ASAP: Adaptive Structure Aware Pooling for Learning Hierarchical Graph Representations

Ekagra Ranjan[1]*, **Soumya Sanyal**[2], **Partha Talukdar**[2]
[1]Indian Institute of Technology, Guwahati
[2]Indian Institute of Science, Bangalore
ekagra.ranjan@gmail.com, {soumyasanyal, ppt}@iisc.ac.in

Abstract

Graph Neural Networks (GNN) have been shown to work effectively for modeling graph structured data to solve tasks such as node classification, link prediction and graph classification. There has been some recent progress in defining the notion of pooling in graphs whereby the model tries to generate a graph level representation by downsampling and summarizing the information present in the nodes. Existing pooling methods either fail to effectively capture the graph substructure or do not easily scale to large graphs. In this work, we propose ASAP (Adaptive Structure Aware Pooling), a sparse and differentiable pooling method that addresses the limitations of previous graph pooling architectures. ASAP utilizes a novel self-attention network along with a modified GNN formulation to capture the importance of each node in a given graph. It also learns a sparse soft cluster assignment for nodes at each layer to effectively pool the subgraphs to form the pooled graph. Through extensive experiments on multiple datasets and theoretical analysis, we motivate our choice of the components used in ASAP. Our experimental results show that combining existing GNN architectures with ASAP leads to state-of-the-art results on multiple graph classification benchmarks. ASAP has an average improvement of 4%, compared to current sparse hierarchical state-of-the-art method.

1 Introduction

In recent years, there has been an increasing interest in developing Graph Neural Networks (GNNs) for graph structured data. CNNs have shown to be successful in tasks involving images (Krizhevsky, Sutskever, and Hinton 2012; He et al. 2016) and text (Kim 2014). Unlike these regular grid data, arbitrary shaped graphs have rich information present in their graph structure. By inherently capturing such information through message propagation along the edges of the graph, GNNs have proved to be more effective for graphs (Gilmer et al. 2017; Hamilton, Ying, and Leskovec 2017). GNNs have been successfully applied in tasks such as semantic role labeling (Marcheggiani and Titov 2017), relation extraction (Vashishth et al. 2018b), neural machine translation (Bastings et al. 2017), document dating (Vashishth et al. 2018a), and molecular feature extraction (Kearnes et al. 2016). While some of the works focus on learning node-level representations to perform tasks such as node classification (Kipf and Welling 2017; Veličković et al. 2017) and link prediction (Schlichtkrull et al. 2017; Vashishth et al. 2019), others focus on learning graph-level representations for tasks like graph classification (Bruna et al. 2013; Henaff, Bruna, and LeCun 2015; Ying et al. 2018; Gao and Ji 2019; Lee, Lee, and Kang 2019) and graph regression (Xie and Grossman 2018; Sanyal et al. 2018). In this paper, we focus on graph-level representation learning for the task of graph classification.

Briefly, the task of graph classification involves predicting the label of an input graph by utilizing the given graph structure and initial node-level representations. For example, given a molecule, the task could be to predict if it is toxic. Current GNNs are inherently *flat* and lack the capability of aggregating node information in a *hierarchical* manner. Such architectures rely on learning node representations through some GNN followed by aggregation of the node information to generate the graph representation (Vinyals, Bengio, and Kudlur 2016; Li et al. 2016; Zhang et al. 2018). But learning graph representations in a hierarchical manner is important to capture local substructures that are present in graphs. For example, in an organic molecule, a set of atoms together can act as a functional group and play a vital role in determining the class of the graph.

To address this limitation, new pooling architectures have been proposed where sets of nodes are recursively aggregated to form a cluster that represents a node in the pooled graph, thus enabling hierarchical learning. DiffPool (Ying et al. 2018) is a differentiable pooling operator that learns a soft assignment matrix mapping each node to a set of clusters. Since this assignment matrix is *dense*, it is not easily scalable to large graphs (Cangea et al. 2018). Following that, TopK (Gao and Ji 2019) is proposed which learns a scalar projection score for each node and selects the top k nodes. They address the sparsity concerns of DiffPool but are unable to capture the rich graph structure effectively. Recently, SAG-

*Research done during internship at Indian Institute of Science, Bangalore.

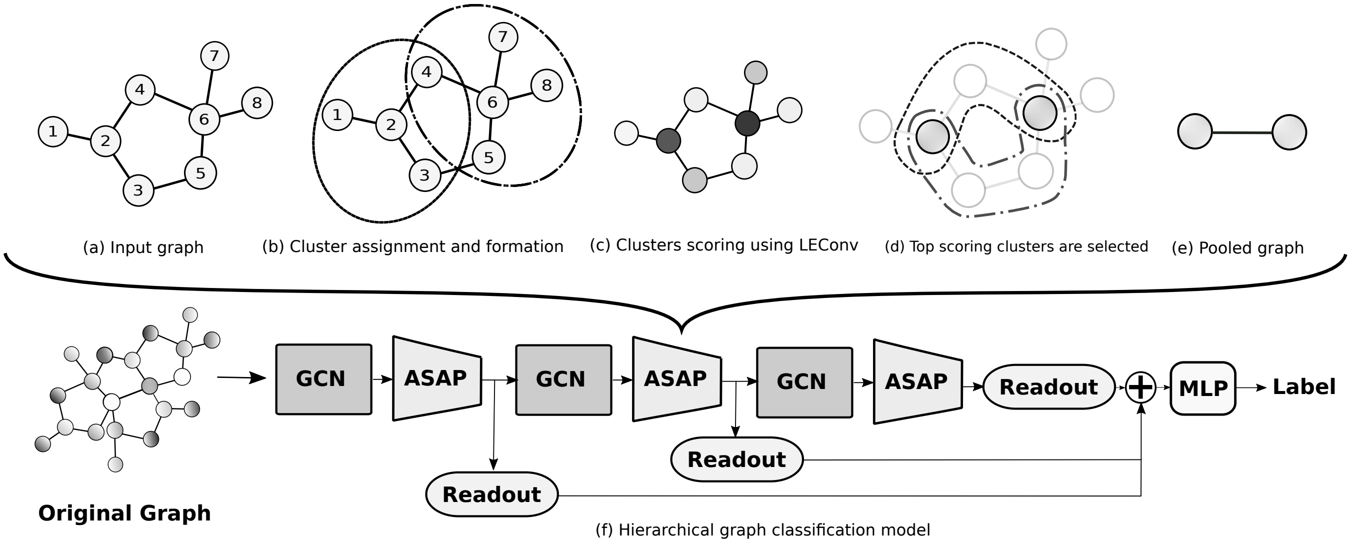

1
2
3
4
5
6
7
8
(a) Input graph
(b) Cluster assignment and formation
(c) Clusters scoring using LEConv
(d) Top scoring clusters are selected
(e) Pooled graph
Original Graph
GCN
ASAP
GCN
ASAP
GCN
ASAP
Readout
Readout
Readout
MLP
Label
(f) Hierarchical graph classification model

Figure 1: Overview of ASAP: (a) Input graph to ASAP. (b) ASAP initially clusters 1-hop neighborhood considering all nodes as medoid[1]. For brevity, we only show cluster formations of nodes 2 & 6 as medoids. Cluster membership is computed using M2T attention (refer Sec. 4.2). (c) Clusters are scored using LEConv (refer Sec. 4.3). Darker shade denotes higher score. (d) A fraction of top scoring clusters are selected in the pooled graph. Adjacency matrix is recomputed using edge weights between the member nodes of selected clusters. (e) Output of ASAP (f) Overview of hierarchical graph classification architecture.

Pool (Lee, Lee, and Kang 2019), a TopK based architecture, has been proposed which leverages self-attention network to learn the node scores. Although local graph structure is used for scoring nodes, it is still not used effectively in determining the connectivity of the pooled graph. Pooling methods that leverage the graph structure effectively while maintaining sparsity currently don't exist. We address the gap in this paper.

In this work, we propose a new sparse pooling operator called Adaptive Structure Aware Pooling (ASAP) which overcomes the limitations in current pooling methods. Our contributions can be summarized as follows:

- We introduce ASAP, a sparse pooling operator capable of capturing local subgraph information hierarchically to learn global features with better edge connectivity in the pooled graph.
- We propose Master2Token (M2T), a new self-attention framework which is better suited for global tasks like pooling.
- We introduce a new convolution operator LEConv, that can adaptively learn functions of local extremas in a graph substrucutre.

2 Related Work

2.1 Graph Neural Networks

Various formulation of GNNs have been proposed which use both spectral and non-spectral approaches. Spectral methods (Bruna et al. 2013; Henaff, Bruna, and LeCun 2015) aim at defining convolution operation using Fourier transformation and graph Laplacian. These methods do not directly generalize to graphs with different structure (Bronstein et al. 2017). Non-spectral methods (Defferrard, Bresson, and Vandergheynst 2016; Kipf and Welling 2017; Xu et al. 2018; Monti et al. 2017; Morris et al. 2018) define convolution through a local neighborhood around nodes in the graph. They are faster than spectral methods and easily generalize to other graphs. GNNs can also be viewed as *message passing* algorithm where nodes iteratively aggregate messages from neighboring nodes through edges (Gilmer et al. 2017).

