arXiv:1904.08082v3 [cs.LG] 24 Apr 2019

Self-Attention Graph Pooling

Junhyun Lee [*1] **Inyeop Lee** [*1] **Jaewoo Kang** [1]

Abstract

Advanced methods of applying deep learning to structured data such as graphs have been proposed in recent years. In particular, studies have focused on generalizing convolutional neural networks to graph data, which includes redefining the convolution and the downsampling (pooling) operations for graphs. The method of generalizing the convolution operation to graphs has been proven to improve performance and is widely used. However, the method of applying downsampling to graphs is still difficult to perform and has room for improvement. In this paper, we propose a graph pooling method based on self-attention. Self-attention using graph convolution allows our pooling method to consider both node features and graph topology. To ensure a fair comparison, the same training procedures and model architectures were used for the existing pooling methods and our method. The experimental results demonstrate that our method achieves superior graph classification performance on the benchmark datasets using a reasonable number of parameters.

1. Introduction

The advent of deep learning has led to extensive improvements in technology used to recognize and utilize patterns in data (LeCun et al., 2015). In particular, convolutional neural networks (CNNs) successfully leverage the properties of data such as images, speech, and video on Euclidean domains (grid structure) (Hinton et al., 2012; Krizhevsky et al., 2012; He et al., 2016; Karpathy et al., 2014). CNNs consist of convolutional layers and downsampling (pooling) layers. The convolutional and pooling layers exploit the shift-invariance (also known as stationary) property and compositionality of grid-structured data (Simoncelli & Olshausen, 2001; Bronstein et al., 2017). As a result, CNNs perform well with a small number of parameters.

In various fields, however, a large amount of data, such as graphs, exists on the non-Euclidean domain. For example, social networks, biological networks, and molecular structures can be represented by nodes and edges of graphs (Lazer et al., 2009; Davidson et al., 2002; Duvenaud et al., 2015). Therefore, attempts have been made to successfully use CNNs in the non-Euclidean domain. Most previous studies have redefined the convolution and pooling layers to process graph data.

To define graph convolution, studies have used the spectral (Bruna et al., 2014; Henaff et al., 2015; Defferrard et al., 2016; Kipf & Welling, 2016) and non-spectral (Monti et al., 2017; Hamilton et al., 2017; Xu et al., 2018a; Velikovi et al., 2018; Morris et al., 2018) methods. The application of graph convolution has resulted in outstanding performance in a variety of fields which include recommender systems (van den Berg et al., 2017; Yao & Li, 2018; Monti et al., 2017), chemical researches (You et al., 2018; Zitnik et al., 2018), natural language processing (Bastings et al., 2017; Peng et al., 2018; Yao et al., 2018), and in many tasks as reported in Zhou et al..

There are fewer methods for graph pooling than for graph convolution. Previous researches have adopted the pooling method that considers only graph topology (Defferrard et al., 2016; Rhee et al., 2018). With growing interest in graph pooling, several improved methods have been proposed (Dai et al., 2016; Duvenaud et al., 2015; Gilmer et al., 2017b; Zhang et al., 2018b). They utilize node features to obtain a smaller graph representation. Recently, Ying et al.; Gao & Ji; Cangea et al. have proposed innovative pooling methods that can learn hierarchical representations of graphs. These methods allow Graph Neural Networks (GNNs) to attain scaled-down graphs after pooling in an end-to-end fashion.

However, the above pooling methods have room for improvement. For example, the differentiable hierarchical pooling method of Ying et al. has a quadratic storage complexity and the number of its parameters is dependent on the number of nodes. Gao & Ji; Cangea et al. have addressed the complexity issue, but their method does not take graph topology into account.

Here, we propose SAGPool which is a Self-Attention Graph

[*]Equal contribution [1]Department of Computer Science and Engineering, Korea University, Seoul, Korea. Correspondence to: Jaewoo Kang <kangj@korea.ac.kr>.

Pooling method for GNNs in the context of hierarchical graph pooling. Our method can learn hierarchical representations in an end-to-end fashion using relatively few parameters. The self-attention mechanism is exploited to distinguish between the nodes that should be dropped and the nodes that should be retained. Due to the self-attention mechanism which uses graph convolution to calculate attention scores, node features and graph topology are considered. In short, SAGPool, which has the advantages of the previous methods, is the first method to use self-attention for graph pooling and achieve high performance.

2. Related Work

GNNs have drawn considerable attention due to their state-of-the-art performance on tasks in the graph domain. Studies on GNNs focus on extending the convolution and pooling operation, which are the main components of CNN, to graphs.

2.1. Graph Convolution

Convolution operation on graphs can be defined in either the spectral or non-spectral domain. Spectral approaches focus on redefining the convolution operation in the Fourier domain, utilizing spectral filters that use the graph Laplacian. Kipf & Welling proposed a layer-wise propagation rule that simplifies the approximation of the graph Laplacian using the Chebyshev expansion method (Defferrard et al., 2016). The goal of non-spectral approaches is to define the convolution operation so that it works directly on graphs. In general non-spectral approaches, the central node aggregates features from adjacent nodes when its features are passed to the next layer rather than defining the convolution operation in the Fourier domain. Hamilton et al. proposed GraphSAGE which learns node embeddings through sampling and aggregation. While GraphSAGE operates in a fixed-size neighborhood, Graph Attention Network (GAT) (Velikovi et al., 2018), based on attention mechanisms (Bahdanau et al., 2014), computes node representations in entire neighborhoods. Both approaches have improved performance on graph-related tasks.

