Haar Graph Pooling: Supplementary Material

Yu Guang Wang [1 2] **Ming Li** [3] **Zheng Ma** [4] **Guido Montúfar** [5 2] **Xiaosheng Zhuang** [6] **Yanan Fan** [1]

A. Efficient Computation for HaarPooling

For the HaarPooling introduced in Definition 1, we can develop a fast computational strategy by virtue of fast adjoint Haar transforms. Let $\mathcal{G}_{0\to K}$ be a coarse-grained chain of the graph $\mathcal{G}_0$. For convenience, we label the vertices of the level-j graph $\mathcal{G}_j$ by $V_j := \{v_1^{(j)}, \dots, v_{N_j}^{(j)}\}$.

An efficient algorithm for HaarPooling The HaarPooling can be computed efficiently by using the hierarchical structure of the chain, as we introduce as follows. For $j = 1, \dots, K$, let $c_k^{(j)}$ be the number of children of $v_k^{(j)}$, i.e. the number of vertices of $\mathcal{G}_{j-1}$ which belongs to the cluster $v_k^{(j)}$, for $k = 1, \dots, N_j$. For $j = 0$, we let $c_k^{(0)} \equiv 1$ for $k = 1, \dots, N_0$. Now, for $j = 0, \dots, K$ and $k = 1, \dots, N_j$, define the weight for the node $v_k^{(j)}$ of layer j by

$$w_k^{(j)} := \frac{1}{\sqrt{c_k^{(j)}}}. \tag{1}$$

Let $W_{0\to K} := \{w_k^{(j)} \,|\, j = 0, \dots, K,\ k = 1, \dots, N_j\}$. Then, for $j = 0, \dots, K$, the weighted chain $(\mathcal{G}_{j\to K}, W_{j\to K})$ becomes a *filtration* if each parent of the chain $\mathcal{G}_{j\to K}$ has at least two children.

Let $j = 0, \dots, K$. For the jth HaarPooling layer, let $\{\phi_\ell^{(j)}\}_{\ell=1}^{N_j}$ be the Haar basis for the jth layer, which we also call the Haar basis for the filtration $(\mathcal{G}_{j\to K}, W_{j\to K})$ of a graph $\mathcal{G}$. For $k = 1, \dots, N_j$, we let $X(v_k^{(j)}) = X(v_k^{(j)}, \cdot) \in \mathbb{R}^{d_j}$ the feature vector at node $v_k^{(j)}$. We define the weighted sum for feature $X \in \mathbb{R}^{N_j \times d_j}$ for $d_j \geq 1$ by

$$\mathcal{S}^{(j)}\big(X, v_k^{(j)}\big) := X(v_k^{(j)}), \quad v_k^{(j)} \in \mathcal{G}_j, \tag{2}$$

and recursively, for $i = j+1, \dots, K$ and $v_k^{(i)} \in \mathcal{G}_i$,

$$\mathcal{S}^{(i)}\big(X, v_k^{(i)}\big) := \sum_{v_{k'}^{(i-1)} \in v_k^{(i)}} w_{k'}^{(i-1)} \mathcal{S}^{(i-1)}\big(X, v_{k'}^{(i-1)}\big). \tag{3}$$

For each vertex $v_k^{(i)}$ of $\mathcal{G}_i$, the $\mathcal{S}^{(i)}\big(X, v_k^{(i)}\big)$ is the weighted sum of the $\mathcal{S}^{(i-1)}\big(X, v_{k'}^{(i-1)}\big)$ at the level $i-1$ for those vertices $v_{k'}^{(i-1)}$ of $\mathcal{G}_{i-1}$ whose parent is $v_k^{(i)}$.

Theorem 1. *For $0 \leq j \leq K-1$, let $\{\phi_\ell^{(i)}\}_{\ell=1}^{N_i}$ for $i = j+1, \dots, K$ be the Haar bases for the filtration $(\mathcal{G}_{j\to K}, W_{j\to K})$ at layer i. Then, the compressive Haar transform for the jth HaarPooling layer can be computed by, for the feature $X \in \mathbb{R}^{N_j \times d_j}$ and $\ell = 1, \dots, N_j$,*

$$\big(\Phi_j^T X\big)_\ell = \sum_{k=1}^{N_i} \mathcal{S}^{(i)}\big(X, v_k^{(i)}\big) w_k^{(i)} \phi_\ell^{(i)}(v_k^{(i)}), \tag{4}$$

where i is the largest possible number in $\{j+1, \dots, K\}$ such that $\phi_\ell^{(i)}$ is the ℓth member of the orthonormal basis $\{\phi_\ell^{(i)}\}_{\ell=1}^{N_i}$ for $l_2(\mathcal{G}_i)$, $v_k^{(i)}$ are the vertices of $\mathcal{G}_i$ and the weights $w_k^{(i)}$ are given by Equation (1).