2.2 Pooling

Pooling layers overcome GNN's inability to aggregate nodes hierarchically. Earlier pooling methods focused on deterministic graph clustering algorithms (Defferrard, Bresson, and Vandergheynst 2016; Fey et al. 2018; Simonovsky and Komodakis 2017). Ying et al. introduced the first differentiable pooling operator which out-performed the previous deterministic methods. Since then, new data-driven pooling methods have been proposed; both spectral (Ma et al. 2019; Dhillon, Guan, and Kulis 2007) and non-spectral (Ying et al. 2018; Gao and Ji 2019). Spectral methods aim at capturing the graph topology using eigen-decomposition algorithms. However, due to higher computational requirement for spectral graph techniques, they are not easily scalable to large graphs. Hence, we focus on non-spectral methods.

Pooling methods can further be divided into global and hierarchical pooling layers. Global pooling summarize the entire graph in just one step. Set2Set (Vinyals, Bengio, and Kudlur 2016) finds the importance of each node in the graph through iterative content-based attention. Global-Attention (Li et al. 2016) uses an attention mechanism to aggregate

[1] medoids are representatives of a cluster. They are similar to centroids but are strictly a member of the cluster.

nodes in the graph. SortPool (Zhang et al. 2018) summarizes the graph by concatenating few nodes after sorting them based on their features. Hierarchical pooling is used to capture the topological information of graphs. **DiffPool** forms a fixed number of clusters by aggregating nodes. It uses GNN to compute a dense soft assignment matrix, making it infeasible for large graphs. **TopK** scores nodes based on a learnable projection vector and samples a fraction of high scoring nodes. It avoids node aggregation and computing soft assignment matrix to maintain the sparsity in graph operations. **SAGPool** improve upon TopK by using a GNN to consider the graph structure while scoring nodes. Since TopK and SAGPool do not aggregate nodes nor compute soft edge weights, they are unable to preserve node and edge information effectively.

Property	**DiffPool**	**TopK**	**SAGPool**	**ASAP**
Sparse		✓	✓	✓
Node Aggregation	✓			✓
Soft Edge Weights	✓			✓
Variable number of clusters		✓	✓	✓

Table 1: Properties desired in hierarchical pooling methods.

To address these limitations, we propose ASAP, which has all the desirable properties of hierarchical pooling without compromising on sparsity in graph operations. Please see Table. 1 for an overall comparison of hierarchical pooling methods. Further comparison discussions between hierarchical architectures are presented in Sec. 8.1.

3 Preliminaries

3.1 Problem Statement

Consider a graph $G(\mathcal{V}, \mathcal{E}, X)$ with $N = |\mathcal{V}|$ nodes and $|\mathcal{E}|$ edges. Each node $v_i \in \mathcal{V}$ has d-dimensional feature representation denoted by x_i. $X \in \mathbb{R}^{N \times d}$ denotes the node feature matrix and $A \in \mathbb{R}^{N \times N}$ represents the weighted adjacency matrix. The graph G also has a label y associated with it. Given a dataset $D = \{(G_1, y_1), (G_2, y_2), ...\}$, the task of graph classification is to learn a mapping $f : \mathcal{G} \rightarrow \mathcal{Y}$, where $\mathcal{G}$ is the set of input graphs and $\mathcal{Y}$ is the set of labels associated with each graph. A pooled graph is denoted by $G^p(\mathcal{V}^p, \mathcal{E}^p, X^p)$ with node embedding matrix X^p and its adjacency matrix as A^p.

3.2 Graph Convolution Networks

We use Graph Convolution Network (GCN) (Kipf and Welling 2017) for extracting discriminative features for graph classification. GCN is defined as:

$$X^{(l+1)} = \sigma(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{\frac{1}{2}}X^{(l)}W^{(l)}), \tag{1}$$

where $\hat{A} = A + I$ for self-loops, $\hat{D} = \sum_j \hat{A}_{i,j}$ and $W^{(l)} \in \mathbb{R}^{d \times f}$ is a learnable matrix for any layer l. We use the initial node feature matrix wherever provided, i.e., $X^{(0)} = X$.

3.3 Self-Attention

Self-attention is used to find the dependency of an input on itself (Cheng, Dong, and Lapata 2016; Vaswani et al. 2017). An alignment score $\alpha_{i,j}$ is computed to map the importance of candidates c_j on target query q_i. In self-attention, target query q_i and candidates c_j are obtained from input entities $\boldsymbol{h} = \{h_1, ..., h_n\}$. Self-attention can be categorized as Token2Token and Source2Token based on the choice of target query q (Shen et al. 2018).

Token2Token (T2T) selects both the target and candidates from the input set $\boldsymbol{h}$. In the context of additive attention (Bahdanau, Cho, and Bengio 2014), $\alpha_{i,j}$ is computed as:

$$\alpha_{i,j} = softmax(\vec{v}^T\sigma(Wh_i \parallel Wh_j)). \tag{2}$$

where $||$ is the concatenation operator.

Source2Token (S2T) finds the importance of each candidate to a specific global task which cannot be represented by any single entity. $\alpha_{i,j}$ is computed by dropping the target query term. Eq. (2) changes to the following:

$$\alpha_{i,j} = softmax(\vec{v}^T\sigma(Wh_j)). \tag{3}$$

3.4 Receptive Field

We extend the concept of receptive field RF from pooling operations in CNN to GNN[2]. We define RF^{node} of a pooling operator as the number of hops needed to cover all the nodes in the neighborhood that influence the representation of a particular output node. Similarly, RF^{edge} of a pooling operator is defined as the number of hops needed to cover all the edges in the neighborhood that affect the representation of an edge in the pooled graph $\mathcal{G}^p$.

4 ASAP: Proposed Method

In this section we describe the components of our proposed method ASAP. As shown in Fig. 1(b), ASAP initially considers all possible local clusters with a fixed receptive field for a given input graph. It then computes the cluster membership of the nodes using an attention mechanism. These clusters are then scored using a GNN as depicted in Fig 1(c). Further, a fraction of the top scoring clusters are selected as nodes in the pooled graph and new edge weights are computed between neighboring clusters as shown in Fig. 1(d). Below, we discuss the working of ASAP in details. Please refer to Appendix Sec. I for a pseudo code of the working of ASAP.

4.1 Cluster Assignment

Initially, we consider each node v_i in the graph as a *medoid* of a cluster $c_h(v_i)$ such that each cluster can represent only the local neighbors $\mathcal{N}$ within a fixed radius of h hops i.e., $c_h(v_i) = \mathcal{N}_h(v_i)$. This effectively means that $RF^{node} = h$ for ASAP. This helps the clusters to effectively capture the information present in the graph sub-structure.

Let x_i^c be the feature representation of a cluster $c_h(v_i)$ centered at v_i. We define $G^c(\mathcal{V}, \mathcal{E}, X^c)$ as the graph with node

[2] Please refer to Appendix Sec. D for more details on similarity between pooling methods in CNN and ASAP.

feature matrix $X^c \in \mathbb{R}^{N \times d}$ and adjacency matrix $A^c = A$. We denote the cluster assignment matrix by $S \in \mathbb{R}^{N \times N}$, where $S_{i,j}$ represents the membership of node $v_i \in \mathcal{V}$ in cluster $c_h(v_j)$. By employing such local clustering (Schaeffer 2007), we can maintain sparsity of the cluster assignment matrix S similar to the original graph adjacency matrix A i.e., space complexity of both S and A is $\mathcal{O}(|\mathcal{E}|)$.

