Haar Graph Pooling

Yu Guang Wang

MPI/UNSW
yuguang.wang@{mis.mpg.de, unsw.edu.au}

Yanan Fan (UNSW) Junbin Gao (U Sydney) Ming Li (ZJNU)
Pietro Liò (Cambridge) Zheng Ma (Princeton)
Guido Montúfar (UCLA/MPI) Xuebin Zheng (U Sydney)
Xiaosheng Zhuang (CityU HK) Bingxin Zhou (U Sydney)

Cambridge 2020

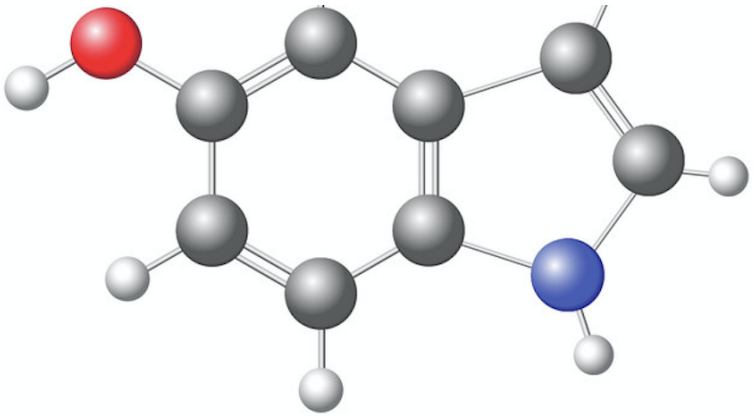



What are Graph Neural Networks?

Graph neural networks (GNNs) are deep nets which take graph-structure data as input.

- The structure of a typical GNN is similar as CNN, which has multiple neural layers, each of which contains a number of neuronal nodes.
- The input of GNN is graph-structured data which usually consists of adjacency matrix (representing edges) plus features on vertices.

Inputs of GNN and CNN



Graph data for GNN

nodes, edges

size varies

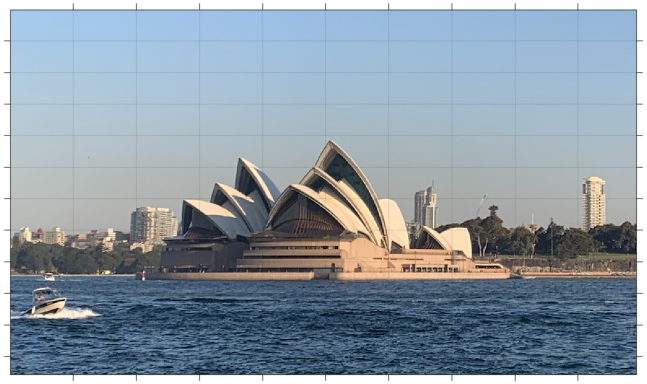
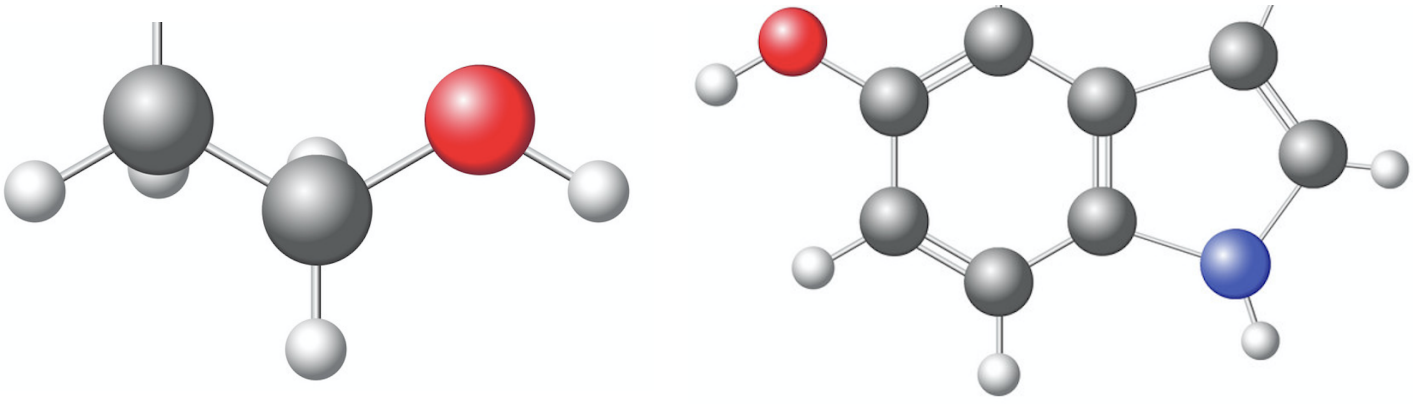


Image for CNN

pixels

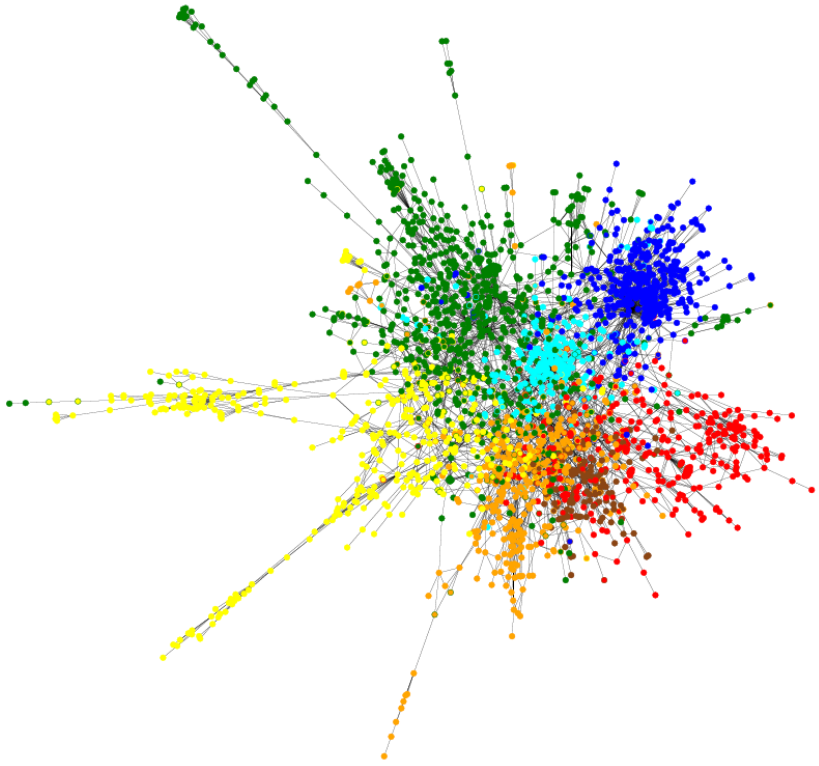
regular grid

Molecules as Graph Data



Graph-structured data: two molecules with atoms as nodes and bonds as edges. The number of nodes of each molecule is different and each has its own molecular structure. The input data set in graph classification or regression is a set of pairs of such individual graph and the feature defined on the graph nodes.

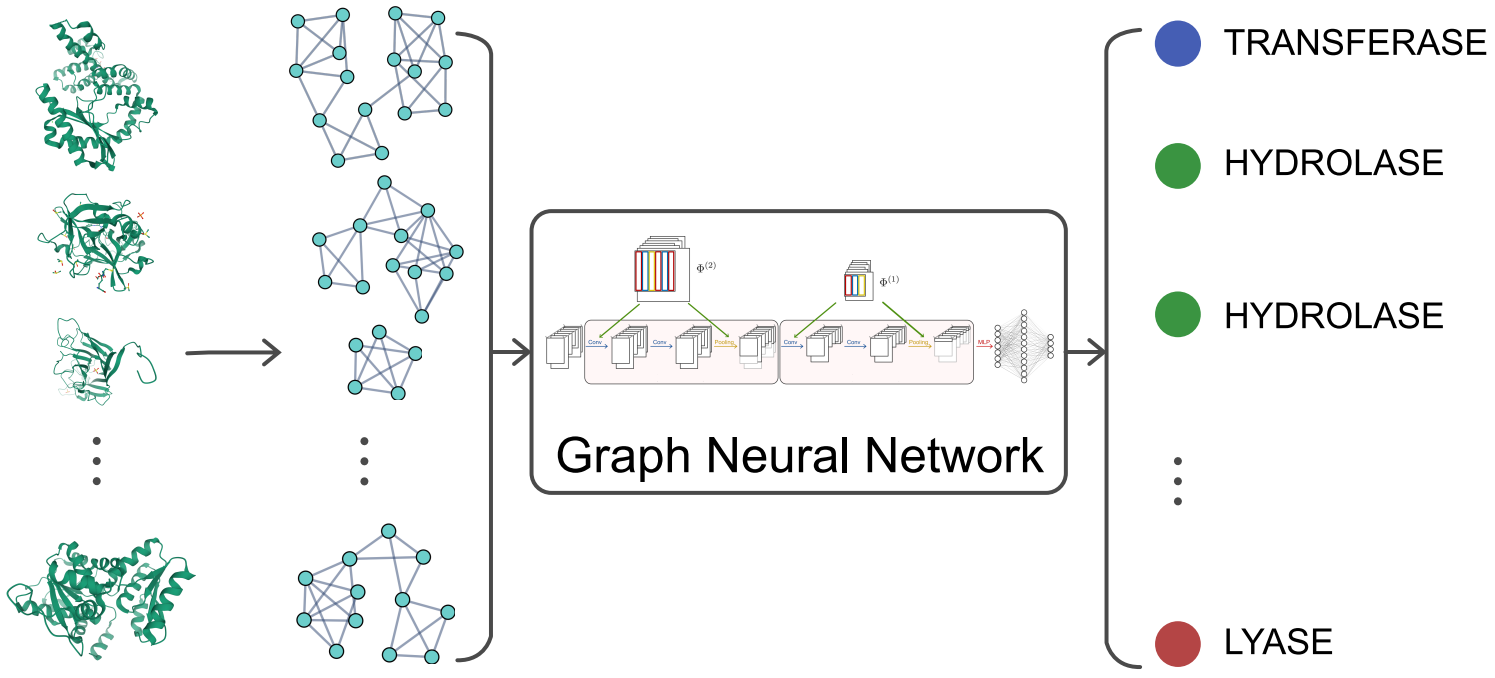
What problems can Graph Neural Networks solve?