We give the algorithmic implementation of Theorem 1 in Algorithm 1, which provides a fast algorithm for HaarPooling at each layer.

B. Proofs

Proof for Equation (6) in Section 4. We only need to prove the first formula. The second is obtained by definition. To simplify notation, we let $f = X_j^{\text{in}}$. By construction of Haar basis, for some layer j, the first N_{j+1} basis vectors

$$\phi_\ell^{(j)}(v) = \phi_\ell^{(j+1)}(p)/\sqrt{|Pa_{\mathcal{G}}(v)|}, \quad \text{for } p = Pa_{\mathcal{G}}(v).$$

*Equal contribution [1]School of Mathematics and Statistics, University of New South Wales, Sydney, Australia [2]Max Planck Institute for Mathematics in the Sciences, Leipzig, Germany [3]School of Teacher Education, Zhejiang Normal University, Jinhua, China [4]Department of Physics, Princeton University, New Jersey, USA [5]Department of Mathematics and Department of Statistics, University of California, Los Angeles [6]Department of Mathematics, City University of Hong Kong, Hong Kong. Correspondence to: Ming Li <ming.li.ltu@gmail.com>, Yu Guang Wang <yuguang.wang@unsw.edu.au>, Guido Montúfar <montufar@math.ucla.edu>.

Proceedings of the 37^{th} International Conference on Machine Learning, Online, PMLR 119, 2020.

Algorithm 1 Fast HaarPooling for One Layer

Input: Input feature $X_j^{\rm in}$ for the jth pooling layer given $j = 0, \ldots, K-1$ in a GNN with total K HaarPooling layers; the chain $\mathcal{G}_{j\to K}$ associated with the HaarPooling; numbers N_i of nodes for layers $i = j, \ldots, K$.

Output: $\Phi_j^T X_j^{\rm in}$ from Definition 1.

Step 1: Evaluate the sums for $i = j, \ldots, K$ recursively, using Equations (2) and (3):
$\mathcal{S}^{(i)}\big(X_j^{\rm in}, v_k^{(i)}\big) \quad \forall v_k^{(i)} \in V_i$.

Step 2:
for $\ell = 1$ **to** N_{j+1} **do**
Set $N_K = 0$.
Compute i such that $N_{i+1} + 1 \le \ell \le N_i$.
Evaluate $\sum_{k=1}^{N_i} \mathcal{S}^{(i)}(X_j^{\rm in}, v_k^{(i)}) w_k^{(i)} \phi_\ell^{(i)}(v_k^{(i)})$ in Equation (4) by the two steps:
(a) Compute the product for all $v_k^{(i)} \in V_i$:
$T_\ell(X_j^{\rm in}, v_k^{(i)}) = \mathcal{S}^{(i)}(X_j^{\rm in}, v_k^{(i)}) w_k^{(i)} \phi_\ell^{(i)}(v_k^{(i)})$.
(b) Evaluate sum $\sum_{k=1}^{N_i} T_\ell(X_j^{\rm in}, v_k^{(i)})$.
end for

Then, the Fourier coefficient of f for the ℓth basis vector is the inner product

$$\begin{aligned}\left\langle f, \phi_\ell^{(j)} \right\rangle &= \sum_{v\in\mathcal{G}_j} f(v)\overline{\phi_\ell^{(j)}(v)} \\ &= \sum_{p\in\mathcal{G}_{j+1}} \sum_{p=Pa_{\mathcal{G}}(v)} f(v)\overline{\phi_\ell^{(j+1)}(p)}/\sqrt{|Pa_{\mathcal{G}}(v)|} \\ &= \sum_{p\in\mathcal{G}_{j+1}} \widetilde{f}(p)\overline{\phi_\ell^{(j+1)}(p)} = \left\langle \widetilde{f}, \phi_\ell^{(j+1)} \right\rangle\end{aligned}$$

where we have let

$$\widetilde{f}(p) := \frac{1}{\sqrt{|Pa_{\mathcal{G}}(v)|}} \sum_{p=Pa_{\mathcal{G}}(v)} f(v).$$

