda.nju.edu.cn
zhandc@nju.edu.cn✉

Graph Classification



(a) Graph structures of proteins

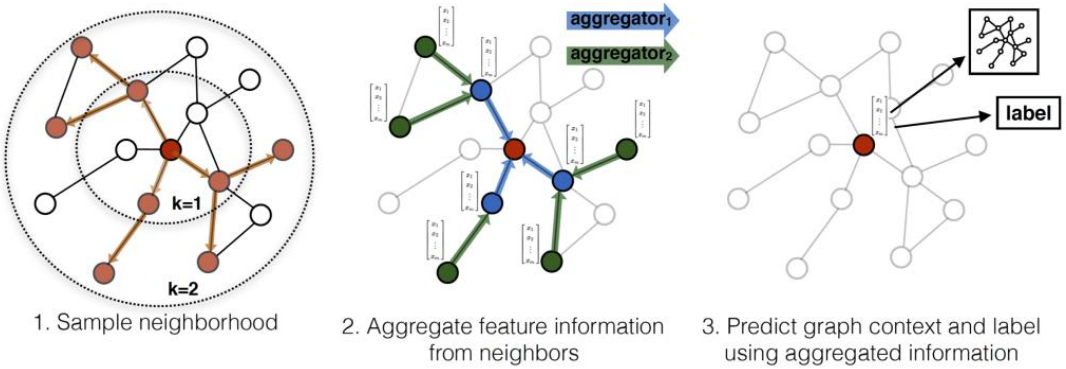


(b) Graph structures of chemical compounds

Proteins, chemical compounds or social groups can be modeled by complex graphs. It will be very useful if we can build a model that can predict some specific properties, e.g. whether a protein is effective for specific purpose.

Graph Neural Networks

Graph neural networks(GNNs) are neural networks designed for graph-structured data, and graph convolutional neural networks (GCNs) are one of the most prominent variants of GNNs.



Graph-SAGE

Graph-SAGE is a typical GNN variant, and Graph-SAGE using mean aggregation is a linear approximation of a localized spectral convolution. This figure shows how most GNNs work.

Inductive Representation Learning on Large Graphs, William L. Hamilton, Rex Ying, Jure Leskovec, 2017

Graph Neural Networks

GCNs and most GNNs are analogous to CNNs by design

(1) GCNs are based on graph convolution, which is analogy of convolution on graph spectral domain.
(2) Popular GCNs use localized kernel so that only near by nodes within several hops are considered. This local property is intuitive and preferred in most GNN architectures. We call features produced by such architectures as **local features**.

We will focus on experiments using GCN architecture, but the following analysis is applicable to all GNNs with local features.

Graph Pooling

Pooling layers are important components in most successful GNN architectures for graph classification, which can increase receptive field and decrease feature size in order to learn a good high-level representation.