Node property prediction (node classification/regression)

- The input is one graph.
- The GNN uses the known labeled nodes and the graph edges to infer the missing labels.

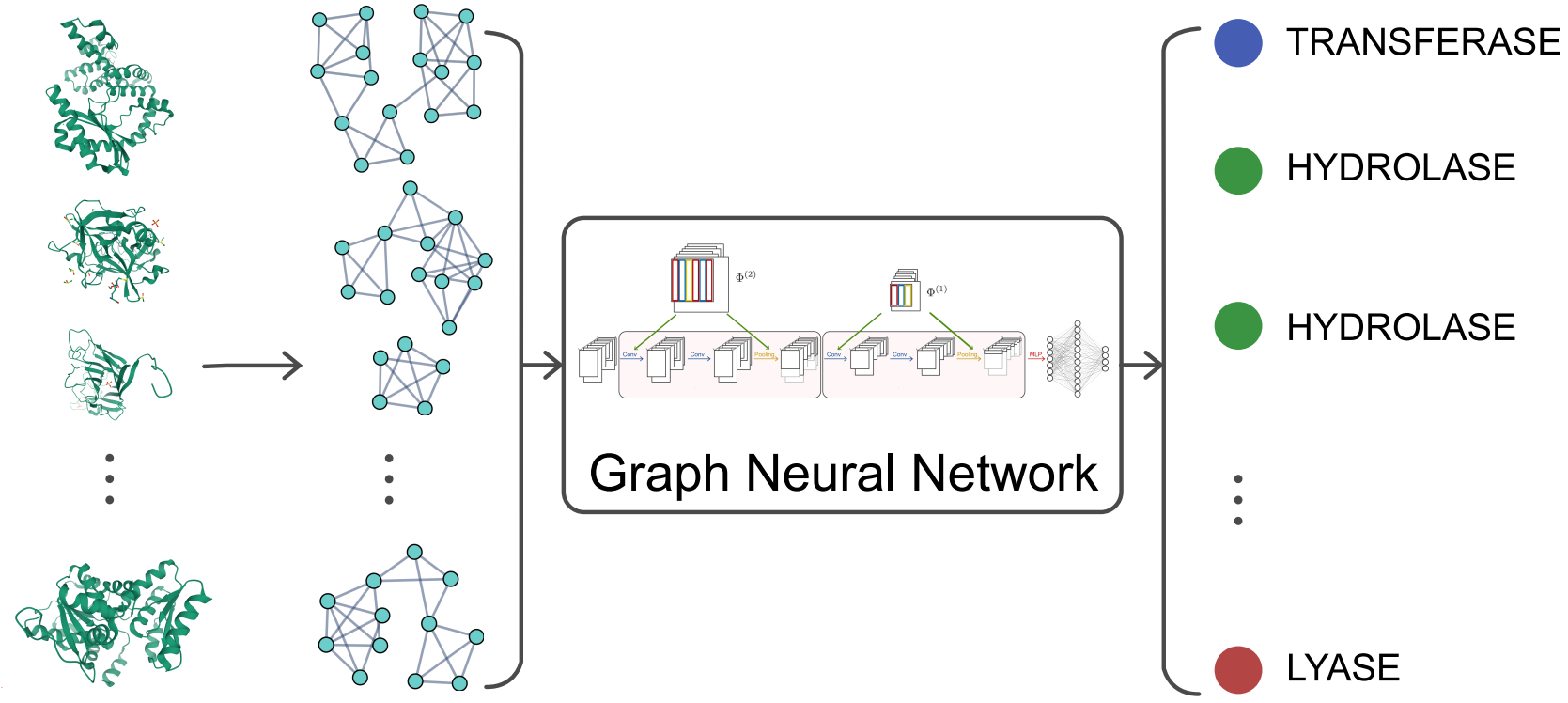
What problems can Graph Neural Networks solve? (Continued)



Graph property prediction (graph classification/regression)

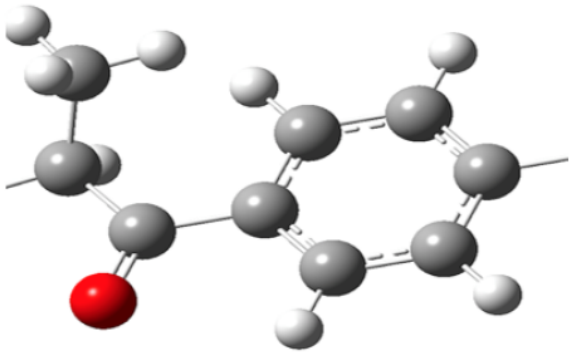
- The input is a set of graphs. Each graph sample has a Y-label.
- The graph samples with known Y-labels and features on the graphs are used in training for GNN. The trained GNN model can be used to infer the missing labels.

Graph Classification for Enzymes



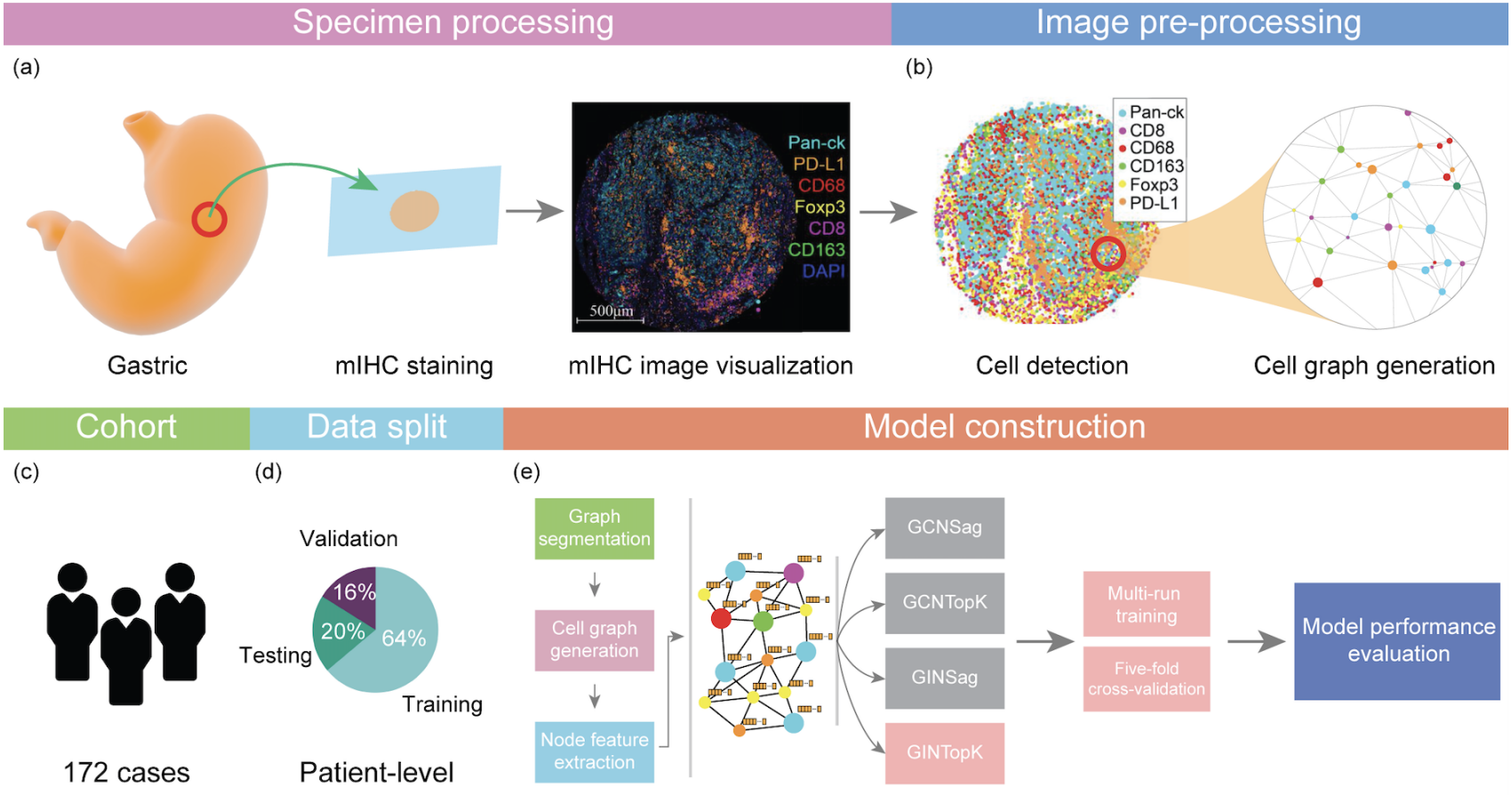
GNNs for labeling enzymes. A set of graphs is extracted as the input data set, where each graph represents a specific protein tertiary structure. The task is to assign each enzyme instance to one of the given EC top-level classes correctly. The GNN's role is to find universally applicable rules to label the graphs by learning the topological and feature information of the input. The structure of enzymes is retrieved from the Protein Data Bank.