2.2. Graph Pooling

Pooling layers enable CNN models to reduce the number of parameters by scaling down the size of representations, and thus avoid overfitting. To generalize CNNs, the pooling method for GNNs is necessary. Graph pooling methods can be grouped into the following three categories: topology based, global, and hierarchical pooling.

Topology based pooling Earlier works used graph coarsening algorithms rather than neural networks. Spectral clustering algorithms use eigendecomposition to obtain coarsened graphs. However, alternatives were needed due to the time complexity of eigendecomposition. Graclus(Dhillon et al., 2007) computes clustered versions of given graphs without eigenvectors because of the mathematical equivalence between a general spectral clustering objective and a weighted kernel k-means objective. Even in recent GNN models (Defferrard et al., 2016; Rhee et al., 2018), Graclus is employed as a pooling module.

Global pooling Unlike the previous methods, global pooling methods consider graph features. Global pooling methods use summation or neural networks to pool all the representations of nodes in each layer. Graphs with different structures can be processed because global pooling methods collect all the representations. Gilmer et al. viewed GNNs as message passing schemes, and proposed a general framework for graph classification where entire graph representations could be obtained by utilizing the Set2Set(Vinyals et al., 2015) method. SortPool(Zhang et al., 2018b) sorts embeddings for nodes according to the structural roles of a graph and feeds the sorted embeddings to the next layers.

Hierarchical pooling Global pooling methods do not learn hierarchical representations which are crucial for capturing structural information of graphs. The main motivation of hierarchical pooling methods is to build a model that can learn feature- or topology-based node assignments in each layer. Ying et al. proposed DiffPool which is a differentiable graph pooling method that can learn assignment matrices in an end-to-end fashion. A learned assignment matrix in layer l, $S^{(l)} \in \mathbb{R}^{n_l \times n_{l+1}}$ contains the probability values of nodes in layer l being assigned to clusters in the next layer $l+1$. Here, n_l denotes the number of nodes in layer l. Specifically, nodes are assigned by the following equation:

$$S^{(l)} = \text{softmax}(\text{GNN}_l(A^{(l)}, X^{(l)}))$$
$$A^{(l+1)} = S^{(l)\top} A^{(l)} S^{(l)} \tag{1}$$

where X denotes the node feature matrix and A is the adjacency matrix.

Cangea et al. utilized gPool(Gao & Ji, 2019) and achieved performance comparable to that of DiffPool. gPool requires a storage complexity of $\mathcal{O}(|V| + |E|)$ whereas DiffPool requires $\mathcal{O}(k|V|^2)$ where V, E, and k denote vertices, edges, and pooling ratio, respectively. gPool uses a learnable vector p to calculate projection scores, and then uses the scores to select the top ranked nodes. Projection scores are obtained by the dot product between p and the features of all the nodes. The scores indicate the amount of information of nodes that can be retained. The following equation roughly describes the pooling procedure in gPool.

$$y = X^{(l)}\mathbf{p}^{(l)}/\|\mathbf{p}^{(l)}\|, \quad \text{idx} = \text{top-rank}(y, \lceil kN \rceil)$$
$$A^{(l+1)} = A^{(l)}_{\text{idx,idx}} \tag{2}$$

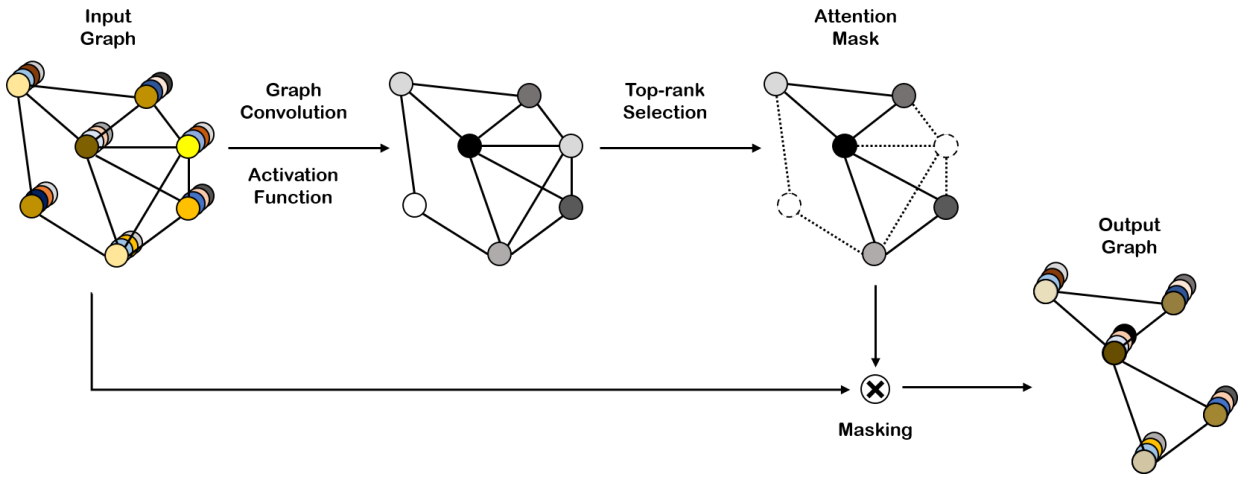

Figure 1. An illustration of the SAGPool layer.

As in Equation (2), the graph topology does not affect the projection scores.

To further improve graph pooling, we propose SAGPool which can use features and topology to yield hierarchical representations with a reasonable complexity of time and space.

3. Proposed Method

The key point of SAGPool is that it uses a GNN to provide self-attention scores. In Section 3.1, we describe the mechanism of SAGPool and its variants. Model architectures for the evaluations are described in Section 3.2. The SAGPool layer and the model architectures are illustrated in Figure 1 and Figure 2, respectively.