4.2 Cluster Formation using Master2Token

Given a cluster $c_h(v_i)$, we learn the cluster assignment matrix S through a self-attention mechanism. The task here is to learn the overall representation of the cluster $c_h(v_i)$ by attending to the relevant nodes in it. We observe that both T2T and S2T attention mechanisms described in Sec. 3.3 do not utilize any intra-cluster information. Hence, we propose a new variant of self-attention called **Master2Token (M2T)**. We further motivate the need for M2T framework later in Sec. 8.2. In M2T framework, we first create a master query $m_i \in \mathbb{R}^d$ which is representative of all the nodes within a cluster:

$$m_i = f_m(x'_j | v_j \in c_h(v_i)\}), \qquad (4)$$

where x'_j is obtained after passing x_j through a separate GCN to capture structural information in the cluster $c_h(v_i)$ [3]. f_m is a master function which combines and transforms feature representation of $v_j \in c_h(v_i)$ to find m_i. In this work we experiment with max master function defined as:

$$m_i = \max_{v_j \in c_h(v_i)} (x'_j). \qquad (5)$$

This master query m_i attends to all the constituent nodes $v_j \in c_h(v_i)$ using additive attention:

$$\alpha_{i,j} = softmax(\vec{w}^T \sigma(W m_i \parallel x'_j)). \qquad (6)$$

where $\vec{w}^T$ and W are learnable vector and matrix respectively. The calculated attention scores $\alpha_{i,j}$ signifies the membership strength of node v_j in cluster $c_h(v_i)$. Hence, we use this score to define the cluster assignment matrix discussed above, i.e., $S_{i,j} = \alpha_{i,j}$. The cluster representation x_i^c for $c_h(v_i)$ is computed as follows:

$$x_i^c = \sum_{j=1}^{|c_h(v_i)|} \alpha_{i,j} x_j. \qquad (7)$$

4.3 Cluster Selection using LEConv

Similar to TopK (Gao and Ji 2019), we sample clusters based on a cluster fitness score ϕ_i calculated for each cluster in the graph G^c using a fitness function f_ϕ. For a given pooling ratio $k \in (0, 1]$, the top $\lceil kN \rceil$ clusters are selected and included in the pooled graph G^p. To compute the fitness scores, we introduce **Local Extrema Convolution (LEConv)**, a graph convolution method which can capture local extremum information. In Sec. 5.1 we motivate the choice of LEConv's formulation and contrast it with the standard GCN formulation. LEConv is used to compute ϕ as follows:

$$\phi_i = \sigma(x_i^c W_1 + \sum_{j \in \mathcal{N}(i)} A^c_{i,j}(x_i^c W_2 - x_j^c W_3)) \qquad (8)$$

where $\mathcal{N}(i)$ denotes the neighborhood of the i^{th} node in G^c. W_1, W_2, W_3 are learnable parameters and $\sigma(.)$ is some activation function. Fitness vector $\Phi = [\phi_1, \phi_2, ..., \phi_N]^T$ is multiplied to the cluster feature matrix X^c to make f_ϕ learnable i.e.,:

$$\hat{X^c} = \Phi \odot X^c,$$

where $\odot$ is broadcasted hadamard product. The function TOP$_k(.)$ ranks the fitness scores and gives the indices $\hat{i}$ of top $\lceil kN \rceil$ selected clusters in G^c as follows:

$$\hat{i} = \text{TOP}_k(\hat{X^c}, \lceil kN \rceil).$$

The pooled graph G^p is formed by selecting these top $\lceil kN \rceil$ clusters. The pruned cluster assignment matrix $\hat{S} \in \mathbb{R}^{N \times \lceil kN \rceil}$ and the node feature matrix $X^p \in \mathbb{R}^{\lceil kN \rceil \times d}$ are given by:

$$\hat{S} = S(:, \hat{i}), \qquad X^p = \hat{X}^c(\hat{i}, :) \qquad (9)$$

where $\hat{i}$ is used for index slicing.

4.4 Maintaining Graph Connectivity

Following (Ying et al. 2018), once the clusters have been sampled, we find the new adjacency matrix A^p for the pooled graph G^p using $\hat{A}^c$ and $\hat{S}$ in the following manner:

$$A^p = \hat{S}^T \hat{A}^c \hat{S} \qquad (10)$$

where $\hat{A}^c = A^c + I$. Equivalently, we can see that $A^p_{i,j} = \sum_{k,l} \hat{S}_{k,i} \hat{A}^c_{k,l} \hat{S}_{l,j}$. This formulation ensures that any two clusters i and j in G^p are connected if there is any common node in the clusters $c_h(v_i)$ and $c_h(v_j)$ or if any of the constituent nodes in the clusters are neighbors in the original graph G (Fig. 1(d)). Hence, the strength of the connection between clusters is determined by both the membership of the constituent nodes through $\hat{S}$ and the edge weights A^c. Note that $\hat{S}$ is a sparse matrix by formulation and hence the above operation can be implemented efficiently.

5 Theoretical Analysis

5.1 Limitations of using GCN for scoring clusters

GCNs cannot learn to assign such a fitness score to a cluster which is a function of the local extremas of its constituent nodes. Scoring the clusters based on local extremas would potentially allow us to sample representative clusters from all parts of the graph. GCN from Eq. (1) can be viewed as an operator which first computes a *pre-score* $\hat{\phi}'$ for each node i.e., $\hat{\phi}' = XW$ followed by a weighted average over neighbors and a non-linearity. If for some node the pre-score is very high, it can increase the scores of its neighbors, inherently biasing the pooling operators to select a node in the local neighborhood instead of sampling clusters which represent the whole graph.

[3] If x_j is used as it is then interchanging any two nodes in a cluster will have not affect the final output, which is undesirable.

Method	D&D	PROTEINS	NCI1	NCI109	FRANKENSTEIN
Set2Set (Vinyals, Bengio, and Kudlur 2016)	71.60 ± 0.87	72.16 ± 0.43	66.97 ± 0.74	61.04 ± 2.69	61.46 ± 0.47
Global-Attention (Li et al. 2016)	71.38 ± 0.78	71.87 ± 0.60	69.00 ± 0.49	67.87 ± 0.40	61.31 ± 0.41
SortPool (Zhang et al. 2018)	71.87 ± 0.96	73.91 ± 0.72	68.74 ± 1.07	68.59 ± 0.67	63.44 ± 0.65
DiffPool (Ying et al. 2018)	66.95 ± 2.41	68.20 ± 2.02	62.32 ± 1.90	61.98 ± 1.98	60.60 ± 1.62
TopK (Gao and Ji 2019)	75.01 ± 0.86	71.10 ± 0.90	67.02 ± 2.25	66.12 ± 1.60	61.46 ± 0.84
SAGPool (Lee, Lee, and Kang 2019)	76.45 ± 0.97	71.86 ± 0.97	67.45 ± 1.11	67.86 ± 1.41	61.73 ± 0.76
ASAP (Ours)	$\mathbf{76.87 \pm 0.7}$	$\mathbf{74.19 \pm 0.79}$	$\mathbf{71.48 \pm 0.42}$	$\mathbf{70.07 \pm 0.55}$	$\mathbf{66.26 \pm 0.47}$

Table 2: Comparison of ASAP with previous global and hierarchical pooling. Average accuracy and standard deviation is reported for 20 random seeds. We observe that ASAP consistently outperforms all the baselines on all the datasets. Please refer to Sec. 7.1 for more details.

Theorem 1. *Let $\mathcal{G}$ be a graph with positive adjacency matrix A i.e., $A_{i,j} \geq 0$. Consider any function $f(X, A) : \mathbb{R}^{N \times d} \times \mathbb{R}^{N \times N} \rightarrow \mathbb{R}^{N \times 1}$ which depends on difference between a node and its neighbors after a linear transformation $W \in \mathbb{R}^{d \times 1}$. For e.g,:*

$$f_i = \sigma(\alpha_i x_i W + \sum_{j \in \mathcal{N}(i)} \beta_{i,j}(x_i W - x_j W))$$

where $f_i, \alpha_i, \beta_{i,j} \in \mathbb{R}$ and $x_i \in \mathbb{R}^d$.

a) If fitness value $\Phi = GCN(X, A)$ then Φ cannot learn f.
b) If fitness value $\Phi = LEConv(X, A)$ then Φ can learn f.

Proof. See Appendix Sec. F for proof. □

Motivated by the above analysis, we propose to use LEConv (Eq. 8) for scoring clusters. LEConv can learn to score clusters by considering both its global and local importance through the use of self-loops and ability to learn functions of local extremas.

5.2 Graph Connectivity

Here, we analyze ASAP from the aspect of edge connectivity in the pooled graph. When considering h-hop neighborhood for clustering, both ASAP and DiffPool have $RF^{edge} = 2h + 1$ because they use Eq. (10) to define the edge connectivity. On the other hand, both TopK and SAGPool have $RF^{edge} = h$. A larger edge receptive field implies that the pooled graph has better connectivity which is important for the flow of information in the subsequent GCN layers.

Theorem 2. *Let the input graph $\mathcal{G}$ be a tree of any possible structure with N nodes. Let k^* be the lower bound on sampling ratio k to ensure the existence of atleast one edge in the pooled graph irrespective of the structure of $\mathcal{G}$ and the location of the selected nodes. For TopK or SAGPool, $k^* \rightarrow 1$ whereas for ASAP, $k^* \rightarrow 0.5$ as $N \rightarrow \infty$.*

Proof. See Appendix Sec. G for proof. □

Theorem 2 suggests that ASAP can achieve a similar degree of connectivity as SAGPool or TopK for a much smaller sampling ratio k. For a tree with no prior information about its structure, ASAP would need to sample only half of the clusters whereas TopK and SAGPool would need to sample almost all the nodes, making TopK and SAGPool inefficient for such graphs. In general, independent of any combination of nodes selected, ASAP will have better connectivity due to its larger receptive field. Please refer to Appendix Sec. G for a similar analysis on path graph and more details.