Quantum Chemistry Graph Regression



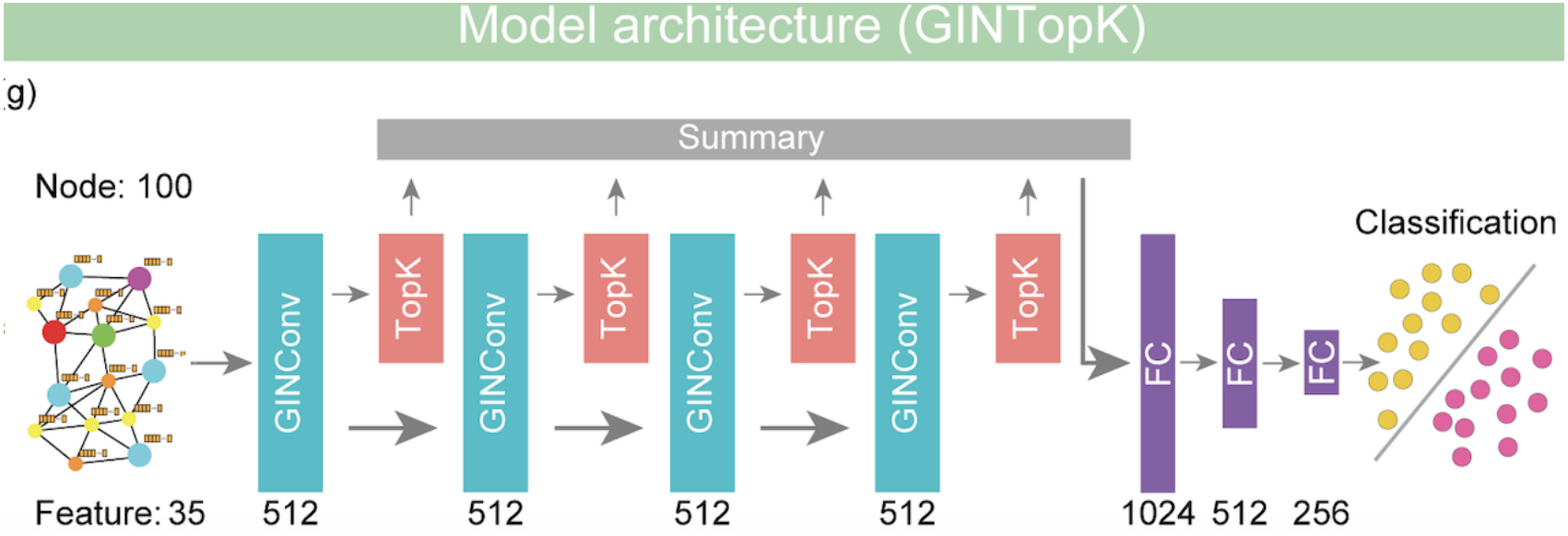
- QM7 is a collection of $7,165$ molecules, train/test $= 4/1$.
- Each molecule contains ≤ 23 atoms (including C, O, N, S), atoms are connected by bonds, molecular structure varies (e.g. double/triple bonds, cycles, carboxy, cyanide ...).
- Molecule is a graph, atoms are nodes, bonds are edges and Coulomb energy as weights, then Coulomb energy matrix is adjacency matrix.
- Task: to predict atomization energy of molecule given the molecular structure.

Cancer Diagnostics by Cell Graphs



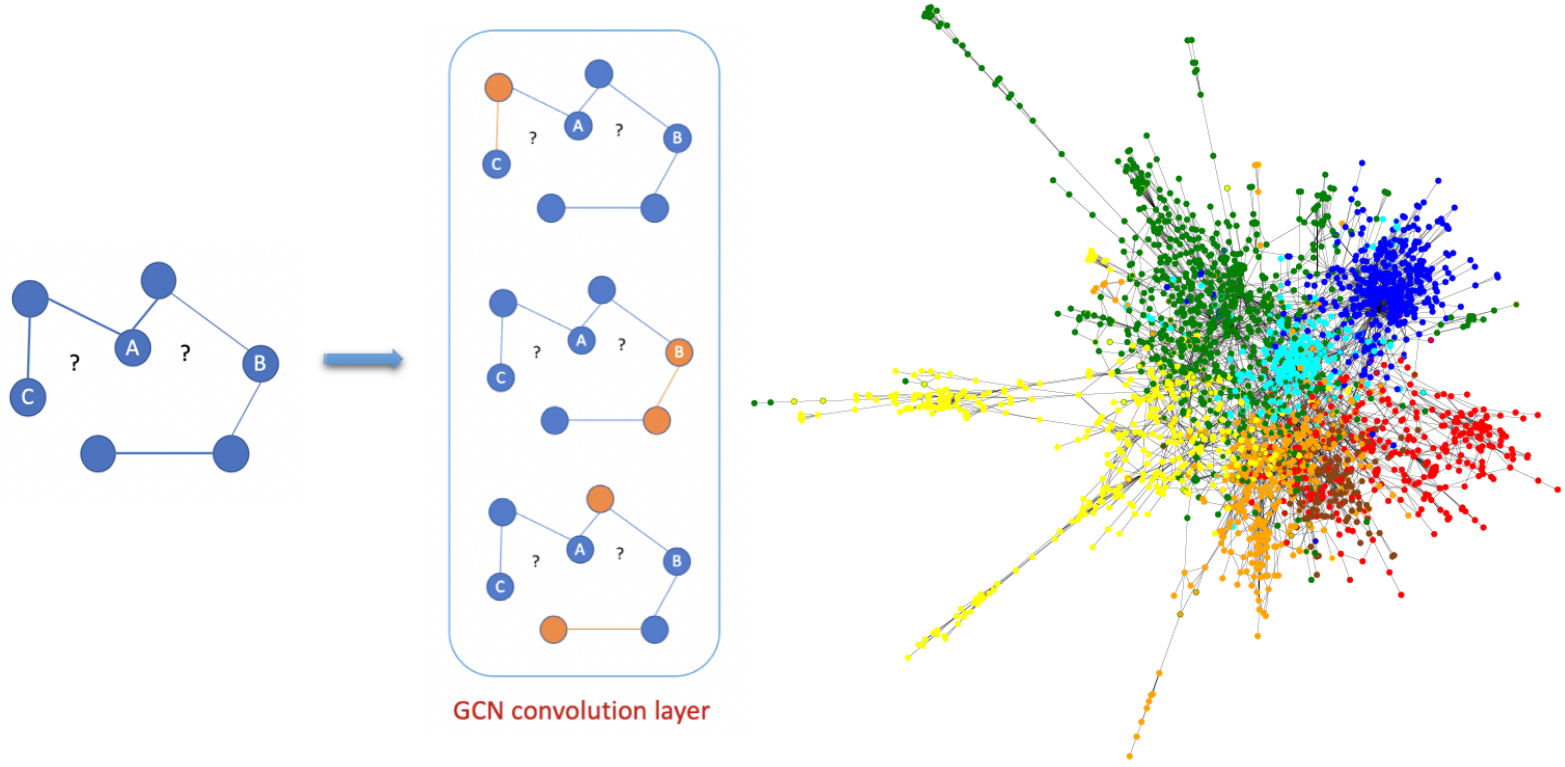
Staining image of cells from Gastric slice is segmented into cell graphs by spatial information (reflecting tumor microenvironment) of cells to generate multiple cell graph samples. Y-label is survival time of a patient (and cell graph). The GNN is trained to infer the survival time of each patients.

Graph Neural Networks for Graph Property Prediction



A GNN here is a feedforward NN. It is typically composed of multiple graph convolution layers, graph pooling layers, and fully connected layers. In this example of Graph Classification, there are around 17,000 graph samples with different size. Each Y-label is a class label. In the figure, the input is a graph with 100 nodes. Each node has 35 features. Computational flow of a Graph Neural Network consisting of 4 blocks of GINConv plus TopKPool layers, followed by an FC's.

What makes Graph Neural Networks function?



Spectral graph convolution $g \star f = U\big((U^T g) \odot (U^T f)\big)$

- Motivated by traditional Fourier convolution
- Have similar function as convolution in CNN
- Simplified example is GCNConv.
- It would preserve the structural information of graph data input.

Graph Convolution Does Not Change Graph Size in GNN

- A typical example is the widely used GCNConv, proposed by *Kipf & Welling (2017)*.

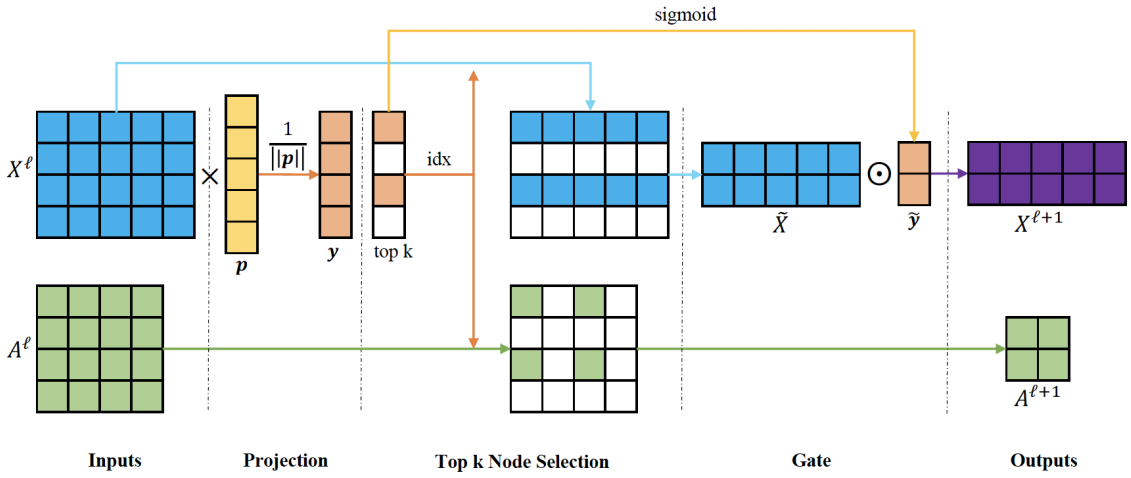
$$X^{\text{out}} = \sigma(\widehat{A}X^{\text{in}}W).$$

- Here $\widehat{A} = \widetilde{D}^{-1/2}(A + I)\widetilde{D}^{-1/2} \in \mathbb{R}^{N\times N}$ is a normalized version of the adjacency matrix A of the input graph, where I is the identity matrix and $\widetilde{D}$ is the degree matrix for $A + I$.
- Further, $X^{\text{in}} \in \mathbb{R}^{N\times d}$ is the array of d-dimensional features on the N nodes of the graph, and $W \in \mathbb{R}^{d\times m}$ is the filter parameter matrix. The output $X^{\text{out}} \in \mathbb{R}^{N\times d'}$ has the same first dimension as the input. Convolution does not change number of nodes.

How GNNs handle input graphs with varying number of nodes and connectivity structures?

- One way is to utilize graph pooling. It is a computational strategy to reduce the number of graph nodes while preserve as much as geometric information of the original input graph data; in this way, one has a unified graph-level representation for graph-structured data when the size and topology of an individual graph are changing.