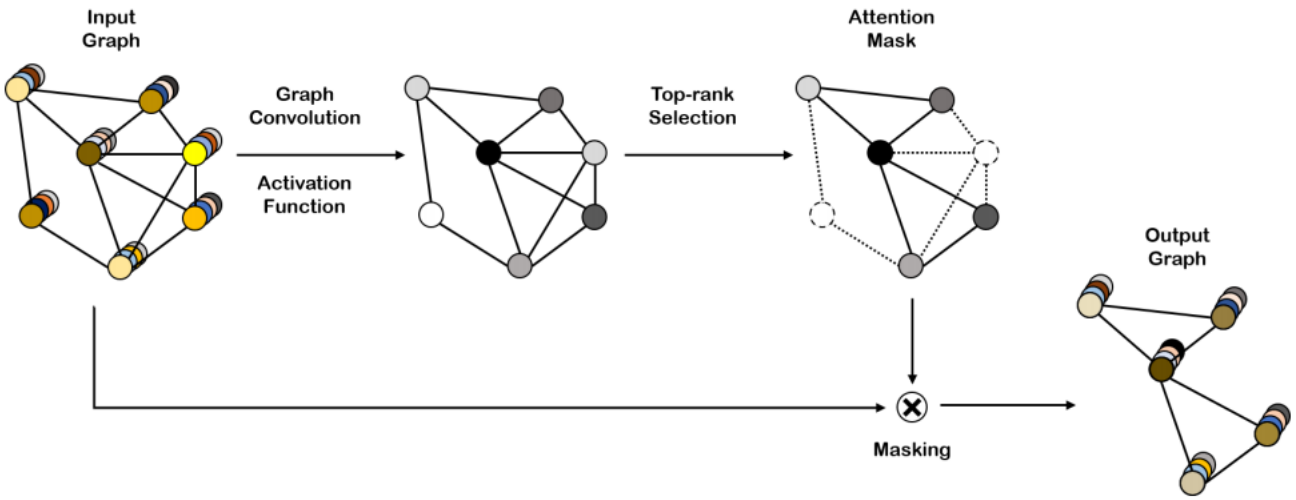


Figure 1. An illustration of the SAGPool layer.

Self-Attention Graph Pooling, Junhyun Lee, Inyeop Lee, Jaewoo Kang, 2019

Existing graph pooling methods

Hierarchical and Global Graph Pooling: Global pooling aims at obtaining a global summary of a whole graph, while hierarchical pooling methods aim to provide a texture-level to object-level representations at different layers. We focus on hierarchical pooling since it's more important in deep models.

Hierarchical pooling methods:

1. **Score-based graph pooling**: a score is calculated for every node, then the top-k nodes with largest scores are selected, SAGPool and gPool are typical score-based pooling methods. Score-based methods have the same complexity as GNN blocks.
2. **Differentiable graph pooling**: DiffPool is a representative differentiable pooling method where a large graph is downsampled to a small graph with pre-fixed size in a fully differentiable approach. However, DiffPool suffers from it's high computational complexity and is not applicable on large graphs. So we only focus on score-based methods.

Existing graph pooling methods

Why did pooling in CNNs work well?

1. The grid structure in image or audio is exploited by convolution, where we share parameters among different spatial location. The convolution layers produce **local features**.
2. The pooling methods select value within a small **local region**, since they are **local features**, they are **comparable**.

The above 1 holds true for GCNs, however, 2 does not hold for graph poolings. We found that there is a **contradiction between local features and global pooling**:

1. Since it's difficult to define "local region" in graphs, existing graph poolings compare scores of **all** nodes within a graph. But this comparison is still based on **local features**.
2. Local features do not have any **global information**, so such global pooling can hardly keep **global structures**. Notice that poolings in CNNs do not have such problem, since the grid pooling preserves global (grid) structures by design.

Proposed method

The analysis of the drawback of graph pooling methods inspired two possible solution:

1. Adjust graph pooling to local pooling. However, there does not exist a good way to define “local” in graphs now.
2. Make local features non-local, so we should inject some global information to help learn a better global pooling.

We propose a bottom-up and top-down graph pooling method that follows the second idea.