5.3 Graph Permutation Equivariance

Proposition 1. *ASAP is a graph permutation equivariant pooling operator.*

Proof. See Appendix Sec. H for proof. □

6 Experimental Setup

In our experiments, we use 5 graph classification benchmarks and compare ASAP with multiple pooling methods. Below, we describe the statistics of the dataset, the baselines used for comparisons and our evaluation setup in detail.

6.1 Datasets

We demonstrate the effectiveness of our approach on 5 graph classification datasets. D&D (Shervashidze et al. 2011; Dobson and Doig 2003) and PROTEINS (Dobson and Doig 2003; Borgwardt et al. 2005) are datasets containing proteins as graphs. NCI1 (Wale, Watson, and Karypis 2008) and NCI109 are datasets for anticancer activity classification. FRANKENSTEIN (Orsini, Frasconi, and De Raedt 2015) contains molecules as graph for mutagen classification. Please refer to Table 3 for the dataset statistics.

Dataset	G_{avg}	C_{avg}	V_{avg}	E_{avg}
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 3: Statistics of the graph datasets. G_{avg}, C_{avg}, V_{avg} and E_{avg} denotes the average number of graphs, classes, nodes and edges respectively.

6.2 Baselines

We compare ASAP with previous state-of-the-art hierarchical pooling operators DiffPool (Ying et al. 2018), TopK (Gao and Ji 2019) and SAGPool (Lee, Lee, and Kang 2019). For comparison with global pooling, we choose Set2Set (Vinyals, Bengio, and Kudlur 2016), Global-Attention (Li et al. 2016) and SortPool (Zhang et al. 2018).

6.3 Training & Evaluation Setup

We use a similar architecture as defined in (Cangea et al. 2018; Lee, Lee, and Kang 2019) which is depicted in Fig. 1(f). For ASAP, we choose $k = 0.5$ and $h = 1$ to be consistent with baselines.[4] Following SAGPool(Lee, Lee, and Kang 2019), we conduct our experiments using 10-fold cross-validation and report the average accuracy on 20 random seeds.

Aggregation type	FITNESS	CLUSTER
None	-	-
Only cluster	-	✓
Both	✓	✓

Table 4: Different aggregation types as mentioned in Sec 7.2.

7 Results

In this section, we attempt to answer the following questions:

Q1 How does ASAP perform compared to other pooling methods at the task of graph classification? (Sec. 7.1)

Q2 Is cluster formation by M2T attention based node aggregation beneficial during pooling? (Sec. 7.3)

Q3 Is LEConv better suited as cluster fitness scoring function compared to vanilla GCN? (Sec. 7.4)

Q4 How helpful is the computation of inter-cluster soft edge weights instead of sampling edges from the input graph? (Sec. 7.5)

7.1 Performance Comparison

We compare the performace of ASAP with baseline methods on 5 graph classification tasks. The results are shown in Table 2. All the numbers for hierarchical pooling (DiffPool, TopK and SAGPool) are taken from (Lee, Lee, and Kang 2019). For global pooling (Set2Set, Global-Attention and SortPool), we modify the architectural setup to make them comparable with the hierarchical variants. [5]. We observe that ASAP consistently outperforms all the baselines on all 5 datasets. We note that ASAP has an average improvement of 4% and 3.5% over previous state-of-the-art hierarchical (SAGPool) and global (SortPool) pooling methods respectively. We also observe that compared to other hierarchical methods, ASAP has a smaller variance in performance which suggests that the training of ASAP is more stable.

7.2 Effect of Node Aggregation

Here, we evaluate the improvement in performance due to our proposed technique of aggregating nodes to form a cluster. There are two aspects involved during the creation of clusters for a pooled graph:

- FITNESS: calculating fitness scores for individual nodes. Scores can be calculated either by using only the medoid or by aggregating neighborhood information.
- CLUSTER: generating a representation for the new cluster node. Cluster representation can either be the medoid's representation or some feature aggregation of the neighborhood around the medoid.

We test three types of aggregation methods: 'None', 'Only cluster' and 'Both' as described in Table 4. As shown in Table 5, we observe that our proposed node aggregation helps improve the performance of ASAP.

Aggregation	FRANKENSTEIN	NCI1
None	67.4 ±0.6	69.9 ± 2.5
Only cluster	67.5 ±0.5	70.6 ± 1.8
Both	$\mathbf{67.8 \pm 0.6}$	$\mathbf{70.7 \pm 2.3}$

Table 5: Performace comparison of different aggregation methods on validation data of FRANKENSTEIN and NCI1.

Attention	FRANKENSTEIN	NCI1
T2T	67.6 ± 0.5	70.3 ± 2.0
S2T	67.7 ± 0.5	69.9 ± 2.0
M2T	$\mathbf{67.8 \pm 0.6}$	$\mathbf{70.7 \pm 2.3}$

Table 6: Effect of different attention framework on pooling evaluated on validation data of FRANKENSTEIN and NCI1. Please refer to Sec. 7.3 for more details.

7.3 Effect of M2T Attention

We compare our M2T attention framework with previously proposed S2T and T2T attention techniques. The results are shown in Table 6. We find that M2T attention is indeed better than the rest in NCI1 and comparable in FRANKENSTEIN.

7.4 Effect of LEConv as a fitness scoring function

In this section, we analyze the impact of LEConv as a fitness scoring function in ASAP. We use two baselines - GCN (Eq. 1) and Basic-LEConv which computes $\phi_i = \sigma(x_iW + \sum_{j\in\mathcal{N}(x_i)} A_{i,j}(x_iW - x_jW))$. In Table 7 we can see that Basic-LEConv and LEConv perform significantly

[4] Please refer to Appendix Sec. A for further details on hyperparameter tuning and Appendix Sec. E for ablation on k.

[5] Please refer to Appendix Sec. B for more details

Fitness function	FRANKENSTEIN	NCI1
GCN	62.7±0.3	65.4±2.5
Basic-LEConv	63.1±0.7	69.8±1.9
LEConv	**67.8±0.6**	**70.7±2.3**

Table 7: Performance comparison of different fitness scoring functions on validation data of FRANKENSTEIN and NCI1. Refer to Sec. 7.4 for details.

better than GCN because of their ability to model functions of local extremas. Further, we observe that LEConv performs better than Basic-LEConv as it has three different linear transformation compared to only one in the latter. This allows LEConv to potentially learn complicated scoring functions which is better suited for the final task. Hence, our analysis in Theorem 1 is emperically validated.

7.5 Effect of computing Soft edge weights

We evaluate the importance of calculating edge weights for the pooled graph as defined in Eq. 10. We use the best model configuration as found from above ablation analysis and then add the feature of computing soft edge weights for clusters. We observe a significant drop in performace when the edge weights are not computed. This proves the necessity of capturing the edge information while pooling graphs.

Soft edge weights	FRANKENSTEIN	NCI1
Absent	67.8 ± 0.6	70.7 ± 2.3
Present	$\mathbf{68.3 \pm 0.5}$	$\mathbf{73.4 \pm 0.4}$

Table 8: Effect of calculating soft edge weights on pooling for validation data of FRANKENSTEIN and NCI1. Please refer to Sec. 7.5 for more details.

8 Discussion

8.1 Comparison with other pooling methods

DiffPool DiffPool and ASAP both aggregate nodes to form a cluster. While ASAP only considers nodes which are within h-hop neighborhood from a node x_i (medoid) as a cluster, DiffPool considers the entire graph. As a result, in DiffPool, two nodes that are disconnected or far away in the graph can be assigned similar clusters if the nodes and their neighbors have similar features. Since this type of cluster formation is undesirable for a pooling operator (Ying et al. 2018), DiffPool utilizes an auxiliary link prediction objective during training to specifically prevent far away nodes from being clustered together. ASAP needs no such additional regularization because it ensures the localness while clustering. DiffPool's soft cluster assignment matrix S is calculated for all the nodes to all the clusters making S a dense matrix. Calculating and storing this does not scale easily for large graphs. ASAP, due to the local clustering over h-hop neighborhood, generates a sparse assignment matrix while retaining the hierarchical clustering properties of Diffpool. Further, for each pooling layer, DiffPool has to predetermine the number of clusters it needs to pick which is fixed irrespective of the input graph size. Since ASAP selects the top k fraction of nodes in current graph, it inherently takes the size of the input graph into consideration.