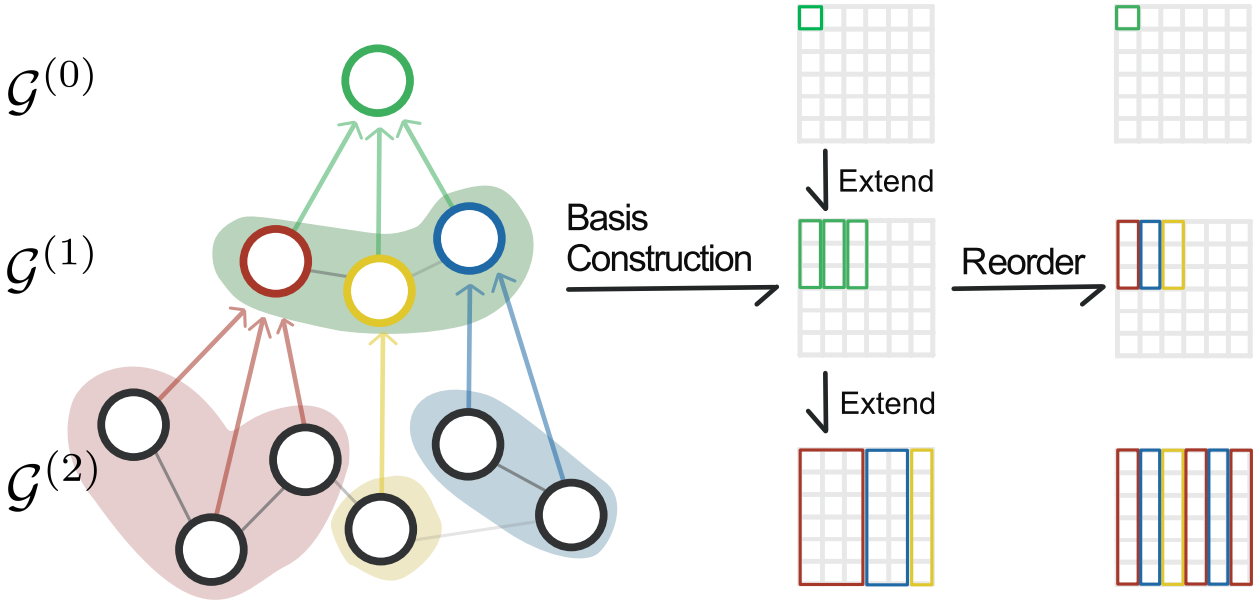
Existing Graph Pooling Methods



TopKPooling *Gao & Ji 2019, Knyazev et al 2019, Cangea, Liò et al 2018.*

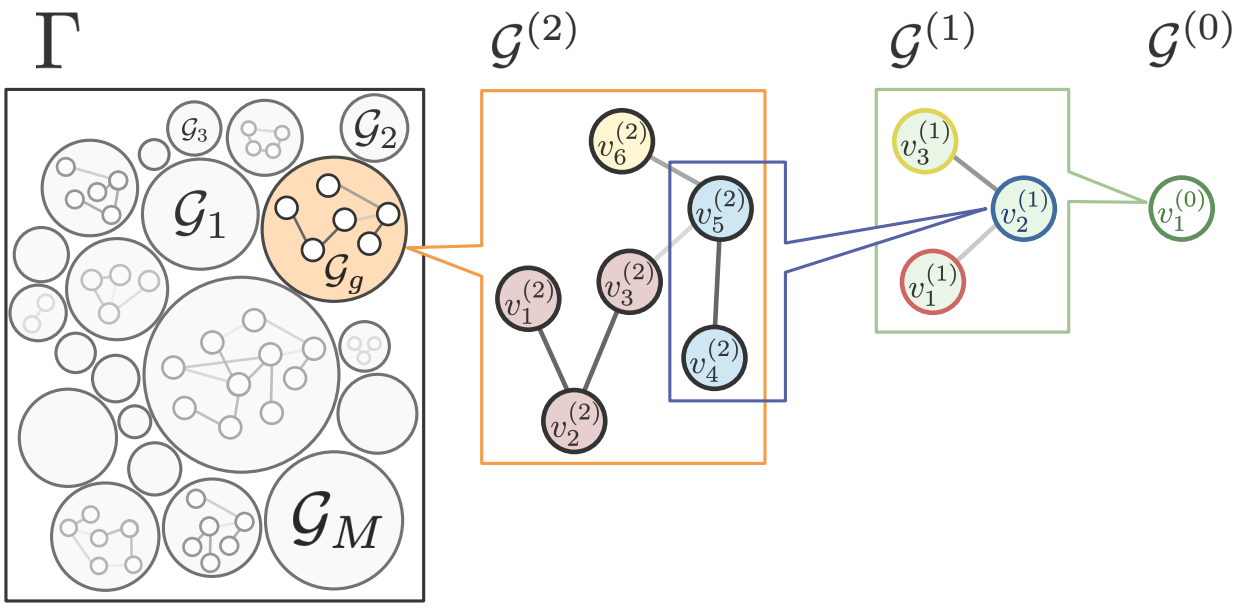
- Spatial Method
  - Global (Non-hierarchical) pooling:
    ChebNet, Set2Set, SortPool, MPNN
  - Hierarchical pooling:
    DiffPool, TopKPool, SAGPool, EdgePool
- Spectral Method
  - LaPool, EigenPool, SOPool

Haar Graph Pooling



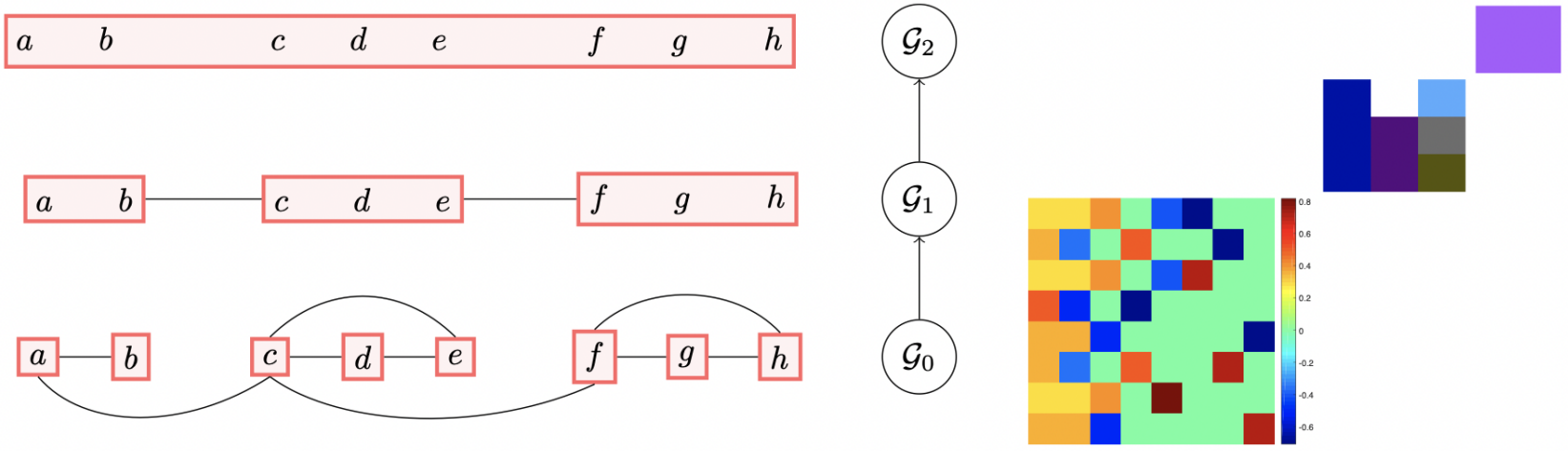
- HaarPool provides cascading pooling layers, i.e., for each layer, we define an orthonormal Haar wavelet basis and its compressive Haar transform.
- Each HaarPool layer pools the graph input from the previous layer to output with a smaller node number and the same feature dimension.
- In this way, all the HaarPool layers together synthesize the features of all graph input samples into feature vectors with the same size. We then obtain an output of a fixed dimension, regardless of the size of the input.

Haar Graph Pooling Components: Chain



HaarPool is a hierarchically structured algorithm, and has a global design. The coarse-grained chain determines the hierarchical relation in different HaarPool layers. The node number of each HaarPool layer is equal to the number of nodes of the subgraph of the corresponding layer of the chain. As the top-level of the chain can have one node, the HaarPool finally reduces the number of nodes to one, thus producing a fixed dimensional output in the last HaarPool layer. Chain is obtained by clustering the graph using like spectral/METIS/k-means clustering.

Haar Graph Pooling Components: Haar Wavelet Basis



Gavish et al. (2010), Chui et al. (2015).

The HaarPool uses the sparse Haar representation on chain structure. In each subgraph of chain level, we generate a Haar wavelet basis.

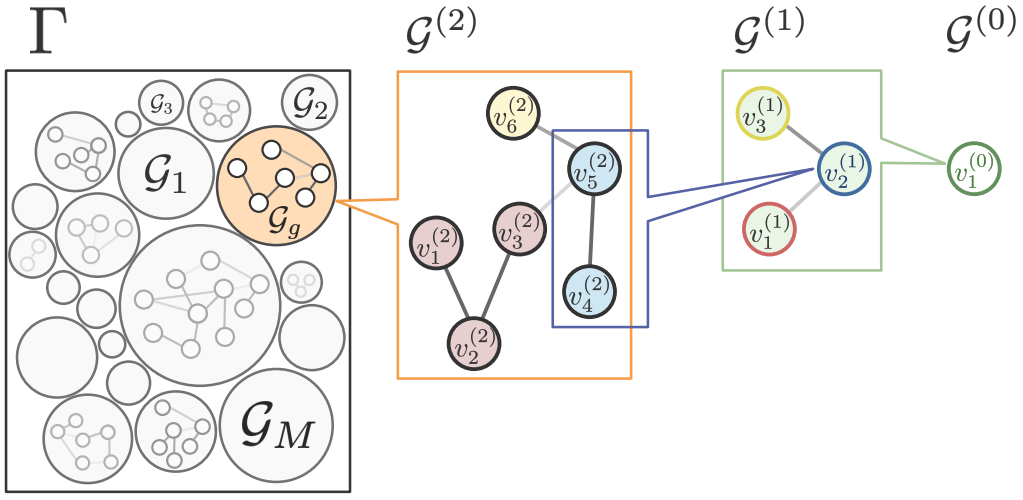
In each HaarPool layer, the representation then combines the features of input X_j^{in} with the structural information of the graphs of the jth and $(j+1)$th layers of the chain.

By the property of the Haar basis, the HaarPool only drops the high-frequency information of the input data. The X_j^{out} mirrors the low-frequency information in the Haar wavelet representation of X_j^{in}. Thus, HaarPool preserves the essential information of the graph input, and the network has small information loss in pooling.