3.1. Self-Attention Graph Pooling

Self-attention mask Attention mechanisms have been widely used in the recent deep learning studies (Parikh et al., 2016; Cheng et al., 2016; Zhang et al., 2018a; Velikovi et al., 2018). Such mechanisms make it possible to focus more on important features and less on unimportant features. In particular, self-attention, commonly referred to as intra-attention, allows input features to be the criteria for the attention itself (Vaswani et al., 2017). We obtain self-attention scores using graph convolution. For instance, if the graph convolution formula of Kipf & Welling is used, the self-attention score $Z \in \mathbb{R}^{N\times 1}$ is calculated as follows.

$$Z = \sigma(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}X\Theta_{att}) \tag{3}$$

where σ is the activation function (e.g. $tanh$), $\tilde{A} \in \mathbb{R}^{N\times N}$ is the adjacency matrix with self-connections (i.e. $\tilde{A} = A+I_N$), $\tilde{D} \in \mathbb{R}^{N\times N}$ is the degree matrix of $\tilde{A}$, $X \in \mathbb{R}^{N\times F}$ is the input features of the graph with N nodes and F-dimensional features, and $\Theta_{att} \in \mathbb{R}^{F\times 1}$ is the only parameter of the SAGPool layer. By utilizing graph convolution to obtain self-attention scores, the result of the pooling is based on both graph features and topology. We adopt the node selection method of Gao & Ji; Cangea et al., which retains a portion of nodes of the input graph even when graphs of varying sizes and structures are inputted. The pooling ratio $k \in (0, 1]$ is a hyperparameter that determines the number of nodes to keep. The top $\lceil kN \rceil$ nodes are selected based on the value of Z.

$$\text{idx} = \text{top-rank}(Z, \lceil kN \rceil), \quad Z_{mask} = Z_{\text{idx}} \tag{4}$$

where top-rank is the function that returns the indices of the top $\lceil kN \rceil$ values, $\cdot_{\text{idx}}$ is an indexing operation and Z_{mask} is the feature attention mask.

Graph pooling An input graph is processed by the operation notated as **masking** in Figure 1.

$$X' = X_{\text{idx},:}, \; X_{out} = X' \odot Z_{mask}, \; A_{out} = A_{\text{idx,idx}} \tag{5}$$

where $X_{\text{idx},:}$ is the row-wise (i.e. node-wise) indexed feature matrix, $\odot$ is the broadcasted elementwise product, and $A_{\text{idx,idx}}$ is the row-wise and col-wise indexed adjacency matrix. X_{out} and A_{out} are the new feature matrix and the corresponding adjacency matrix, respectively.

Variation of SAGPool The main reason for using graph convolution in SAGPool is to reflect the topology as well as node features. The various formulas of GNNs can be substituted for Equation (3), if GNNs take the node feature and the adjacency matrix as inputs. The generalized equation for calculating the attention score $Z \in \mathbb{R}^{N\times 1}$ is as follows.

$$Z = \sigma(\text{GNN}(X, A)) \tag{6}$$

where X denotes the node feature matrix and A is the adjacency matrix.

There are several ways to calculate attention scores using not only adjacent nodes but also multi-hop connected nodes. In Equation (7) and (8), we illustrate examples of using the two-hop connections which involve the augmentation of edges and the stack of GNN layers. Adding the square of an adjacency matrix creates edges between two-hop neighbors.

$$Z = \sigma(\text{GNN}(X, A + A^2)) \tag{7}$$

The stack of GNN layers allows for the indirect aggregation of two-hop nodes. In this case, the nonlinearity and the number of parameters of the SAGPool layer increase.

$$Z = \sigma(\text{GNN}_2(\sigma(\text{GNN}_1(X, A)), A)) \tag{8}$$

Equations (7) and (8) can be applied to the multi-hop connections.

Another variant is to average multiple attention scores. The average attention score is obtained by M GNNs as follows:

$$Z = \frac{1}{M}\sum_m \sigma(\text{GNN}_m(X, A)) \tag{9}$$

In this paper, the models using Equation (7), (8), and (9) are referred to as $\text{SAGPool}_{\text{augmentation}}$, $\text{SAGPool}_{\text{serial}}$, and $\text{SAGPool}_{\text{parallel}}$, respectively.

3.2. Model Architecture

According to Lipton & Steinhardt, if numerous modifications are made to a model, it may be difficult to identify which modification contributes to improving performance. For a fair comparison, we adopted the model architectures from Zhang et al. and Cangea et al., and compared the baselines and our method using the same architectures.

Convolution layer As mentioned in Section 2.1, there are many definitions for graph convolution. Other types of graph convolution may improve performance, but we utilize the widely used graph convolution proposed by Kipf & Welling for all the models. Equation (10) is the same as Equation (3), except for the dimension of Θ.

$$h^{(l+1)} = \sigma(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}h^{(l)}\Theta) \tag{10}$$

where $h^{(l)}$ is the node representation of l-th layer and $\Theta \in \mathbb{R}^{F\times F'}$ is the convolution weight with input feature dimension F and output feature dimension F'. The Rectified Linear Unit (ReLU) (Nair & Hinton, 2010) function is used as an activation function.

Readout layer Inspired by the JK-net architecture (Xu et al., 2018b), Cangea et al. proposed a readout layer that aggregates node features to make a fixed size representation. The summarized output feature of the readout layer is as follows:

$$s = \frac{1}{N}\sum_{i=1}^{N} x_i \,||\, \max_{i=1}^{N} x_i \tag{11}$$

where N is the number of nodes, x_i is the feature vector of i-th node, and $||$ denotes concatenation.