TopK & SAGPool While TopK completely ignores the graph structure during pooling, SAGPool modifies the TopK formulation by incorporating the graph structure through the use of a GCN network for computing node scores ϕ. To enforce sparsity, both TopK and SAGPool avoid computing the cluster assignment matrix S that DiffPool proposed. Instead of grouping multiple nodes to form a cluster in the pooled graph, they *drop* nodes from the original graph based on a score (Cangea et al. 2018) which might potentially lead to loss of node and edge information. Thus, they fail to leverage the overall graph structure while creating the clusters. In contrast to TopK and SAGPool, ASAP can capture the rich graph structure while aggregating nodes to form clusters in the pooled graph. TopK and SAGPool sample edges from the original graph to define the edge connectivity in the pooled graph. Therefore, they need to sample nodes from a local neighborhood to avoid isolated nodes in the pooled graph. Maintaining graph connectivity prevents these pooling operations from sampling representative nodes from the entire graph. The pooled graph in ASAP has a better edge connectivity compared to TopK and SAGPool because soft edge weights are computed between clusters using upto three hop connections in the original graph. Also, the use of LEConv instead of GCN for finding fitness values ϕ further allows ASAP to sample representative clusters from local neighborhoods over the entire graph.

8.2 Comparison of Self-Attention variants

Source2Token & Token2Token T2T models the membership of a node by generating a query based only on the medoid of the cluster. Graph Attention Network (GAT) (Veličković et al. 2017) is an example of T2T attention in graphs. S2T finds the importance of each node for a global task. As shown in Eq. 3, since a query vector is not used for calculating the attention scores, S2T inherently assigns the same membership score to a node for all the possible clusters that node can belong to. Hence, both S2T and T2T mechanisms fail to effectively utilize the intra-cluster information while calculating a node's cluster membership. On the other hand, M2T uses a master function f_m to generate a query vector which depends on all the entities within the cluster and hence is a more representative formulation. To understand this, consider the following scenario. If in a given cluster, a non-medoid node is removed, then the unnormalized membership scores for the rest of the nodes will remain unaffected in S2T and T2T framework whereas the change will reflect in the scores calculated using M2T mechanism. Also, from Table 6, we find that M2T performs better than S2T and T2T attention showing that M2T is better suited for global tasks like pooling.

9 Conclusion

In this paper, we introduce ASAP, a sparse and differentiable pooling method for graph structured data. ASAP clusters local subgraphs hierarchically which helps it to effectively learn the rich information present in the graph structure. We propose Master2Token self-attention framework which enables our model to better capture the membership of each node in a cluster. We also propose LEConv, a novel GNN formulation that scores the clusters based on its local and global importance. ASAP leverages LEConv to compute cluster fitness scores and samples the clusters based on it. This ensures the selection of representative clusters throughout the graph. ASAP also calculates sparse edge weights for the selected clusters and is able to capture the edge connectivity information efficiently while being scalable to large graphs. We validate the effectiveness of the components of ASAP both theoretically and empirically. Through extensive experiments, we demonstrate that ASAP achieves state-of-the-art performace on multiple graph classification datasets.

10 Acknowledgements

We would like to thank the developers of Pytorch_Geometric (Fey and Lenssen 2019) which allows quick implementation of geometric deep learning models. We would like to thank Matthias Fey for actively maintaining the library and quickly responding to our queries on github.

References

[Bahdanau, Cho, and Bengio 2014] Bahdanau, D.; Cho, K.; and Bengio, Y. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

[Bastings et al. 2017] Bastings, J.; Titov, I.; Aziz, W.; Marcheggiani, D.; and Simaan, K. 2017. Graph convolutional encoders for syntax-aware neural machine translation. *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*.

[Borgwardt et al. 2005] Borgwardt, K. M.; Ong, C. S.; Schönauer, S.; Vishwanathan, S.; Smola, A. J.; and Kriegel, H.-P. 2005. Protein function prediction via graph kernels. *Bioinformatics*.

[Bronstein et al. 2017] Bronstein, M. M.; Bruna, J.; LeCun, Y.; Szlam, A.; and Vandergheynst, P. 2017. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*.

[Bruna et al. 2013] Bruna, J.; Zaremba, W.; Szlam, A.; and LeCun, Y. 2013. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*.

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

[Cheng, Dong, and Lapata 2016] Cheng, J.; Dong, L.; and Lapata, M. 2016. Long short-term memory-networks for machine reading. *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*.

[Defferrard, Bresson, and Vandergheynst 2016] Defferrard, M.; Bresson, X.; and Vandergheynst, P. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems*.

[Dhillon, Guan, and Kulis 2007] Dhillon, I. S.; Guan, Y.; and Kulis, B. 2007. Weighted graph cuts without eigenvectors a multilevel approach. *IEEE transactions on pattern analysis and machine intelligence*.

[Dobson and Doig 2003] Dobson, P. D., and Doig, A. J. 2003. Distinguishing enzyme structures from non-enzymes without alignments. *Journal of molecular biology*.

[Fey and Lenssen 2019] Fey, M., and Lenssen, J. E. 2019. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*.

[Fey et al. 2018] Fey, M.; Eric Lenssen, J.; Weichert, F.; and Müller, H. 2018. Splinecnn: Fast geometric deep learning with continuous b-spline kernels. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

[Gao and Ji 2019] Gao, H., and Ji, S. 2019. Graph u-nets. *arXiv preprint arXiv:1905.05178*.

[Gilmer et al. 2017] Gilmer, J.; Schoenholz, S. S.; Riley, P. F.; Vinyals, O.; and Dahl, G. E. 2017. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*.

[Hamilton, Ying, and Leskovec 2017] Hamilton, W.; Ying, Z.; and Leskovec, J. 2017. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*.

[He et al. 2016] He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*.

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

[Kearnes et al. 2016] Kearnes, S.; McCloskey, K.; Berndl, M.; Pande, V.; and Riley, P. 2016. Molecular graph convolutions: moving beyond fingerprints. *Journal of Computer-Aided Molecular Design* 30(8):595608.

[Kim 2014] Kim, Y. 2014. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*.

[Kingma and Ba 2014] Kingma, D. P., and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[Kipf and Welling 2017] Kipf, T. N., and Welling, M. 2017. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*.

[Krizhevsky, Sutskever, and Hinton 2012] Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In Pereira, F.; Burges, C. J. C.; Bottou, L.; and Weinberger, K. Q., eds., *Advances in Neural Information Processing Systems 25*.

[Lee, Lee, and Kang 2019] Lee, J.; Lee, I.; and Kang, J. 2019. Self-attention graph pooling. In *Proceedings of the 36th International Conference on Machine Learning*.

[Li et al. 2016] Li, Y.; Tarlow, D.; Brockschmidt, M.; and Zemel, R. S. 2016. Gated graph sequence neural networks. *CoRR* abs/1511.05493.

[Ma et al. 2019] Ma, Y.; Wang, S.; Aggarwal, C. C.; and Tang, J. 2019. Graph convolutional networks with eigenpooling. *arXiv preprint arXiv:1904.13107*.

[Marcheggiani and Titov 2017] Marcheggiani, D., and Titov, I. 2017. Encoding sentences with graph convolutional networks for semantic role labeling. *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*.

[Monti et al. 2017] Monti, F.; Boscaini, D.; Masci, J.; Rodola, E.; Svoboda, J.; and Bronstein, M. M. 2017. Geometric deep learning on graphs and manifolds using mixture model cnns. In *Proceedings* of the IEEE Conference on Computer Vision and Pattern Recognition.

[Morris et al. 2018] Morris, C.; Ritzert, M.; Fey, M.; Hamilton, W. L.; Lenssen, J. E.; Rattan, G.; and Grohe, M. 2018. Weisfeiler and leman go neural: Higher-order graph neural networks.

[Orsini, Frasconi, and De Raedt 2015] Orsini, F.; Frasconi, P.; and De Raedt, L. 2015. Graph invariant kernels. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*.

[Sanyal et al. 2018] Sanyal, S.; Balachandran, J.; Yadati, N.; Kumar, A.; Rajagopalan, P.; Sanyal, S.; and Talukdar, P. 2018. MT-CGCNN: Integrating crystal graph convolutional neural network with multitask learning for material property prediction. *arXiv preprint arXiv:1811.05660*.

[Schaeffer 2007] Schaeffer, S. E. 2007. Graph clustering. *Computer science review*.

[Schlichtkrull et al. 2017] Schlichtkrull, M.; Kipf, T. N.; Bloem, P.; Berg, R. v. d.; Titov, I.; and Welling, M. 2017. Modeling relational data with graph convolutional networks. *arXiv preprint arXiv:1703.06103*.

[Shen et al. 2018] Shen, T.; Zhou, T.; Long, G.; Jiang, J.; Pan, S.; and Zhang, C. 2018. Disan: Directional self-attention network for rnn/cnn-free language understanding. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

[Shervashidze et al. 2011] Shervashidze, N.; Schweitzer, P.; Leeuwen, E. J. v.; Mehlhorn, K.; and Borgwardt, K. M. 2011. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*.

[Simonovsky and Komodakis 2017] Simonovsky, M., and Komodakis, N. 2017. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *Proceedings of the IEEE conference on computer vision and pattern recognition*.

[Vashishth et al. 2018a] Vashishth, S.; Dasgupta, S. S.; Ray, S. N.; and Talukdar, P. 2018a. Dating documents using graph convolution networks. *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*.