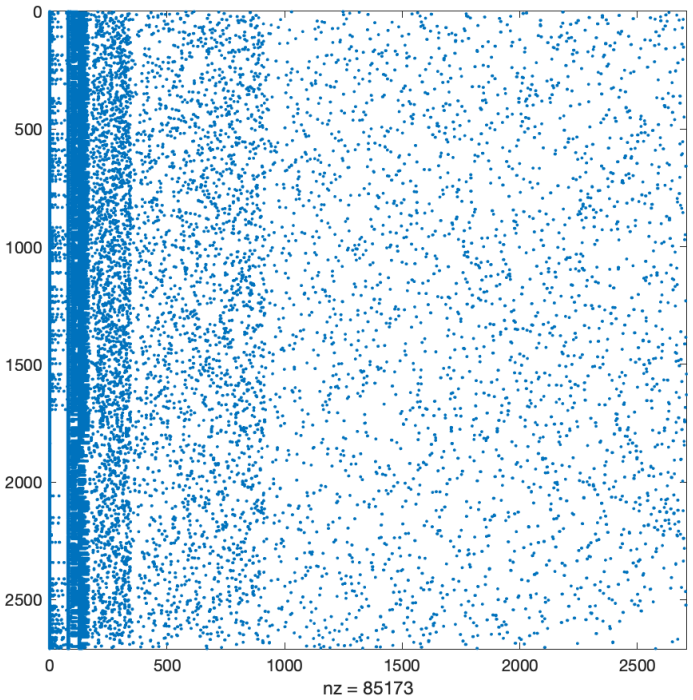
Orthogonality and Locality of Haar Basis

- $\langle \phi_\ell, \phi_{\ell'} \rangle = \delta_{\ell,\ell'}$
  For each level $j = J_0, \ldots, J$, the $\{\phi_\ell^{(j)}\}_{\ell=1}^{N_j}$ is an orthonormal basis for $l_2(\mathcal{G}_j)$, and in particular, $\{\phi_\ell^{(J)}\}_{\ell=1}^{N}$ is an orthonormal basis for $l_2(\mathcal{G})$ and the chain. (Effective Representation)

- If each parent of level $\mathcal{G}_j$, $j = J-1, J-2, \ldots, J_0$, contains at least two children, the number of different values of the Haar basis ϕ_ℓ, $\ell = 1, \ldots, N$, is bounded by a constant. For example, a binary tree. It thus has Fast Algorithms in $\mathcal{O}(N)$.

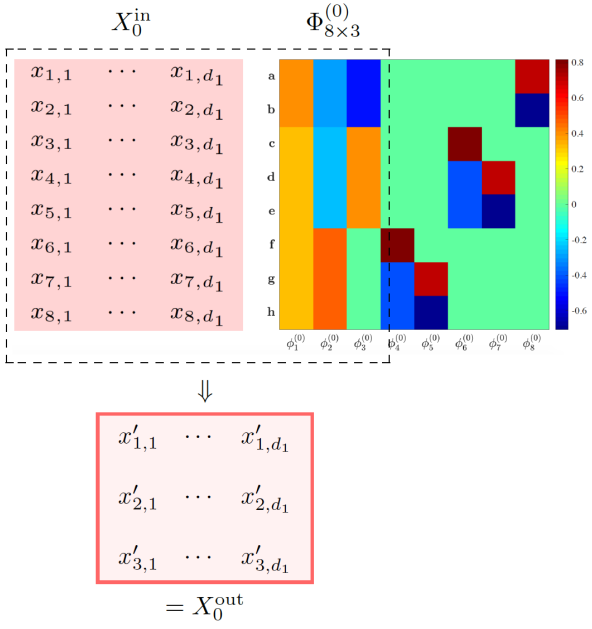


Sparsity of Haar Basis Matrix

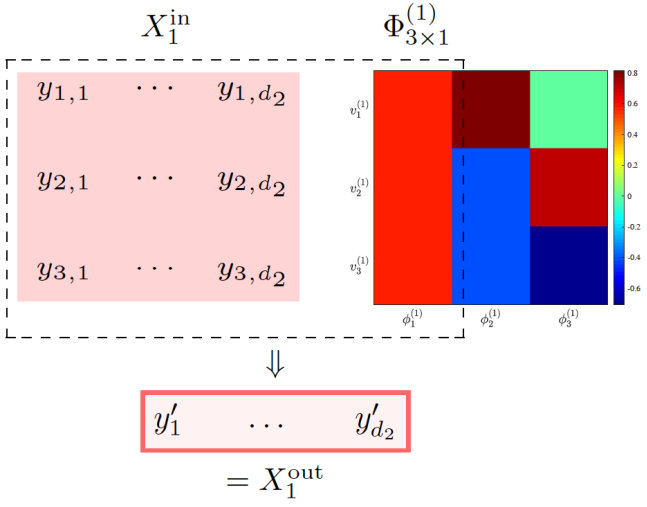


- Haar Basis for Cora
- Citation network Cora: 2708 nodes, 5429 edges
- Chain by METIS
- Sparsity: 98.84%

Computing Strategy of HaarPool



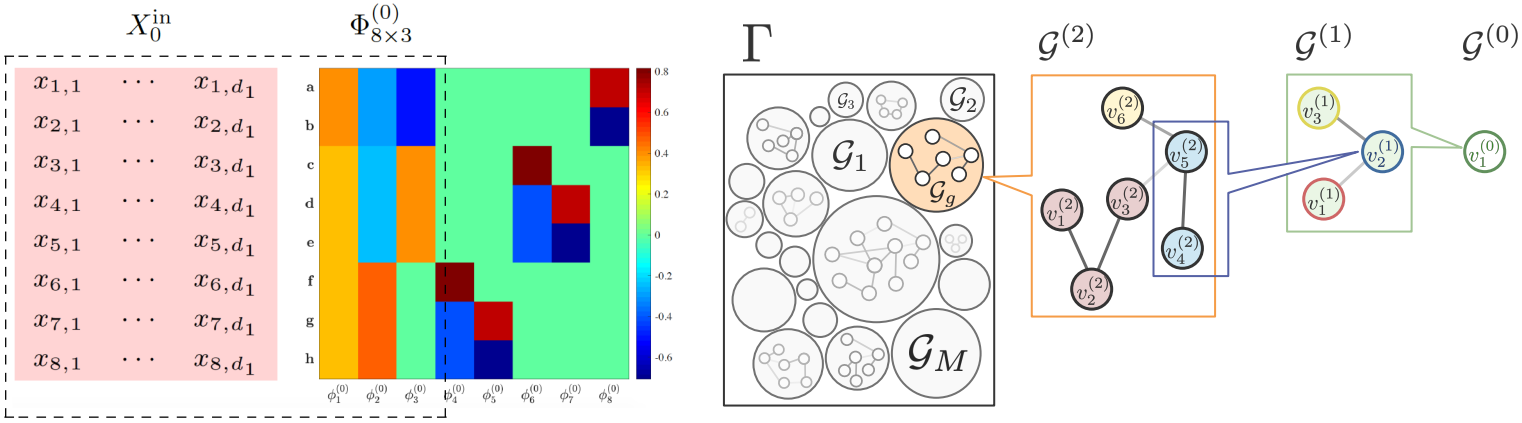
(a) First HaarPool Layer for $\mathcal{G}_0 \to \mathcal{G}_1$.



(b) Second HaarPool Layer for $\mathcal{G}_1 \to \mathcal{G}_2$.

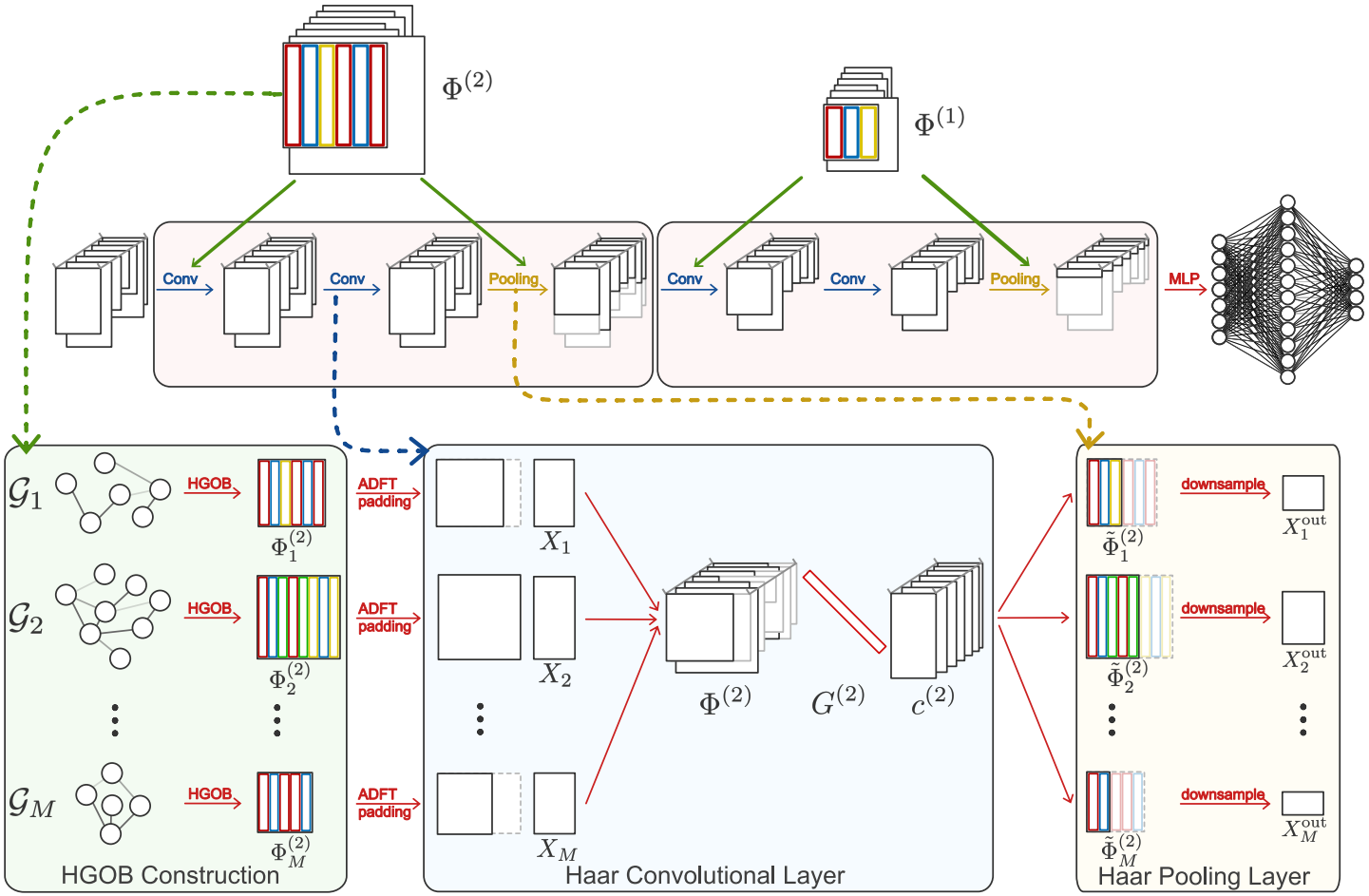
- In the first layer, the input X_1^{in} of size $8 \times d_1$ is transformed by the compressive Haar basis matrix $\Phi_{8\times 3}^{(0)}$ that consists of the first 3 column vectors of the full Haar basis $\Phi_{8\times 8}^{(0)}$ in (a), and the output is a $3 \times d_1$ matrix X_1^{out}.
- This example shows that the HaarPool amalgamates the node feature by adding the same weight to the nodes that are in the same cluster of the coarser layer, and in this way, pools the feature using the graph clustering information.