This then gives

$$\sum_{\ell=1}^{N_{j+1}} \left|\left\langle f, \phi_\ell^{(j)} \right\rangle\right|^2 = \sum_{\ell=1}^{N_{j+1}} \left|\left\langle \widetilde{f}, \phi_\ell^{(j+1)} \right\rangle\right|^2. \tag{5}$$

Since $\{\phi_\ell\}_{\ell=1}^{N_{j+1}}$ forms an orthonormal basis for $\ell_2(\mathcal{G}_{j+1})$,

$$\begin{aligned}\left\|\Phi_j^T f\right\|^2 &= \sum_{\ell=1}^{N_{j+1}} \left|\left\langle \widetilde{f}, \phi_\ell^{(j+1)} \right\rangle\right|^2 = \left\|\widetilde{f}\right\|^2 = \sum_{p\in\mathcal{G}_{j+1}} \left|\widetilde{f}(p)\right|^2 \\ &= \sum_{p\in\mathcal{G}_{j+1}} \left| \frac{1}{\sqrt{|Pa_{\mathcal{G}}(v)|}} \sum_{p=Pa_{\mathcal{G}}(v)} f(v) \right|^2.\end{aligned}$$

This proves the left formula in Equation (6) in Section 4. □

Proof of Theorem 1. By the relation between $\phi_\ell^{(i)}$ and $\phi_\ell^{(j)}$, for $i = j+1, \ldots, K$ and $\ell = 1, \ldots, N_{j+1}$,

$$\begin{aligned}\big(\Phi_j^T X\big)_\ell &= \sum_{k=1}^{N_j} X(v_k^{(j)}) \phi_\ell^{(j)}(v_k^{(j)}) \\ &= \sum_{k'=1}^{N_{j+1}} \left(\sum_{Pa_{\mathcal{G}}(v_k^{(j)}) = v_{k'}^{(j+1)}} X(v_k^{(j)}) \right) w_{k'}^{(j+1)} \phi_\ell^{(j+1)}(v_{k'}^{(j+1)}) \\ &= \sum_{k'=1}^{N_{j+1}} \mathcal{S}^{(j+1)}(X, v_{k'}^{(j+1)}) w_{k'}^{(j+1)} \phi_\ell^{(j+1)}(v_{k'}^{(j+1)}) \\ &= \sum_{k''=1}^{N_{j+2}} \left(\sum_{Pa_{\mathcal{G}}(v_{k'}^{(j+1)}) = v_{k''}^{(j+2)}} \mathcal{S}^{(j+1)}(X, v_{k'}^{(j+1)}) w_{k'}^{(j+1)} \right) \\ &\quad \times w_{k''}^{(j+2)} \phi_\ell^{(j+2)}(v_{k''}^{(j+2)}) \\ &= \sum_{k''=1}^{N_{j+2}} \mathcal{S}^{(j+2)}(X, v_{k''}^{(j+2)}) w_{k''}^{(j+2)} \phi_\ell^{(j+2)}(v_{k''}^{(j+2)}) \\ &\cdots\cdots \\ &= \sum_{k=1}^{N_i} \mathcal{S}^{(i)}(X, v_k^{(i)}) w_k^{(i)} \phi_\ell^{(i)}(v_k^{(i)}),\end{aligned}$$

where $v_{k'}^{(j+1)}$ is the parent of $v_k^{(j)}$ and $v_{k''}^{(j+2)}$ is the parent of $v_k^{(j+1)}$, and we recursively compute the summation to obtain the last equality, thus completing the proof. □

C. Experimental Setting

The hyperparameters include batch size; learning rate, weight decay rate (these two for optimization); the maximal number of epochs; patience for early stopping. Table 1 shows the choice of hyperparameters for classification benchmark datasets.