[Vashishth et al. 2018b] Vashishth, S.; Joshi, R.; Prayaga, S. S.; Bhattacharyya, C.; and Talukdar, P. 2018b. Reside: Improving distantly-supervised neural relation extraction using side information. *arXiv preprint arXiv:1812.04361*.

[Vashishth et al. 2019] Vashishth, S.; Sanyal, S.; Nitin, V.; and Talukdar, P. 2019. Composition-based multi-relational graph convolutional networks. *arXiv preprint arXiv:1911.03082*.

[Vaswani et al. 2017] Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. In *Advances in neural information processing systems*.

[Veličković et al. 2017] Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Lio, P.; and Bengio, Y. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903*.

[Vinyals, Bengio, and Kudlur 2016] Vinyals, O.; Bengio, S.; and Kudlur, M. 2016. Order matters: Sequence to sequence for sets. In *International Conference on Learning Representations (ICLR)*.

[Wale, Watson, and Karypis 2008] Wale, N.; Watson, I. A.; and Karypis, G. 2008. Comparison of descriptor spaces for chemical compound retrieval and classification. *Knowledge and Information Systems*.

[Wikipedia contributors 2017] Wikipedia contributors. 2017. Starlike tree — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Starlike_tree&oldid=791882487. [Online; accessed 17-November-2019].

[Wikipedia contributors 2019] Wikipedia contributors. 2019. Path graph — Wikipedia, the free encyclopedia. [Online; accessed 17-November-2019].

[Xie and Grossman 2018] Xie, T., and Grossman, J. C. 2018. Crystal graph convolutional neural networks for an accurate and interpretable prediction of material properties. *Phys. Rev. Lett.* 120:145301.

[Xu et al. 2018] Xu, K.; Hu, W.; Leskovec, J.; and Jegelka, S. 2018. How powerful are graph neural networks?

[Ying et al. 2018] Ying, R.; You, J.; Morris, C.; Ren, X.; Hamilton, W. L.; and Leskovec, J. 2018. Hierarchical graph representation learning with differentiable pooling. In *Proceedings of the 32Nd International Conference on Neural Information Processing Systems*.

[Zhang et al. 2018] Zhang, M.; Cui, Z.; Neumann, M.; and Chen, Y. 2018. An end-to-end deep learning architecture for graph classification. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Appendix

A Hyperparameter Tuning

For all our experiments, Adam (Kingma and Ba 2014) optimizer is used. 10-fold cross-validation is used with 80% for training and 10% for validation and test each. Models were trained for 100 epochs with lr decay of 0.5 after every 50 epochs. The range of hyperparameter search are provided in Table 9. The model with best validation accuracy was selected for testing. Our code is based on Pytorch Geometric library (Fey and Lenssen 2019).

Hyperparameter	**Range**
Hidden dimension	$\{16, 32, 64, 128\}$
Learning rate	$\{0.01, 0.001\}$
Dropout	$\{0, 0.1, 0.2, 0.3, 0.4, 0.5\}$
L2 regularization	$5e^{-4}$

Table 9: Hyperparameter tuning Summary.

B Details of Hierarchical Pooling Setup

For hierarchical pooling, we follow SAGPool (Lee, Lee, and Kang 2019) and use three layers of GCN, each followed by a pooling layer. After each pooling step, the graph is summarized using a readout function which is a concatenation of the $mean$ and max of the node representations (similar to SAGPool). The summaries are then added and passed through a network of fully-connected layers separated by dropout layers to predict the class.

C Details of Global Pooling Setup

Global Pooling architecture is same as the hierarchical architecture with the only difference that pooling is done only after all GCN layers. We do not use readout function for global pooling as they do not require them. To be comparable with other models, we restrict the feature dimension of the pooling output to be no more than 256. For global pooling layers, range for hidden dimension and lr search was same as ASAP.

Method	Range
Set2Set	processing-step $\in \{5, 10\}$
Global-Attention	transform $\in \{True, False\}$
SortPool	K is chosen such that output of pooling ≤ 256

Table 10: Global Pooling Hyperparameter Tuning Summary.

D Similarities between pooling in CNN and ASAP

In CNN, pooling methods (e.g mean pool and max pool) have two hyperparameter: kernel size and stride. Kernel size decides the number of pixels being considered for computing each new pixel value in the next layer. Stride decides the fraction of new pixels being sampled thereby controlling the size of the image in next layer. In ASAP, RF^{node} determines the neighborhood radius of clusters and k decides the sampling ratio. This makes RF^{node} and k are analogous to kernel size and stride of CNN pooling respectively. There are however some key differences. In CNN, a given kernel size corresponds to a fixed number of pixels around a central pixel whereas in ASAP, the number of nodes being considered is variable, although the neighborhood RF^{node} is constant. In CNN, stride uniformly samples from new pixels whereas in ASAP, the model has the flexibility to attend to different parts of the graph and sample accordingly.

E Ablation on pooling ratio k

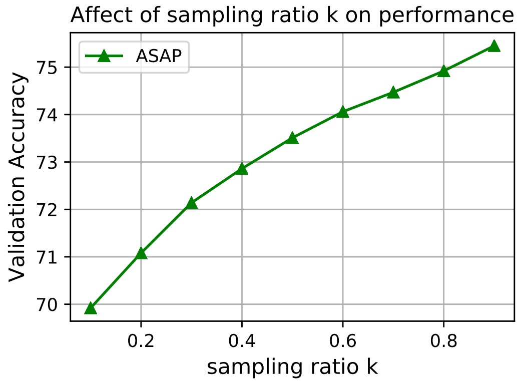


Figure 2: Validation Accuracy vs sampling ratio k on NCI1 dataset.

Intuitively, higher k will lead to more information retention. Hence, we expect an increase in performance with increasing k. This is empirically observed in Fig. 2. However, as k increases, the computational resources required by the model also increase because a relatively larger pooled graph gets propagated to the later layers. Hence, there is a trade-off between performance and computational requirement while deciding on the pooling ratio k.

F Proof of Theorem 1

Theorem 1. *Let $\mathcal{G}$ be a graph with positive adjacency matrix A i.e., $A_{i,j} \geq 0$. Consider any function $f(X, A) : \mathbb{R}^{N \times d} \times \mathbb{R}^{N \times N} \rightarrow \mathbb{R}^{N \times 1}$ which depends on difference between a node and its neighbors after a linear transformation $W \in \mathbb{R}^{d \times d}$. For e.g:*

$$f_i = \sigma(\alpha_i x_i W + \sum_{j \in \mathcal{N}(i)} \beta_{i,j}(x_i W - x_j W))$$

where $f_i, \alpha_i, \beta_{i,j} \in \mathbb{R}$ and $x_i \in \mathbb{R}^d$.

a) *If fitness value $\phi = GCN(X, A)$ then ϕ cannot learn f.*

b) *If fitness value $\phi = LEConv(X, A)$ then ϕ can learn f.*

Proof. For GCN, $\phi_i = \sigma(\sum_{j \in \mathcal{N}(x_i) \cup \{i\}} A_{i,j} x_j W)$ where W is a learnable matrix. Since $A_{i,j} \geq 0$, ϕ_i cannot have a term of the form $\beta_{i,j}(x_i W - x_j W)$ which proves the first part of the theorem. We prove the second part by showing that LEConv can learn the following function f:

$$f_i = \sigma(\alpha_i x_i W + \sum_{j \in \mathcal{N}(i)} \beta_{i,j}(x_i W - x_j W)) \tag{11}$$

LEConv formulation is defined as:

$$\phi_i = \sigma(x_i W_1 + \sum_{j \in \mathcal{N}(i)} A_{i,j}(x_i W_2 - x_j W_3)) \tag{12}$$

where W_1, W_2 and W_3 are learnable matrices. For $W_3 = W_2 = W_1$, $\alpha_1 = 1$ and $\beta_{i,j} = A_{i,j}$ we find Eq. (12) is equal to Eq. (11). □

G Graph Connectivity

Proof of Theorem 2

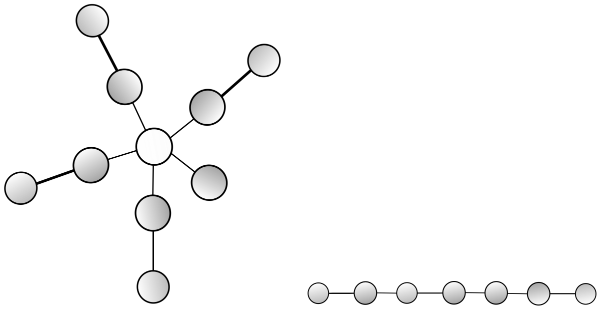
Definition 1. *For a graph $\mathcal{G}$, we define optimum-nodes $n_h^*(\mathcal{G})$ as the maximum number of nodes that can be selected which are atleast h hops away from each other.*

Definition 2. *For a given number of nodes N, we define optimum-tree $\mathcal{T}_N^*$ as the tree which has maximum optimum-nodes $n_h^*(\mathcal{T}_N)$ among all possible trees $\mathcal{T}_N$ with N nodes.*