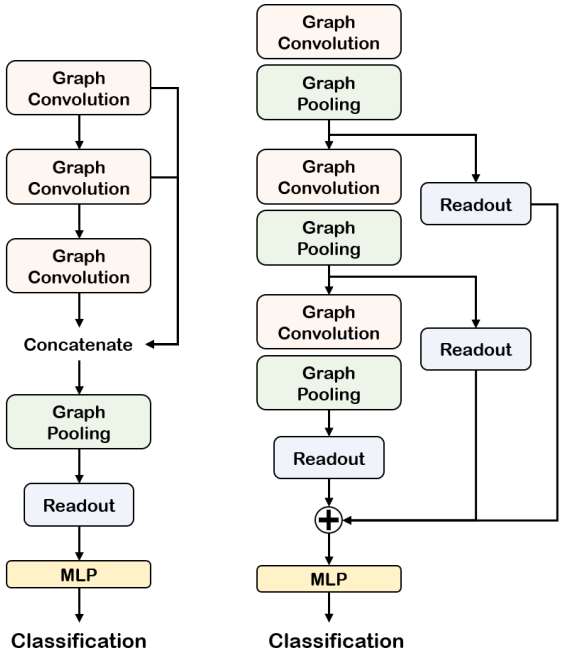

Figure 2. The global pooling architecture (left) and the hierarchical pooling architecture (right). These architectures are applied to all the baselines and SAGPool for a fair comparison. In this paper, the architecture on the left side is referred to as $POOL_g$ and the architecture on the right side is referred to as $POOL_h$ with the $POOL$ method (e.g. SAGPool_g, gPool_h).

Global pooling architecture We implemented the global pooling architecture proposed by Zhang et al.. As shown in Figure 2, the global pooling architecture consists of three graph convolutional layers and the outputs of each layer are concatenated. Node features are aggregated in the readout layer which follows the pooling layer. Then graph feature representations are passed to the linear layer for classification.

Hierarchical pooling architecture In this setting, we implemented the hierarchical pooling architecture from the recent hierarchical pooling study of Cangea et al.. As shown in Figure 2, the architecture is comprised of three blocks each of which consists of a graph convolutional layer and a graph pooling layer. The outputs of each block are summarized in the readout layer. The summation of the outputs of each readout layer is fed to the linear layer for classifica-

Table 1. Statistics of data sets.

Data set	Number of Graphs	Number of Classes	Avg. # of Nodes per Graph	Avg. # of Edges per Graph
D&D	1178	2	284.32	715.66
PROTEINS	1113	2	39.06	72.82
NCI1	4110	2	29.87	32.30
NCI109	4127	2	29.68	32.13
FRANKENSTEIN	4337	2	16.90	17.88

Table 2. The grid search space for the hyperparameters. The pooling ratio is used only for the hierarchical pooling architecture because the the global pooling architecture uses the same node selection strategy as SortPool. The node selection strategy of SortPool does not require the pooling ratio.

Hyperparameter	Range
Learning rate	1e-2, 5e-2, 1e-3, 5e-3, 1e-4, 5e-4
Hidden size	16, 32, 64, 128
Weight decay (L2 regularization)	1e-2, 1e-3, 1e-4, 1e-5
Pooling ratio	1/2, 1/4

tion.

4. Experiments

We evaluate the global pooling and hierarchical pooling methods on the graph classification task. In Section 4.1, we discuss the datasets used for evaluation. Section 4.3 describes how we train the models. The methods compared in the experiments are introduced in Sections 4.4 and 4.5.

4.1. Datasets

Five datasets with a large number of graphs ($>$ 1k) were selected among the benchmark datasets (Kersting et al., 2016). The statistics of the datasets are summarized in Table 1.

D&D (Dobson & Doig, 2003; Shervashidze et al., 2011) contains graphs of protein structures. A node represents an amino acid and edges are constructed if the distance of two nodes is less than 6 Å. A label denotes whether a protein is an enzyme or non-enzyme. **PROTEINS** (Dobson & Doig, 2003; Borgwardt et al., 2005) is also a set of proteins, where nodes are secondary structure elements. If nodes have edges, the nodes are in an amino acid sequence or in a close 3D space. **NCI** (Wale et al., 2008) is a biological dataset used for anticancer activity classification. In the dataset, each graph represents a chemical compound, with nodes and edges representing atoms and chemical bonds, respectively. **NCI1** and **NCI109** are commonly used as benchmark datasets for graph classification. **FRANKENSTEIN** (Orsini et al., 2015) is a set of molecular graphs (Costa & Grave, 2010) with node features containing continuous values. A label denotes whether a molecule is a mutagen or non-mutagen.

4.2. Evaluation of GNNs

In addition, the same early stopping criterion and hyperparameter selection strategy are used for all the models to ensure a fair comparison.

4.3. Training Procedures

Shchur et al. demonstrate that different splits of data can affect the performance of GNN models. In our experiments, we evaluated the pooling methods over 20 random seeds using 10-fold cross validation. A total of 200 testing results were used to obtain the final accuracy of each method on each dataset. 10 percent of the training data was used for validation in the training session. We used the Adam optimizer (Kingma & Ba, 2014), early stopping criterion, patience, and hyperparameter selection strategy for the global pooling architecture and hierarchical pooling architecture. We stopped the training if the validation loss did not improve for 50 epochs in an epoch termination condition with a maximum of 100k epochs, as done in (Shchur et al., 2018). The optimal hyperparameters are obtained by grid search. The ranges of grid search are summarized in Table 2.