Proposed method

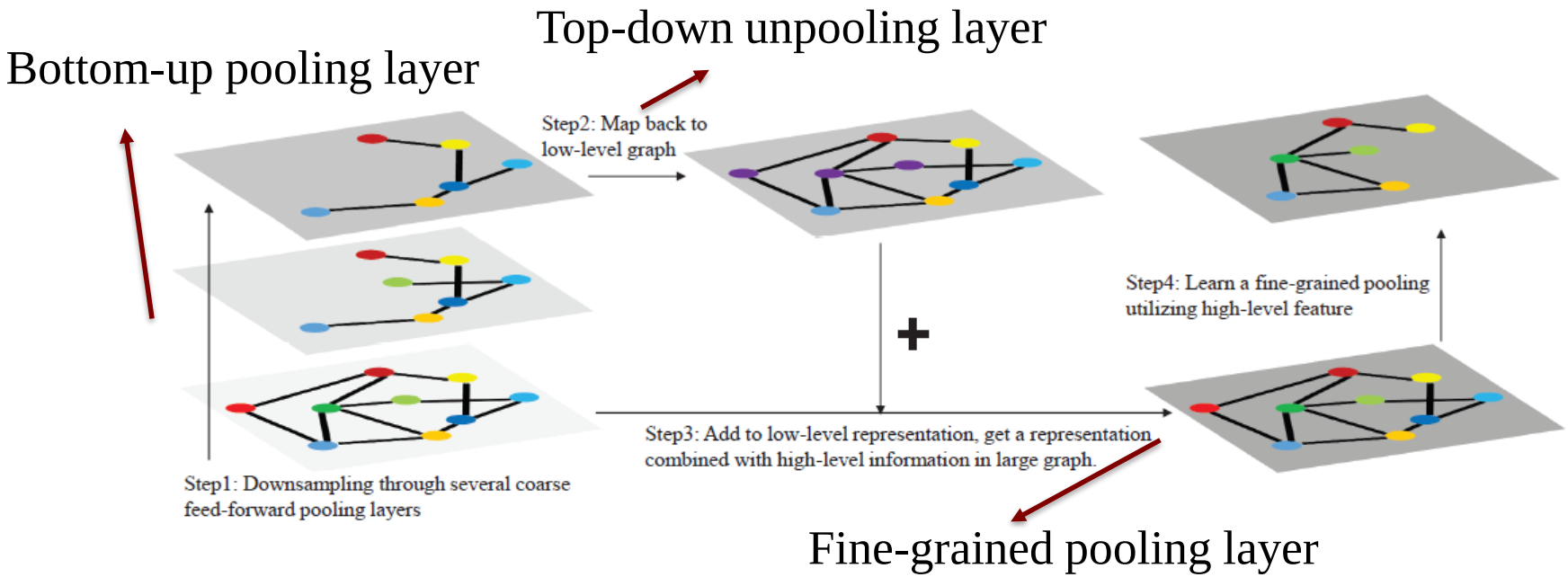


Fig. 1. A bottom-up and top-down graph pooling layer. We use different background color to denote effective receptive field, the darker the background the larger the effective receptive field. Nodes with same color denote the same nodes at different level. This figure depicted a possible pooling procedure on a simple graph.

Proposed method

➢ Bottom-up Pooling

A bottom-up pooling layer (BUPL) can be defined as a stack of base pooling layers, for example, a bottom-up pooling layer with 2 base pooling layers can be defined by

$$\tilde{X}_1^\ell, \tilde{A}_1^\ell = \mathrm{BPL}_{\Theta_{bu1}}(X^\ell, A^\ell, r_{bu}) \tag{2}$$

$$\tilde{X}^\ell, \tilde{A}^\ell = \mathrm{BPL}_{\Theta_{bu2}}(\tilde{X}_1^\ell, \tilde{A}_1^\ell, r_{bu}) \tag{3}$$

The corresponding bottom-up pooling layer is denoted by

$$\tilde{X}^\ell, \tilde{A}^\ell, \mathrm{idx}_{\mathrm{bu}} = \mathrm{BUPL}_{\Theta_{bu1};\Theta_{bu2}}(X^\ell, A^\ell, r_{bu}) \tag{4}$$

Where $\tilde{X}^\ell, \tilde{A}^\ell$ is the output of a bottom-up pooling layer. Different from SAGPool layer, we return an index $\mathrm{idx}_{\mathrm{bu}}$ in BUPL to memorize the map between input nodes and output nodes.

Proposed method

- Top-down Unpooling

The top-down unpooling can be denoted as follows:

$$\hat{X}^{\ell}, \hat{A}^{\ell} = \mathrm{TDUPL}(\tilde{X}^{\ell}, \tilde{A}^{\ell}, \mathrm{idx}_{\mathrm{bu}}) \tag{5}$$

Where TDUPL is the top-down unpooling layer.

If a node is retained during bottom-up pooling, the high-level features are mapped back. Otherwise we use the mean value of all retained nodes.