Lemma 1. *Let $\mathcal{T}_N^*$ be an optimum-tree of N vertices and $\mathcal{T}_{N-1}^*$ be an optimum tree with $N-1$ vertices. The optimum-nodes of $\mathcal{T}_N^*$ and $\mathcal{T}_{N-1}^*$ differ by atmost one, i.e., $0 \leq n_h^*(\mathcal{T}_N^*) - n_h^*(\mathcal{T}_{N-1}^*) \leq 1$.*

Proof. Consider $\mathcal{T}_N^*$ which has N nodes. We can remove any one of the leaf nodes in $\mathcal{T}_N^*$ to obtain a tree $\mathcal{T}_{N-1}$ with $N-1$ nodes. If any one of the nodes in $n_h^*(\mathcal{T}_N^*)$ was removed, then $n_h^*(\mathcal{T}_{N-1})$ would become $n_h^*(\mathcal{T}_N^*) - 1$. If any other node was removed , then being a leaf it does not constitute the shortest path between any of the $n_h^*(\mathcal{T}_{N-1})$ nodes. This implies that the optimum-nodes for $\mathcal{T}_{N-1}$ is atleast $n_h^*(\mathcal{T}_N^*) - 1$, i.e.,

$$n_h^*(\mathcal{T}_N^*) - 1 \leq n_h^*(\mathcal{T}_{N-1}) \leq n_h^*(\mathcal{T}_N^*) \tag{13}$$



(a) Starlike tree (b) Path Graph

Figure 3: (a) Balanced Starlike tree with height 2. (b) Path Graph

Since $\mathcal{T}^*_{N-1}$ is the optimal-tree, we know that:

$$n^*_h(\mathcal{T}_{N-1}) \leq n^*_h(\mathcal{T}^*_{N-1}) \tag{14}$$

Using Eq. (13) and (14) we can write:

$$n^*_h(\mathcal{T}^*_N) - n^*_h(\mathcal{T}^*_{N-1}) \leq 1$$

which proves our lemma. □

Lemma 2. *Let $\mathcal{T}^*_N$ be an optimum-tree of N vertices and $\mathcal{T}^*_{N-1}$ be an optimum-tree of $N-1$ vertices. $\mathcal{T}^*_{N-1}$ is an induced subgraph of $\mathcal{T}^*_N$.*

Proof. Let us choose a node to be removed from $\mathcal{T}^*_N$ and join its neighboring nodes to obtain a tree $\mathcal{T}_{N-1}$ with $N-1$ nodes with an objective of ensuring a maximum $n^*_h(\mathcal{T}_{N-1})$. To do so, we can only remove a leaf node from $\mathcal{T}^*_N$. This is because removing non-leaf nodes can reduce the shortest path between multiple pairs of nodes whereas removing leaf-nodes will reduce only the shortest path to nodes from the new leaf at that position. This ensures least reduction in optimum-nodes for $\mathcal{T}_{N-1}$. Removing a leaf node implies that $n^*_h(\mathcal{T}_{N-1})$ cannot be lesser than $n^*_h(\mathcal{T}^*_N) - 1$ as it affects only the paths involving that particular leaf node. Using Lemma 1, we see that $\mathcal{T}_{N-1}$ is equivalent to $\mathcal{T}^*_{N-1}$, i.e., $\mathcal{T}_{N-1}$ is one of the possible optimal-trees with $N-1$ nodes. Since $\mathcal{T}_{N-1}$ was formed by removing a leaf node from $\mathcal{T}^*_N$, we find that $\mathcal{T}^*_{N-1}$ is indeed an induced subgraph of $\mathcal{T}^*_N$. □

Definition 3. *A starlike tree is a tree having atmost one node (root) with degree greater than two (Wikipedia contributors 2017). We consider starlike tree with height $\lceil h/2 \rceil$ to be balanced, if there is atmost one leaf which is at a height less than $\lceil h/2 \rceil$ while the rest are all at a height $\lceil h/2 \rceil$ from the root. Figure 3(a) depicts an example of a balanced starlike tree with $h = 2$.*

Definition 4. *A path graph is a graph such that its nodes can be placed on a straight line. There are ~~not more than~~ only two nodes in a path graph which have degree one while the rest have a degree of two. Figure 3(b) shows an example of a path graph (Wikipedia contributors 2019).*

Lemma 3. *For a balanced starlike tree with height $h/2$, where h is even, $n^*_h(\mathcal{T}_N) = \left\lfloor \frac{N-1}{\frac{h}{2}} \right\rfloor$, i.e., when the leaves are selected.*

Lemma 4. *Among all the possible trees $\mathcal{T}_N$ which have N vertices, the maximum $n^*_h(\mathcal{T}_N)$ achievable is $\left\lfloor \frac{N-1}{\frac{h}{2}} \right\rfloor$, which is obtained if the tree is a balanced starlike tree with height $h/2$ if h is even.*

Proof. To prove the lemma, we use induction. Here, the base case corresponds to a path graph $\mathcal{T}_{h+1}$ with $h+1$ nodes, a trivial case of starlike graph, as it has only 2 nodes which are h hops away. From the formula $\left\lfloor \frac{N-1}{\frac{h}{2}} \right\rfloor$, we get $n^*_h(\mathcal{T}_{h+1}) = 2$ which verifies the base case.

For any $N-1$, let us assume that the lemma is true, i.e., a balanced starlike tree with height $h/2$ achieves the maximum $n^*_h(\mathcal{T}_{N-1})$ for any tree $\mathcal{T}_{N-1}$ with $N-1$ vertices. Consider $\mathcal{T}^*_N$ to be the optimal-tree for N nodes. From Lemma (2), we know that $\mathcal{T}^*_{N-1}$ is an induced subgraph of $\mathcal{T}^*_N$. This means that $\mathcal{T}^*_N$ can be obtained by adding a node to $\mathcal{T}^*_{N-1}$. Since we are constructing $\mathcal{T}^*_N$, we need to add a node to $\mathcal{T}^*_{N-1}$ such that maximum nodes can be selected which are atleast h hops away. There are three possible structures for the tree $\mathcal{T}^*_{N-1}$ depending on the minimum height among all its branches: (a) minimum height among all the branches is less than $h/2 - 1$, (b) minimum height among all the branches is equal to $h/2 - 1$ and (c) minimum height among all the branches is equal to $h/2$. Although case (a) is not possible as we assumed $\mathcal{T}^*_{N-1}$ to be a balanced starlike tree, we consider it for the sake of completeness. For case (a), no matter where we add the node, $n^*_h(\mathcal{T}^*_N)$ will not increase. However, we should add the node to the leaf of the branch with least height as it will allow the new leaf of that branch to be chosen in case the number of nodes in tree is increased to some $N^{'} > N$ such that height of that branch becomes $h/2$. For case (b), we should add the node to the leaf of the branch with least height so that its height becomes $h/2$ and the new leaf of that branch gets selected. For case (c), no matter where we add the node, $n^*_h(\mathcal{T}^*_N)$ will not increase. Unlike case (a), we should add the new node to the root so as to start a new leaf which could be selected if that branch grows to a height $h/2$ for some $N^{'} > N$. For all the three cases, $\mathcal{T}^*_N$ is a balanced starlike tree as the new node is either added to the leaf of a branch if minimum height of a leaf is less than $h/2$ or to the root if the minimum height of the branches is $h/2$. Hence, by induction, the lemma is proved. □

Theorem 2. *Let the input graph $\mathcal{T}$ be a tree of any possible structure with N nodes. Let k^* be the lower bound on sampling ratio k to ensure the existence of atleast one edge in the pooled graph irrespective of the structure of $\mathcal{T}$ and the location of the selected nodes. For TopK or SAGPool, $k^* \to 1$ whereas for ASAP, $k^* \to 0.5$ as $N \to \infty$.*

Proof. From Lemma (4) and (3), we know that among all the possible trees $\mathcal{T}_N$ which have N vertices, the maximum $n^*_h(\mathcal{T}_N)$ achievable is $\left\lfloor \frac{N-1}{\frac{h}{2}} \right\rfloor$. Using pigeon-hole principle