4.4. Baselines

We consider the following four pooling methods as baselines: Set2Set, SortPool, DiffPool, and gPool. DiffPool, gPool, and SAGPool$_h$ were compared using the hierarchical pooling architecture while Set2Set, SortPool, and SAGPool$_g$ were compared using the global pooling architecture. We used the same hyperparameter search strategy for all the baselines and SAGPool. The hyperparameters are summarized in Table 2.

Set2Set (Vinyals et al., 2015) requires an additional hyperparameter which is the number of processing steps for the LSTM(Hochreiter & Schmidhuber, 1997) module. We use 10 processing steps for all the experiments. We assume that the readout layer is unnecessary because the LSTM module

Table 3. Average accuracy and standard deviation of the 20 random seeds. The subscript g (e.g. $POOL_g$) denotes the global pooling architecture and the subscript h (e.g. $POOL_h$) denotes the hierarchical pooling architecture.

Models	D&D	PROTEINS	NCI1	NCI109	FRANKENSTEIN
Set2Set$_g$	71.27 ± 0.84	66.06 ± 1.66	68.55 ± 1.92	69.78 ± 1.16	61.92 ± 0.73
SortPool$_g$	72.53 ± 1.19	66.72 ± 3.56	73.82 ± 0.96	74.02 ± 1.18	60.61 ± 0.77
SAGPool$_g$ (Ours)	$\mathbf{76.19} \pm 0.94$	$\mathbf{70.04} \pm 1.47$	$\mathbf{74.18} \pm 1.20$	$\mathbf{74.06} \pm 0.78$	$\mathbf{62.57} \pm 0.60$
DiffPool$_h$	66.95 ± 2.41	68.20 ± 2.02	62.32 ± 1.90	61.98 ± 1.98	60.60 ± 1.62
gPool$_h$	75.01 ± 0.86	71.10 ± 0.90	67.02 ± 2.25	66.12 ± 1.60	61.46 ± 0.84
SAGPool$_h$ (Ours)	$\mathbf{76.45} \pm 0.97$	$\mathbf{71.86} \pm 0.97$	$\mathbf{67.45} \pm 1.11$	$\mathbf{67.86} \pm 1.41$	$\mathbf{61.73} \pm 0.76$

Table 4. Experimental results of SAGPool$_h$ variants. We compare ChebConv(K=2) (Defferrard et al., 2016), GCNConv (Kipf & Welling, 2016), SAGEConv (Hamilton et al., 2017), and GATConv(heads=6) (Velikovi et al., 2018). GCNConv is applied to SAGPool$_h$, SAGPool$_{h,\text{augmentation}}$, SAGPool$_{h,\text{serial}}$, and SAGPool$_{h,\text{parallel}}$.

Graph Convolution	D&D	PROTEINS
SAGPool$_h$	76.45 ± 0.97	71.86 ± 0.97
SAGPool$_{h,\text{Cheb}}$	75.82 ± 0.79	71.98 ± 0.93
SAGPool$_{h,\text{SAGE}}$	76.28 ± 1.06	71.93 ± 0.82
SAGPool$_{h,\text{GAT}}$	75.49 ± 0.93	71.98 ± 1.01
SAGPool$_{h,\text{augmentation}}$	77.07 ± 0.82	71.82 ± 0.81
SAGPool$_{h,\text{serial,2layers}}$	76.68 ± 0.96	72.17 ± 0.87
SAGPool$_{h,\text{parallel},M=2}$	75.79 ± 0.96	72.05 ± 0.43
SAGPool$_{h,\text{parallel},M=4}$	76.77 ± 0.61	71.66 ± 0.98

produces embeddings for graphs invariant to the order of nodes.

SortPool (Zhang et al., 2018b) is a recent global pooling method which uses sorting for pooling. The K number of nodes is set such that 60% of graphs have more than K nodes. In the global pooling setting, SAGPool$_g$ has the same K number of output nodes as SortPool.

DiffPool (Ying et al., 2018) is the first end-to-end trainable graph pooling method that can produce hierarchical representations of graphs. We did not use batch normalization for DiffPool, which is not related to the pooling method. For the hyperparameter search, the pooling ratio ranges from 0.25 to 0.5 for the following reasons. In the reference implementation, the cluster size is set to 25% of the maximum number of nodes. DiffPool$_h$ causes the out of memory error when the pooling ratio is larger than 0.5.

gPool (Gao & Ji, 2019) selects top-ranked nodes for pooling, which makes it similar to our method. The comparison between our method and gPool demonstrates that considering topology can help improve performance on the graph classification task.

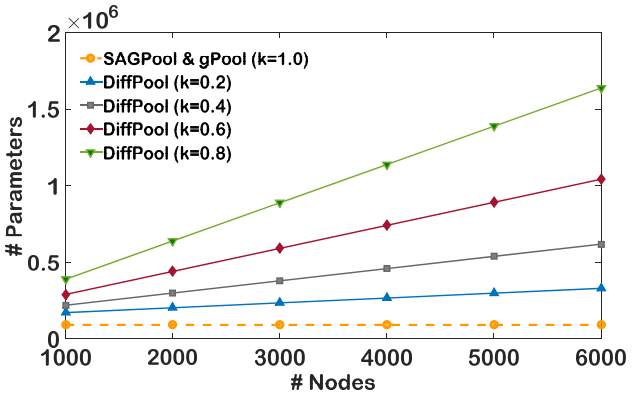

Figure 3. The increase in the number of parameters according to the number of graph nodes. The x-axis label denotes the number of input graph nodes and the y-axis label denotes the number of parameters of the hierarchical pooling models: the number of input node features is 128, the hidden feature size is 128, and the number of classes is 2. Equation (3) is used as a graph convolution of SAGPool. k denotes the pooling ratio and $k = 1.0$ indicates that the entire node is preserved after pooling. gPool and SAGPool have a consistent number of parameters regardless of the input graph size and the pooling ratio.