Proposed method

- Fine-Grained Pooling Layer

The high-level features (result of unpooling) are added with original features, then a fine-grained pooling layer is learnt on these features.

$$Z^{\ell} = X^{\ell} + \hat{X}^{\ell} \tag{6}$$

$$X^{\ell+1}, A^{\ell+1} = \mathrm{FGPL}_{\Theta_{fg}}(Z^{\ell}, A^{\ell}, r) \tag{7}$$

Where FGPL is the fine-grained pooling layer, which can be any score based pooling layer. Notice that the feature Z used by FGPL is a feature mixed with local feature and global feature, thus the FGPL has the potential to overcome the drawback analyzed before.

Proposed method

Algorithm 1: A Bottom-Up and Top-Down Pooling Layer

Input : Adjacent matrix A^{ℓ}; Input feature matrix X^{ℓ}; Bottom-up pooling layer parameters Θ_{bu}^{ℓ} and fine-grained pooling layer parameters Θ_{fg}^{ℓ}; Bottom-up pooling ratio r_{bu} and pooling ratio r

Output: A smaller graph with adjacent matrix $A^{\ell+1}$; Output feature matrix $X^{\ell+1}$

1 $\tilde{X}^{\ell}, \tilde{A}^{\ell}, \mathrm{idx_{bu}} = \mathrm{BUPL}_{\Theta_{bu}^{\ell}}(X^{\ell}, A^{\ell}, r_{bu})$ Bottom-up pooling

2 $\hat{X}^{\ell}, \hat{A}^{\ell} = \mathrm{TDUPL}(\tilde{X}^{\ell}, \tilde{A}^{\ell}, \mathrm{idx_{bu}})$ Top-down unpooling

3 $X^{\ell+1}, A^{\ell+1} = \mathrm{FGPL}_{\Theta_{fg}^{\ell}}(\hat{X}^{\ell} + X^{\ell}, \hat{A}^{\ell}, r)$

This addition is the key point that global information is mapped back to local features.

Experiments

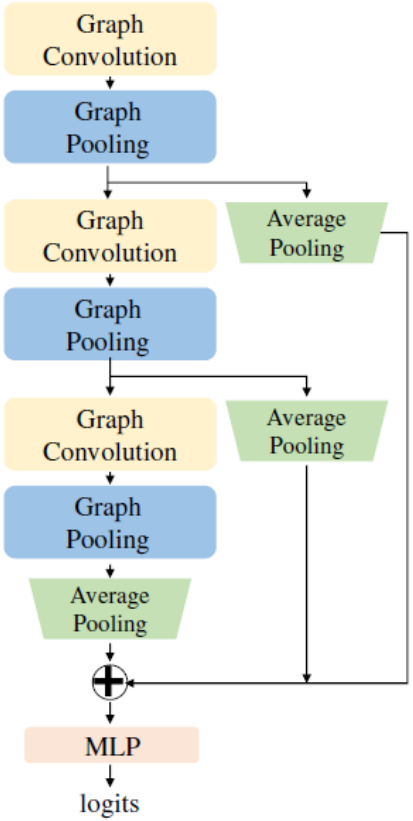


Fig. 2. Model Architecture

For fair comparison, we use the same architecture for all compared methods. Only graph pooling blocks are changed to different pooling methods.

Experiments

Table 2. Average accuracy and standard deviation of 20 random runs. *: About 2% largest of graphs in D&D dataset are dropped because of being too large for efficiency when training and evaluating DiffPool. NA: We found DiffPool on REDDIT dataset is very slow or cause out-of-memory, since we focus on efficient pooling method we excluded this experiment.

Models	**D&D**	**PROTEINS**	**NCI1**	**NCI109**	**REDDIT**
DiffPool	77.24*	74.55	70.87	**72.73**	NA
gPool	75.12	73.61	71.71	69.14	48.02
SAGPool	75.50	75.22	**73.09**	72.01	49.70
BUTDPool(ours)	**77.43**	**75.44**	73.00	**72.28**	**52.14**

From Table 2, we can see that, BUTDPool achieves clear performance gain over the compared methods, especially on D&D and REDDIT dataset with large graphs.

Thanks!