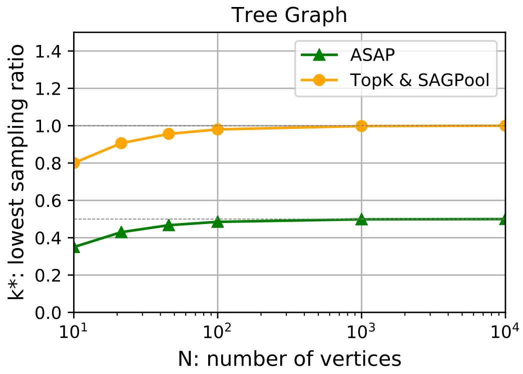


Figure 4: Minimum sampling ratio k^* vs N for Path-Graph

we can show that for a pooling method with RF^{edge} if the number of sampled clusters is greater than $n^*_{RF^{edge}}(\mathcal{T}_N)$ then there will always be an edge in the pooled graph irrespective of the position of the selected clusters:

$$\begin{aligned} \lceil k^*N \rceil &> \left\lfloor \frac{N-1}{\frac{RF^{edge}+1}{2}} \right\rfloor \\ k^*N &> \left\lfloor \frac{N-1}{\frac{RF^{edge}+1}{2}} \right\rfloor \\ k^*N+1 &> \frac{N-1}{\frac{RF^{edge}+1}{2}} \end{aligned} \tag{15}$$

Let us consider 1-hop neighborhood for pooling, i.e., $h = 1$. Substituting $RF^{edge} = h$ in Eq. (15) for TopK and SAGPool we get:

$$k^* > 1 - \frac{2}{N}$$

and as N $\rightarrow \infty$ we obtain $k^* \rightarrow 1$. Substituting $RF^{edge} = 2h+1$ in Eq. (15) for ASAP we get:

$$k^* > \frac{1}{2} - \frac{3}{2N}$$

and as N $\rightarrow \infty$ we obtain $k^* \rightarrow 0.5$ □

Similar Analysis for Path Graph

Lemma 5. *For a path graph $\mathcal{G}_{path}$ with N nodes, $n^*_h(G_{path}) = \lceil \frac{N}{h} \rceil$.*

Lemma 6. *Consider the input graph to be a path graph $\mathcal{G}_{path}$ with N nodes. To ensure that a pooling operator with RF^{edge} and sampling ratio k has at least one edge in the pooled graph, irrespective of the location of selected clusters, we have the following inequality on k: $k \geq (\frac{1}{RF^{edge}+1} + \frac{1}{N})$.*

Proof. From Lemma (5), we know that $n^*_{RF^{edge}}(G_{path}) = \lceil \frac{N}{RF^{edge}} \rceil$. Using pigeon-hole principle we can show that for a pooling method with RF^{edge}, if the number of sampled clusters is greater than $n^*_{RF^{edge}}(G_{path})$, then there will always be an edge in the pooled graph irrespective of the position of the selected clusters:

$$\begin{aligned} \lceil kN \rceil &> \left\lceil \frac{N}{RF^{edge}+1} \right\rceil \\ kN &> \left\lceil \frac{N}{RF^{edge}+1} \right\rceil \\ kN &\geq \frac{N}{RF^{edge}+1} + 1 \end{aligned} \tag{16}$$

From Eq. (16), we get $k \geq (\frac{1}{RF^{edge}+1} + \frac{1}{N})$ which completes the proof. □

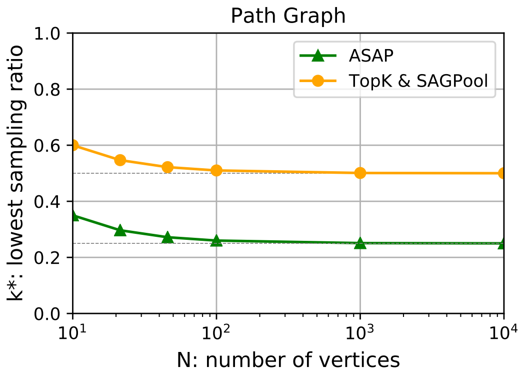


Figure 5: Minimum sampling ratio k^* vs N for Path-Graph

Theorem 3. *Consider the input graph to be a path graph with N nodes. Let k^* be the lower bound on sampling ratio k to ensure the existence of atleast one edge in the pooled graph. For TopK or SAGPool, $k^* \rightarrow 0.5$ as $N \rightarrow \infty$ whereas for ASAP, $k^* \rightarrow 0.25$ as $N \rightarrow \infty$.*

Proof. From Lemma (6), we get $k^* = \frac{1}{RF^{edge}+1} + \frac{1}{N}$. Using h = 1 for TopK and SAGPool when N tends to infinity i.e. $k^* = \lim_{N\rightarrow\infty} \frac{1}{2} + \frac{1}{N}$, we get $k^* \rightarrow 0.25$. Using h = 3 for ASAP when N tends to infinity i.e. $k^* = \lim_{N\rightarrow\infty} \frac{1}{4} + \frac{1}{N}$, we get $k^* \rightarrow 0.25$. □

Graph Connectivity via k^{th} Graph Power.

To minimize the possibility of nodes getting isolated in pooled graph, TopK employs k^{th} graph power i.e. $\hat{A}^k$ instead of $\hat{A}$. This helps in increasing the density of the graph before pooling. While using k^{th} graph power, TopK can connect two nodes which are atmost k hops away whereas ASAP in this setting can connect upto $k + 2h$ hops in the original graph. As $h \geq 1$, ASAP will always have better connectivity given k^{th} graph power.

H Graph Permutation Equivariance

Given a permutation matrix $P \in \{0,1\}^{n\times n}$ and a function $f(X, A)$ depending on graph with node feature matrix X and adjacency matrix A, graph permutation is defined as $f(PX, PAP^T)$, node permutation is defined as $f(PX, A)$

and edge permutation is defined as $f(X, PAP^T)$. Graph pooling operations should produce pooled graphs which are isomorphic after graph permutation i.e., they need to be graph permutation equivariant or invariant. We show that ASAP has the property of being graph permutation equivariant.

Proposition 1. *ASAP is a graph permutation equivariant pooling operator.*

Proof. Since S is computed by an attention mechanism which attends to all edges in the graph, we have:

$$S \rightarrow PSP^T \tag{17}$$

Selecting top $\lceil kN \rceil$ clusters denoted by indices i, changes $\hat{S}$ as:

$$\hat{S} \rightarrow P\hat{S}(P[i,i])^T \tag{18}$$

Using Eq. (18) and $\hat{S} = S(:,\hat{i}), X^p = \hat{X}^c(\hat{i})$, we can write:

$$X^p \rightarrow P[i,i]X^p \tag{19}$$

Since $A^p = \hat{S}^T \hat{A}^c \hat{S}$ and $\hat{S} = S(:,\hat{i}), X^p = \hat{X}^c(\hat{i})$, we get:

$$A^p \rightarrow P[i,i]A^p(P[i,i])^T \tag{20}$$

From Eq. (19) and Eq. (20), we see that graph permutation does not change the output features. It only changes the order of the computed feature and hence is isomorphic to the pooled graph. □

I Pseudo Code

Algorithm 1 is a pseudo code of ASAP. The Master2Token working is explained in Algorithm 2.

```
Algorithm 1: ASAP algorithm
Input        : Graph $G(\mathcal{V},\mathcal{E})$; Node features $X$;
               Weighted adjacency matrix $A$;
               Master2Token attention function
               MASTER2TOKEN; Local Extrema
               Convolution operator LECONV; pooling
               ratio $k$; Top-k selection operator TOPK;
               non-linearity $\sigma$
Intermediate: Clustered graph $G^c(\mathcal{V},\mathcal{E})$ with node
               features $X^c$ and weighted adjacency
               matrix $A^c$; Cluster assignment matrix $S$;
               Cluster fitness vector $\Phi$
Output       : Pooled graph $G^p$ with node features $X^p$
               and weighted adjacency matrix $A^p$
$X^c, S \leftarrow$ MASTER2TOKEN$(X, A)$;
$A^c \leftarrow A$;
$\Phi \leftarrow$ LECONV$(X^c, A^c)$;
$\hat{X}^c \leftarrow \Phi \odot X^c$;
$\hat{i} \leftarrow$ TOPK$(\Phi, k)$;
$\hat{S} \leftarrow S(:,\hat{i})$;
$X^p \leftarrow \hat{X}^c(\hat{i},:)$;
$A^p \leftarrow \hat{S}^T \hat{A}^c \hat{S}$
```

```
Algorithm 2: MASTER2TOKEN algorithm
Input  : Graph $G(\mathcal{V},\mathcal{E})$; Node features $X$; Weighted
         adjacency matrix $A$; Graph Convolution
         operator GCN; Weight matrix $W$, weight
         vector $\vec{w}$; softmax function $softmax$; Cluster
         neighborhood function $c_h$
Output: Clustered graph node features $X^c$; Cluster
         assignment matrix $S$
$X' \leftarrow \mathrm{GCN}(X, A)$;
for $i = 1...|\mathcal{V}|$ do
    $x_i^c \leftarrow \vec{0}$;
    $m_i \leftarrow \max_{j \in c_h(v_i)}(x'_j)$.;
    for $j \in c_h(v_i)$ do
        $\alpha_{i,j} \leftarrow softmax(\vec{w}^T \sigma(W m_i \parallel x'_j))$;
        $S_{i,j} \leftarrow \alpha_{i,j}$;
        $x_i^c \leftarrow x_i^c + \alpha_{i,j} x_j$;
    end
end
```