4.5. Variations of SAGPool

As mentioned in section 3.1, three variations of SAGPool are used to obtain attention scores Z. In our experiments, we compared each variant on the two datasets. First, any kind of GNNs can be applied to Equation (6). We compared the performance of the three most widely used GNNs (SAGPool$_{\text{Cheb}}$, SAGPool$_{\text{SAGE}}$, SAGPool$_{\text{GAT}}$). Second, we made the following modifications to SAGPool so that it can consider the two-hop connection: an edge augmentation (SAGPool$_{\text{augmentation}}$) in Equation (7) and a stack of GNN layers (SAGPool$_{\text{serial}}$) in Equation (8). Last, multiple GNNs calculate attention scores and the scores are averaged to obtain the final attention score (SAGPool$_{\text{parallel}}$). We evaluated the performance of $M = 2$ and $M = 4$ using Equation (9). The results are summarized in Table 4.

4.6. Summary of Results

The results are summarized in Table 3 and 4. The accuracies and standard deviations are given in percentages. From the comparison of the global pooling methods and SAGPool, the results demonstrate that SAGPool generally performs well, but it performs especially well on D&D and PROTEINS. In the experiments, SAGPool outperformed the hierarchical pooling methods on all the datasets. We also compared variants of SAGPool with the hierarchical pooling architecture on the two benchmark datasets. The performance of the variants of SAGPool varied. The experimental results of the SAGPool variants show that SAGPool has the potential to improve performance. A detailed analysis of the experimental results is provided in the next section.

5. Analysis

In this section, we provide an analysis of the experimental results. In Section 5.1, we compare global pooling and hierarchical pooling. Section 5.2 provides an explanation on how the SAGPool method addresses the shortcomings of the gPool method. In the 5.3 and 5.4 sections, we compare the efficiency of SAGPool with that of DiffPool. We provide an analysis of SAGPool variants in Section 5.5.

5.1. Global and Hierarchical Pooling

It is difficult to determine whether the global pooling architecture or hierarchical pooling architecture is completely beneficial to graph classification. Since the global pooling architecture $POOL_g$ (SAGPool$_g$, SortPool$_g$, Set2Set$_g$) minimizes the loss of information, it performs better than the hierarchical pooling architecture $POOL_h$ (SAGPool$_h$, gPool$_h$, DiffPool$_h$) on datasets with fewer nodes (NCI1, NCI109, FRANKENSTEIN). However, $POOL_h$ is more effective on datasets with a large number of nodes (D&D, PROTEINS) because it efficiently extracts useful information from large scale graphs. Therefore, it is important to use the pooling architecture that is the most suitable for the given data. Nonetheless, SAGPool tends to perform well with each architecture.

5.2. Effect of Considering Graph Topology

To calculate the attention scores of nodes, SAGPool$_h$ utilizes the graph convolution in Equation (3). Unlike gPool, SAGPool uses the $\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$ term, which is the first order approximation of the graph Laplacian. This term allows SAGPool to consider graph topology. As shown in Table 3, considering graph topology improves performance. In addition, the graph Laplacian does not have to be recalculated because it is the term used in a previous graph convolutional layer in the same block. Although SAGPool has the same parameters as gPool (Figure 3), it achieves superior performance in the graph classification task.

5.3. Sparse Implementation

Manipulating graph data with a sparse matrix is important for GNNs because the adjacency matrix is usually sparse. When graph convolution is calculated using a dense matrix, the computational complexity of multiplication AX is $\mathcal{O}(|V|^2)$ where A is the adjacency matrix, X is the feature matrix of nodes, and V denotes vertices. Pooling with a dense matrix causes the memory efficiency problem, as mentioned by (Cangea et al., 2018). However, if a sparse matrix is used in the same operation, the computational complexity is reduced to $\mathcal{O}(|E|)$ where E represents the edges. Since SAGPool is a sparse pooling method, it can reduce its computational complexity, unlike DiffPool which is a dense pooling method. Sparseness also affects space complexity. Since SAGPool uses GNN for obtaining attention scores, SAGPool requires $\mathcal{O}(|V|+|E|)$ of storage for sparse pooling whereas dense pooling methods need $\mathcal{O}(|V|^2)$.

5.4. Relation with the Number of Nodes

In DiffPool, the cluster size has to be defined when constructing a model because a GNN produces an assignment matrix S as stated in Equation (1). The cluster size has to be proportional to the maximum number of nodes according to the reference implementation. These requirements of DiffPool can lead to two problems. First, the number of parameters is dependent on the maximum number of nodes as shown in Figure 3. Second, it is difficult to determine the right cluster size when the number of nodes varies greatly. For example, only 10 out of 1178 graphs have over 1000 nodes, where the maximum number of nodes is 5748 and the minimum is 30. The cluster size is 574 if the pooling ratio is 10%, which expands the size of graphs after pooling for most of the data. On the other hand, in SAGPool, the number of parameters is independent of the cluster size. In addition, the cluster size can be changed based on the number of input nodes.