Compressive Haar Transforms



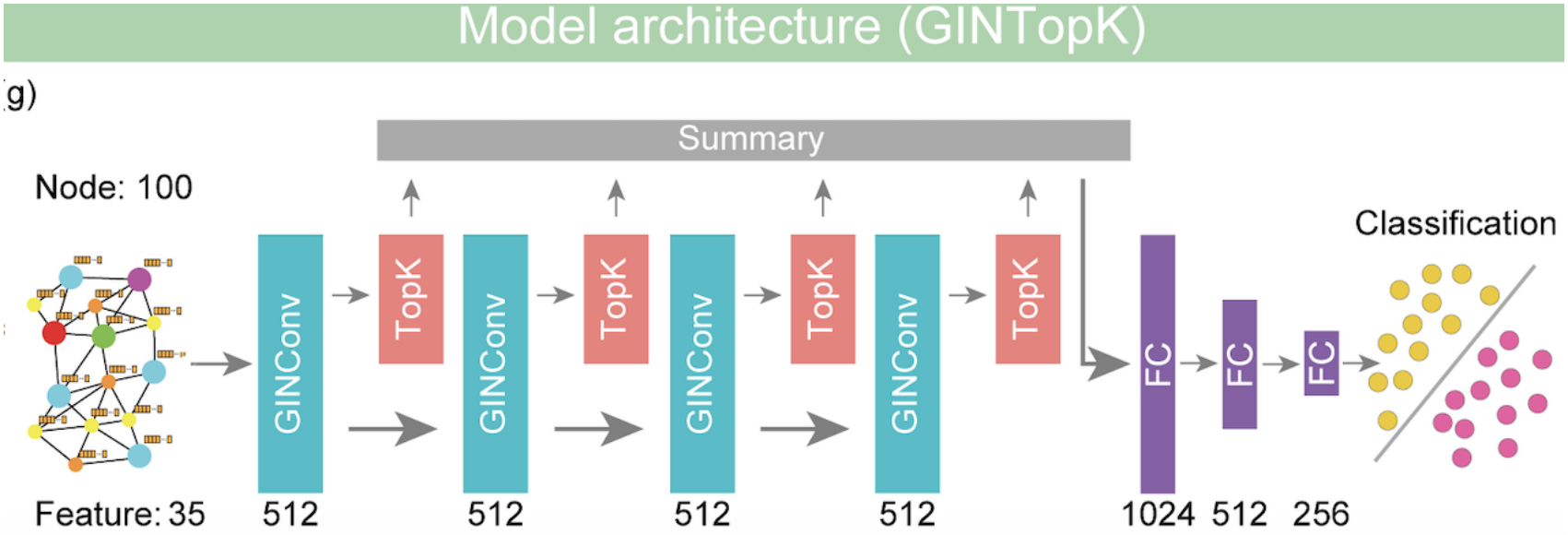
- We preserve the main (approximation) information of the graph signal.
- Reduce the dimension of the graph each time we use compressive Haar transforms.
- Due to the sparsity of the Haar basis matrix and then fast Haar transforms, the HaarPool is computationally feasible.

Graph Neural Networks with HaarPool: Architecture



- Graph neural network the similar architecture as the traditional convolutional neural network.
- Input data is a graph, including adjacency matrix and weighted nodes.

Network Architecture of HaarPool based GNNs



The network architecture varies on different tasks and datasets. The architecture of GNN is identified by the layer type and the number of hidden nodes at each layer. For example, the above GNN uses a 4 layers of GINConv and TopKPool layers, each with 512 hidden nodes, plus 3 fully connected layers with 1024, 512 and 256 neurons.

HaarPool for Benchmark Graph Classification

We include datasets from different domains, samples, and graph sizes to give a comprehensive understanding of how the HaarPool performs with datasets in various scenarios. Table 1 summarizes some statistical information of the datasets: each dataset containing graphs with different sizes and structures, the number of data samples ranges from 188 to 4,337, the average number of nodes is from 17.93 to 39.06, and the average number of edges is from 19.79 to 72.82.

Table 1. Summary statistics of the graph classification datasets.

Dataset	MUTAG	PROTEINS	NCI1	NCI109	MUTAGEN
max #nodes	28	620	111	111	417
min #nodes	10	4	3	4	4
avg #nodes	17.93	39.06	29.87	29.68	30.32
avg #edges	19.79	72.82	32.30	32.13	30.77
#graphs	188	1,113	4,110	4,127	4,337
#classes	2	2	2	2	2

- We compare **HaarPool** with **SortPool** *Zhang et al., (2018).* , **DiffPool** *Ying et al., (2018).* , **gPool** *Gao & Ji, (2019).* , **SAGPool** *Lee et al., (2019).* , **EigenPool** *Ma et al., (2019).* , **CSM** *Kriege & Mutzel, (2012).* and **GIN** *Xu et al., (2019).* on the above datasets.

HaarPool for Graph Classification (TUDataset)

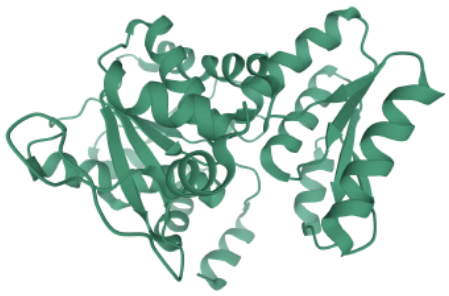


Table 2 reports the classification test accuracy. GNNs with HaarPool have excellent performance on all datasets. In 4 out of 5 datasets, it achieves top accuracy. It shows that HaarPool, with an appropriate graph convolution, can achieve top performance on a variety of graph classification tasks, and in some cases, improve state of the art by a few percentage points.

Table 2. Performance comparison for graph classification tasks (test accuracy in percent, showing the standard deviation over ten repetitions of the experiment).

Method	MUTAG	PROTEINS	NCI1	NCI109	MUTAGEN
CSM	85.4	–	–	–	–
GIN	89.4	76.2	**82.7**	–	–
SortPool	85.8	75.5	74.4	72.3*	78.8*
DiffPool	–	76.3	76.0*	74.1*	80.6*
gPool	–	77.7	–	–	–
SAGPool	–	72.1	74.2	74.1	–
EigenPool	–	76.6	77.0	74.9	79.5
HaarPool (ours)	**90.0±3.6**	**80.4±1.8**	78.6±0.5	**75.6±1.2**	**80.9±1.5**

'*' indicates records retrieved from EigenPool (Ma et al., 2019a), '–' means that there are no public records for the method on the dataset, and bold font is used to highlight the best performance in the list.

HaarPool for QM7

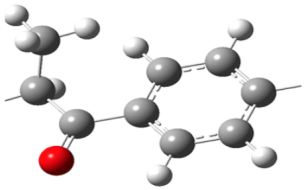
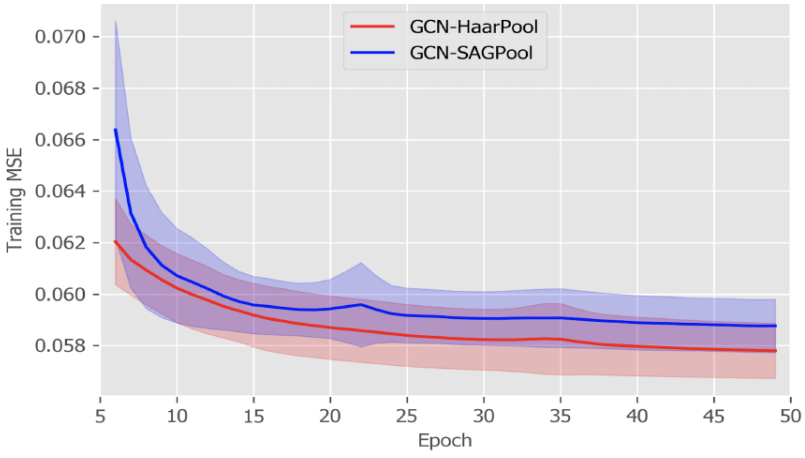


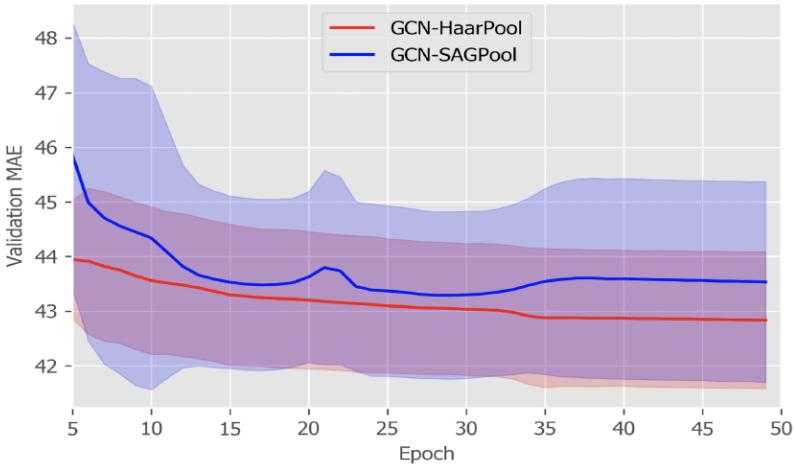
Table 5 shows the results for GCN-HaarPool and GCN-SAGPool, with the public results of the other methods from *Wu et al. (2018)*. Compared to the GCN-SAGPool, the GCN-HaarPool has a lower average test MAE and a smaller SD and ranks the top in the table. Given the simple architecture of our GCN-HaarPool model, we can interpret the GCN-HaarPool an effective method.