D. Property Comparison of Pooling Methods

Here we provide a comparison of the properties of HaarPooling with existing pooling methods. The properties in the comparison include time complexity and space complexity, and whether involving the clustering, hierarchical pooling (which is then not a global pooling), spectral-based, node feature or graph structure, and sparse representation. We compare HaarPooling (denoted by HaarPool in the table) to other methods (SortPool, DiffPool, gPool, SAGPool, and EigenPool).

- The SortPool (i.e., SortPooling) is a global pooling which uses node signature (i.e., Weisfeiler-Lehman color of vertex) to sort all vertices by the values of the channels of the input data. Thus, the time complexity

Table 1. Hyperparameter setting

Data Set	MUTAG	PROTEINS	NCI1	NCI109	MUTAGEN
batch size	60	50	100	100	100
max #epochs	30	20	150	150	50
early stopping	15	20	50	50	50
learning rate	0.01	0.001	0.001	0.01	0.01
weight decay	0.0005	0.0005	0.0005	0.0001	0.0005

Table 2. Property comparison for pooling methods

Method	Time Complexity	Space Complexity	Clustering-based	Spectral-based	Hierarchical Pooling	Use Node Feature	Use Graph Structure	Sparse Representation
SortPool	$\mathcal{O}(\|V\|^2)$	$\mathcal{O}(\|V\|)$				✓		
DiffPool	$\mathcal{O}(\|V\|^2)$	$\mathcal{O}(k\|V\|^2)$			✓	✓		
gPool	$\mathcal{O}(\|V\|^2)$	$\mathcal{O}(\|V\|+\|E\|)$			✓	✓		
SAGPool	$\mathcal{O}(\|E\|)$	$\mathcal{O}(\|V\|+\|E\|)$			✓	✓	✓	
EigenPool	$\mathcal{O}(\|V\|^2)$	$\mathcal{O}(\|V\|^2)$	✓	✓	✓	✓	✓	
HaarPool	$\mathcal{O}(\|V\|)$	$\mathcal{O}(\|V\|^2\epsilon)$	✓	✓	✓	✓	✓	✓

'$|V|$' is the number of vertices of the input graph; '$|E|$' is the number of edges of the input graph; 'ϵ' in HaarPooling is the sparsity of the compressive Haar transform matrix; 'k' in the DiffPool is the pooling ratio.

(worst case) of SortPool is $\mathcal{O}(|V|^2)$ and space complexity is $\mathcal{O}(|V|)$. Other pooling methods mentioned here are all hierarchical pooling.

- DiffPool and gPool both use the node feature and have time complexity $\mathcal{O}(|V|^2)$. The DiffPool learns the assignment matrices in an end-to-end manner and has space complexity $\mathcal{O}(k|V|^2)$ for pooling ratio k. The gPool projects all nodes to a learnable vector to generate scores for nodes, and then sorts the nodes by the projection scores; the space complexity is $\mathcal{O}(|V|+|E|)$.

- SAGPool uses the graph convolution to calculate the attention scores of nodes and then selects top-ranked nodes for pooling. The time complexity of SAGPool is $\mathcal{O}(|E|)$, and the space complexity is $\mathcal{O}(|V|+|E|)$ due to the sparsity of the pooling matrix.

- EigenPool, which considers both the node feature and graph structure, uses the eigendecomposition of subgraphs (from clustering) of the input graph, and pools the input data by the Fourier transforms of the assembled basis matrix. Due to the computational cost of eigendecomposition, the time complexity of EigenPool is $\mathcal{O}(|V|^2)$, and the space complexity is $\mathcal{O}(|V|^2)$.

- HaarPool which uses the sparse representation of data by compressive Haar basis has linear time complexity $\mathcal{O}(|V|)$ (up to a $\log|V|$ term), and the space complexity is $\mathcal{O}(|V|^2\epsilon)$, where ϵ is the sparsity of compressive Haar transform matrix and is usually very small. HaarPooling can be even faster in practice, as the cost of the compressive Haar transform is dependent on the sparsity of the Haar basis matrix. The sparsity of the compressive Haar basis matrix is mainly reliant on the chain/tree for the graph. From our construction, the compressive Haar basis matrix is always highly sparse. Thus, the computational cost does not skyrocket as the size of the graph increases.

In Table 2, the HaarPool is the only pooling method which has time complexity proportional to the number of nodes and thus has a faster implementation.