5.5. Comparison of the SAGPool Variants

To investigate the potential of our method, we evaluated SAGPool variants on two datasets. SAGPool can be modified to perform the following: changing the type of GNN, considering the two-hop connections, and averaging the attention scores of multiple GNNs. As shown in Table 4, the performance on graph classification varies depending on which dataset and type of GNN in SAGPool are used. We used two techniques to consider two-hop connections. The attention scores obtained by the two sequential GNN layers (SAGPool$_{\text{serial}}$) reflect the information of two-hop neighbors. Another technique is to add the square of an adjacency matrix to itself, resulting in a new adjacency matrix

that has two-hop connectivity. Without any modifications to the SAGPool layer, the new adjacency matrix can be processed in $\text{SAGPool}_{\text{augmentation}}$. The information of two-hop neighbors may help improve performance. The last variants of SAGPool is to average the attention scores from multiple GNNs. We found that choosing the right M for the dataset can help achieve stable performance.

5.6. Limitations

We retain a certain percentage (pooling ratio k) of nodes to handle different input graphs of various sizes, which has also been done in previous studies (Gao & Ji, 2019; Cangea et al., 2018). In SAGPool, we cannot parameterize the pooling ratios to find optimal values for each graph. To address this limitation, we used binary classification to decide which nodes to preserve, but this did not completely solve the issue.

6. Conclusion

In this paper, we proposed SAGPool which is a novel graph pooling method based on self-attention. Our method has the following features: hierarchical pooling, consideration of both node features and graph topology, reasonable complexity, and end-to-end representation learning. SAGPool uses a consistent number of parameters regardless of the input graph size. Extensions of our work may include using learnable pooling ratios to obtain optimal cluster sizes for each graph and studying the effects of multiple attention masks in each pooling layer, where final representations can be derived by aggregating different hierarchical representations. Our experiments were run on a NVIDIA TitanXp GPU. We implemented all the baselines and SAGPool using PyTorch (Paszke et al., 2017) and the geometric deep learning extension library provided by Fey et al..

References

Bahdanau, D., Cho, K., and Bengio, Y. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2014. URL http://arxiv.org/abs/1409.0473.

Bastings, J., Titov, I., Aziz, W., Marcheggiani, D., and Simaan, K. Graph convolutional encoders for syntax-aware neural machine translation. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pp. 1957–1967, 2017.

Borgwardt, K. M., Ong, C. S., Schönauer, S., Vishwanathan, S., Smola, A. J., and Kriegel, H.-P. Protein function prediction via graph kernels. *Bioinformatics*, 21(suppl_1): i47–i56, 2005.

Bronstein, M. M., Bruna, J., LeCun, Y., Szlam, A., and Vandergheynst, P. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4): 18–42, 2017.

Bruna, J., Zaremba, W., Szlam, A., and Lecun, Y. Spectral networks and locally connected networks on graphs. In *International Conference on Learning Representations (ICLR2014), CBLS, April 2014*, 2014.

Cangea, C., Veličković, P., Jovanović, N., Kipf, T., and Liò, P. Towards sparse hierarchical graph classifiers. *arXiv preprint arXiv:1811.01287*, 2018.

Cheng, J., Dong, L., and Lapata, M. Long short-term memory-networks for machine reading. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 551–561, 2016.

Costa, F. and Grave, K. D. Fast neighborhood subgraph pairwise distance kernel. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, pp. 255–262. Omnipress, 2010.

Dai, H., Dai, B., and Song, L. Discriminative embeddings of latent variable models for structured data. In *International Conference on Machine Learning*, pp. 2702–2711, 2016.

Davidson, E. H., Rast, J. P., Oliveri, P., Ransick, A., Calestani, C., Yuh, C.-H., Minokawa, T., Amore, G., Hinman, V., Arenas-Mena, C., et al. A genomic regulatory network for development. *science*, 295(5560): 1669–1678, 2002.

Defferrard, M., Bresson, X., and Vandergheynst, P. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*, pp. 3844–3852, 2016.

Dhillon, I. S., Guan, Y., and Kulis, B. Weighted graph cuts without eigenvectors a multilevel approach. *IEEE transactions on pattern analysis and machine intelligence*, 29(11), 2007.

Dobson, P. D. and Doig, A. J. Distinguishing enzyme structures from non-enzymes without alignments. *Journal of molecular biology*, 330(4):771–783, 2003.

Duvenaud, D. K., Maclaurin, D., Iparraguirre, J., Bombarell, R., Hirzel, T., Aspuru-Guzik, A., and Adams, R. P. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in neural information processing systems*, pp. 2224–2232, 2015.

Fey, M., Lenssen, J. E., Weichert, F., and Müller, H. SplineCNN: Fast geometric deep learning with continuous B-spline kernels. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

Gao, H. and Ji, S. Graph u-net, 2019. URL https://openreview.net/forum?id=HJePRoAct7.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. Neural message passing for quantum chemistry. *CoRR*, abs/1704.01212, 2017a. URL http://arxiv.org/abs/1704.01212.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. Neural message passing for quantum chemistry. In *International Conference on Machine Learning*, pp. 1263–1272, 2017b.

Hamilton, W., Ying, Z., and Leskovec, J. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pp. 1024–1034, 2017.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Henaff, M., Bruna, J., and LeCun, Y. Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163*, 2015.

Hinton, G., Deng, L., Yu, D., Dahl, G. E., Mohamed, A.-r., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T. N., et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal processing magazine*, 29(6):82–97, 2012.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Karpathy, A., Toderici, G., Shetty, S., Leung, T., Sukthankar, R., and Fei-Fei, L. Large-scale video classification with convolutional neural networks. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 1725–1732, 2014.