Table 5. Test mean absolute error (MAE) comparison on QM7, with the standard deviation over ten repetitions of the experiments.

Method	Test MAE
RF	122.7 ± 4.2
Multitask	123.7 ± 15.6
KRR	110.3 ± 4.7
GC	77.9 ± 2.1
GCN-SAGPool	43.3 ± 1.6
GCN-HaarPool (ours)	$\mathbf{42.9 \pm 1.2}$

Loss MSE and Validation MAE





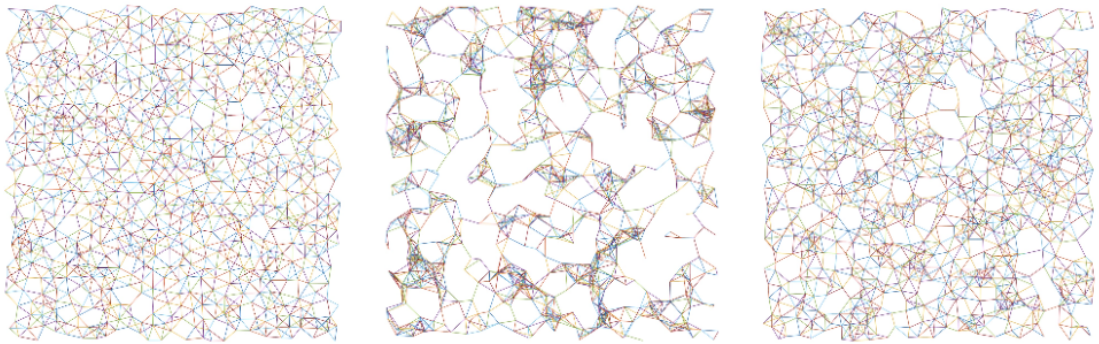
We present the mean and SD of the training MSE loss (for normalized input) and the validation MAE (which is in the original label domain) versus the epoch. It illustrates that the learning and generalization capabilities of the GCN-HaarPool are better than those of the GCN-SAGPool; in this aspect, HaarPool provides a more efficient graph pooling for GNN in this graph regression task.

HaarPool for PointPattern Classification

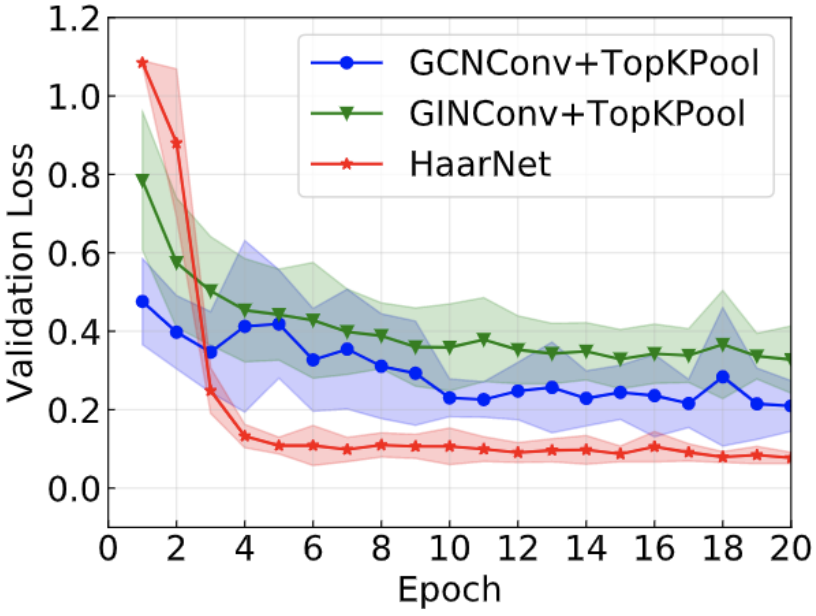
PointPattern dataset is constructed by simple point patterns from statistical mechanics. We simulated three point patterns in 2D: hard disks in equilibrium (HD), Poisson point process, and random sequential adsorption (RSA) of disks. The HD and Poisson distributions can be seen as simple models that describe the microstructures of liquids and gases, while the RSA is a nonequilibrium stochastic process that introduces new particles one by one subject to nonoverlapping conditions. For each point pattern, the particles are treated as nodes, and edges are subsequently drawn according to whether two particles are within a threshold distance.

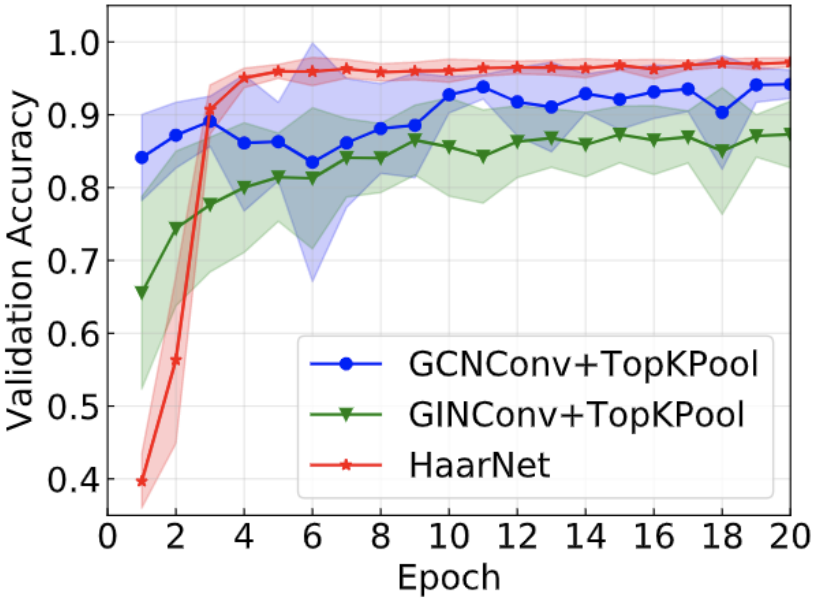
Table 1: Summary information of PointPattern datasets.

PointPattern	$\phi_{\text{RSA}} = 0.3$	$\phi_{\text{RSA}} = 0.35$	$\phi_{\text{RSA}} = 0.4$
#classes	3	3	3
#graphs	15,000	15,000	15,000
max #nodes	1000	1000	1000
min #nodes	100	100	100
avg #nodes	478	474	475



HaarPool for PointPattern: Validation





We compare HaarConv+HaarPool with the baselines GCNConv+TopKPool and GINConv+TopKPool. The figure shows that HaarPool case obtains much smaller variations in term of validation loss and validation accuracy. The lower validation loss, higher validation accuracy and higher test accuracy indicate the superior learning and generalization abilities.

Computational Complexity

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.

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.

Discussion

- HaarPool combines the features and structural information of a graph by compressive Haar transforms.
- Wavelet analysis and graph signal processing are incooperated in GNNs.
- There is a a lot of potentials for interaction between GNNs and Signal Processing to probe!

Thank you!

References

- Haar Graph Pooling. *ICML 2020*
- Path Integral Based Convolution and Pooling for Graph Neural Networks. *NeurIPS 2020*
- Graph Neural Networks with Haar Transform-Based Convolution and Pooling: A Complete Guide. *arXiv:2007.11202*
- Fast Haar Transforms for Graph Neural Networks. *Neural Networks 2020*

Codes YuGuangWang@Github

Fast Haar Transforms

Adjoint Haar Transform $\Phi^T f$

1. Evaluation of the following sums for $j = J_0, \ldots, J-1$.

$$\mathcal{S}^{(j)}(f, v_k^{(j)}), \quad v_k^{(j)} \in V_j.$$

2. For each ℓ, let j be the integer such that $N_j + 1 \le \ell \le N_{j-1}$, where $N_J := 0$. Evaluating

$$\sum_{k=1}^{N_j} \mathcal{S}^{(j)}(f, v_k^{(j)}) w_k^{(j)} \phi_\ell^{(j)}(v_k^{(j)})$$

by the following two steps.

(a) Compute the product for all $v_k^{(j)} \in V_j$:

$$T_\ell(f, v_k^{(j)}) = \mathcal{S}^{(j)}(f, v_k^{(j)}) w_k^{(j)} \phi_\ell^{(j)}(v_k^{(j)}).$$

(b) Evaluate sum $\sum_{k=1}^{N_j} T_\ell(f, v_k^{(j)})$.

Forward Haar Transform Φc

1. For each ℓ, j is the integer s.t. $N_j + 1 \le \ell \le N_{j-1}$, $N_J := 0$. For all $k = 1, \ldots, N_j$, compute the product

$$t_\ell(c, v_k^{(j)}) := c_\ell \phi_\ell^{(j)}(v_k^{(j)}).$$

2. For each $j = J_0, \ldots, J$, evaluate the sums

$$s(c, v_{k_j}^{(j)}) := \sum_{\ell=N_j+1}^{N_{j-1}} t_\ell(c, v_{k_j}^{(j)}).$$

3. Compute the $W_k^{(j)}$ for $k = 1, \ldots, N$ and $j = J_0, \ldots, J-1$.

4. Compute the weighted sum

$$(\Phi c)_k = \sum_{j=J_0}^{J} W_k^{(j)} s(c, v_{k_j}^{(j)})$$

for $k = 1, \ldots, N$.

- nearly linear computational complexity $\mathcal{O}\left(N(\log N)^2\right)$
- speed up to $\mathcal{O}\left(k \log N\right)$ by sparse FFTs, $k =$ #non-zeros
  Hassanieh et al. (2012).