Kersting, K., Kriege, N. M., Morris, C., Mutzel, P., and Neumann, M. Benchmark data sets for graph kernels, 2016. URL http://graphkernels.cs.tu-dortmund.de.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL http://arxiv.org/abs/1412.6980.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105, 2012.

Lazer, D., Pentland, A. S., Adamic, L., Aral, S., Barabasi, A. L., Brewer, D., Christakis, N., Contractor, N., Fowler, J., Gutmann, M., et al. Life in the network: the coming age of computational social science. *Science (New York, NY)*, 323(5915):721, 2009.

LeCun, Y., Bengio, Y., and Hinton, G. Deep learning. *nature*, 521(7553):436, 2015.

Lipton, Z. C. and Steinhardt, J. Troubling trends in machine learning scholarship. *arXiv preprint arXiv:1807.03341*, 2018.

Monti, F., Boscaini, D., Masci, J., Rodola, E., Svoboda, J., and Bronstein, M. M. Geometric deep learning on graphs and manifolds using mixture model cnns. In *Proc. CVPR*, volume 1, pp. 3, 2017.

Morris, C., Ritzert, M., Fey, M., Hamilton, W. L., Lenssen, J. E., Rattan, G., and Grohe, M. Weisfeiler and leman go neural: Higher-order graph neural networks. *CoRR*, abs/1810.02244, 2018. URL http://arxiv.org/abs/1810.02244.

Nair, V. and Hinton, G. E. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814, 2010.

Orsini, F., Frasconi, P., and De Raedt, L. Graph invariant kernels. In *Proceedings of the Twenty-fourth International Joint Conference on Artificial Intelligence*, pp. 3756–3762, 2015.

Parikh, A., Täckström, O., Das, D., and Uszkoreit, J. A decomposable attention model for natural language inference. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 2249–2255, 2016.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

Peng, H., Li, J., He, Y., Liu, Y., Bao, M., Wang, L., Song, Y., and Yang, Q. Large-scale hierarchical text classification with recursively regularized deep graph-cnn. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web*, pp. 1063–1072. International World Wide Web Conferences Steering Committee, 2018.

Rhee, S., Seo, S., and Kim, S. Hybrid approach of relation network and localized graph convolutional filtering for breast cancer subtype classification. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pp. 3527–3534. International Joint Conferences on Artificial Intelligence Organization,

7 2018. doi: 10.24963/ijcai.2018/490. URL https://doi.org/10.24963/ijcai.2018/490.

Shchur, O., Mumme, M., Bojchevski, A., and Günnemann, S. Pitfalls of graph neural network evaluation. *CoRR*, abs/1811.05868, 2018. URL http://arxiv.org/abs/1811.05868.

Shervashidze, N., Schweitzer, P., Leeuwen, E. J. v., Mehlhorn, K., and Borgwardt, K. M. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(Sep):2539–2561, 2011.

Simoncelli, E. P. and Olshausen, B. A. Natural image statistics and neural representation. *Annual review of neuroscience*, 24(1):1193–1216, 2001.

van den Berg, R., Kipf, T. N., and Welling, M. Graph convolutional matrix completion. *stat*, 1050:7, 2017.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In *Advances in Neural Information Processing Systems*, pp. 5998–6008, 2017.

Velikovi, P., Cucurull, G., Casanova, A., Romero, A., Li, P., and Bengio, Y. Graph attention networks. In *International Conference on Learning Representations*, 2018.

Vinyals, O., Bengio, S., and Kudlur, M. Order matters: Sequence to sequence for sets. *arXiv preprint arXiv:1511.06391*, 2015.

Wale, N., Watson, I. A., and Karypis, G. Comparison of descriptor spaces for chemical compound retrieval and classification. *Knowledge and Information Systems*, 14 (3):347–375, 2008.

Xu, K., Li, C., Tian, Y., Sonobe, T., ichi Kawarabayashi, K., and Jegelka, S. Representation learning on graphs with jumping knowledge networks. In *ICML*, 2018a.

Xu, K., Li, C., Tian, Y., Sonobe, T., Kawarabayashi, K.-i., and Jegelka, S. Representation learning on graphs with jumping knowledge networks. *arXiv preprint arXiv:1806.03536*, 2018b.

Yao, K.-L. and Li, W.-J. Convolutional geometric matrix completion. *arXiv preprint arXiv:1803.00754*, 2018.

Yao, L., Mao, C., and Luo, Y. Graph convolutional networks for text classification. *arXiv preprint arXiv:1809.05679*, 2018.

Ying, R., You, J., Morris, C., Ren, X., Hamilton, W. L., and Leskovec, J. Hierarchical graph representation learning with differentiable pooling. *CoRR*, abs/1806.08804, 2018. URL http://arxiv.org/abs/1806.08804.

You, J., Liu, B., Ying, Z., Pande, V., and Leskovec, J. Graph convolutional policy network for goal-directed molecular graph generation. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 31*, pp. 6412–6422. Curran Associates, Inc., 2018.

Zhang, H., Goodfellow, I., Metaxas, D., and Odena, A. Self-attention generative adversarial networks. *arXiv preprint arXiv:1805.08318*, 2018a.

Zhang, M., Cui, Z., Neumann, M., and Chen, Y. An end-to-end deep learning architecture for graph classification. In *Proceedings of AAAI Conference on Artificial Inteligence*, 2018b.

Zhou, J., Cui, G., Zhang, Z., Yang, C., Liu, Z., and Sun, M. Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434*, 2018.

Zitnik, M., Agrawal, M., and Leskovec, J. Modeling polypharmacy side effects with graph convolutional networks. *Bioinformatics*, 34(13):457466, 2018.