Fast Haar Transforms (Continued)

- Let $\{\phi_\ell\}_{\ell=1}^N$ be the Haar basis for the filtration $(\mathcal{G}_{J_0\to J}, W_{J_0\to J})$ of a graph $\mathcal{G}$. We define the weighted sum for $f \in l_2(\mathcal{G})$ by

$$\mathcal{S}^{(J_0)}\big(f, v_k^{(J_0)}\big) := f(v_k^{(J_0)}), \quad v_k^{(J_0)} \in \mathcal{G}_{J_0},$$

and for $j = J_0+1, \ldots, J$ and $v_k^{(j)} \in \mathcal{G}_j$,

$$\mathcal{S}^{(j+1)}\big(f, v_k^{(j+1)}\big) := \sum_{v_{k'}^{(j)} \in v_k^{(j+1)}} w_{k'}^{(j)} \mathcal{S}^{(j)}\big(f, v_{k'}^{(j)}\big).$$

- For $j = J_0+1, \ldots, J$, let $c_k^{(j)}$ be the number of children of $v_k^{(j)}$ for $k = 1, \ldots, N_j$. For $j = J_0$, let $c_k^{(J_0)} \equiv 1$ for $k = 1, \ldots, N$. For $j = J_0, \ldots, J$ and $k = 1, \ldots, N_j$, the weight factor for $v_k^{(j)}$ is $w_k^{(j)} := 1/\sqrt{c_k^{(j)}}$. $W_k^{(J_0)} := 1$ and

$$W_k^{(j)} := \prod_{n=J_0+1}^{j} w_{k_n}^{(n)} \text{ for } j = J_0+1, \ldots, J.$$

Extracting Structural Information by Graph Convolution

- Spectral-based Graph Convolution

$$g \star f = U\big((U^T g) \odot (U^T f)\big)$$

- an un-directed weighted graph $\mathcal{G} = (V, E, w)$
- U is the matrix of orthonormal basis for $l_2(\mathcal{G})$.
- $\odot$ is the element-wise Hadamard product.
- This spectral-based convolution is a generalization of convolution of Fourier analysis in $\mathbb{R}^d$. This is one major way of defining convolution for graph neural networks (GNNs), which would preserve the structural information of graph data input.

Graph Convolution by Graph Laplacian

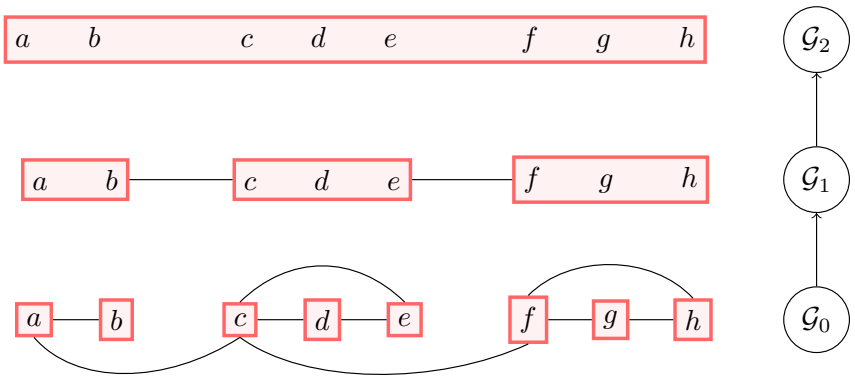
Bruna et al. (2014).

- By graph Laplacian

$$g \star f = U\big((U^T g) \odot (U^T f)\big)$$

- U is the matrix of eigenvectors of graph Laplacian $\mathcal{L} = D - A$, which reflects the neighbourhood information.
- $U^T f$ and Uc are the adjoint and forward discrete Fourier transforms of f and c on $\mathcal{G}$.
- Computing U, $U^T f$ and $U f$ is expensive. (In general, there is no FFTs on graph as U is non-sparse and graph nodes are non-regular.)

Chain

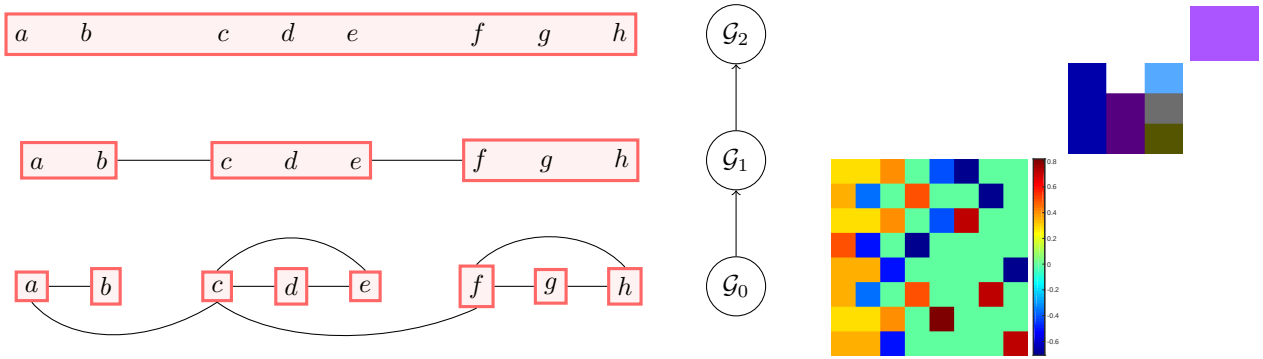


- Based on chain

$$\mathcal{G}_{J_0 \to J} = (\mathcal{G}_{J_0}, \ldots, \mathcal{G}_J)$$

- Chain by clustering methods, spectral clustering, k-means, METIS

Construction of Haar Basis



Gavish et al. (2010), Chui et al. (2015).

- Haar basis is constructed from top to bottom, $\ell \le N^{(1)}$

$$\phi_1^{(2)}(u^{(2)}) = 1, \quad \phi_1^{(1)}(u^{(1)}) = \mathbf{1}(u^{(1)})/\sqrt{N_1}$$

$$\phi_\ell^{(1)}(u^{(1)}) = \sqrt{\frac{N^{(1)}-\ell+1}{N^{(1)}-\ell+2}}\left(\chi_{\ell-1}^{(1)}(u^{(1)}) - \frac{\sum_{j=\ell}^{N^{(1)}}\chi_j^{(1)}(u^{(1)})}{N^{(1)}-\ell+1}\right).$$

- Extend to layer $\mathcal{G}^{(0)}$: for $k = 2, \ldots, k_\ell$, $k_\ell = |u_\ell^{(1)}|$, we let

$$\phi_{\ell,1}(v) := \frac{\phi_\ell^{(1)}(v^{(1)})}{\sqrt{|v^{(1)}|}}, \quad \phi_{\ell,k} = \sqrt{\frac{k_\ell-k+1}{k_\ell-k+2}}\left(\chi_{\ell,k-1} - \frac{\sum_{j=k}^{k_\ell}\chi_{\ell,j}}{k_\ell-k+1}\right)$$

where $\chi_{\ell,j}$ for $j = 1, \ldots, k_\ell$, $\chi_{\ell,j}$ is the indicator function on $\{v_{\ell,j}\}$.

Haar Graph Pooling Components: Haar Wavelet Basis

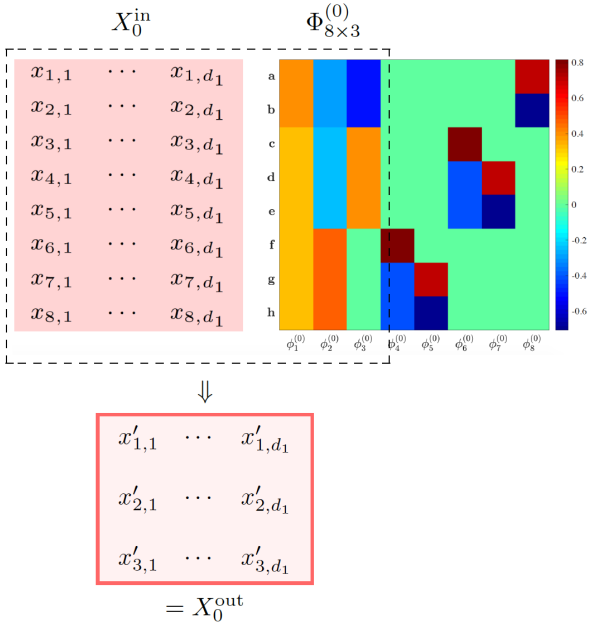
$$X_j^{\text{out}} = \Phi_j^T X_j^{\text{in}}, \quad j = 0, 1, \ldots, K-1.$$

First, the HaarPool is a hierarchically structured algorithm, and has a global design. The coarse-grained chain determines the hierarchical relation in different HaarPool layers. The node number of each HaarPool layer is equal to the number of nodes of the subgraph of the corresponding layer of the chain. As the top-level of the chain can have one node, the HaarPool finally reduces the number of nodes to one, thus producing a fixed dimensional output in the last HaarPool layer.

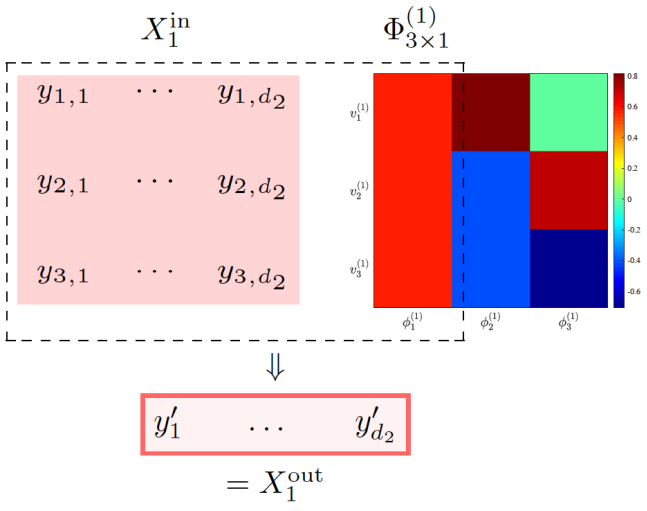
The HaarPool uses the sparse Haar representation on chain structure. In each HaarPool layer, the representation then combines the features of input X_j^{in} with the structural information of the graphs of the jth and $(j+1)$th layers of the chain.

By the property of the Haar basis, the HaarPool only drops the high-frequency information of the input data. The X_j^{out} mirrors the low-frequency information in the Haar wavelet representation of X_j^{in}. Thus, HaarPool preserves the essential information of the graph input, and the network has small information loss in pooling.

Computing Strategy of HaarPool



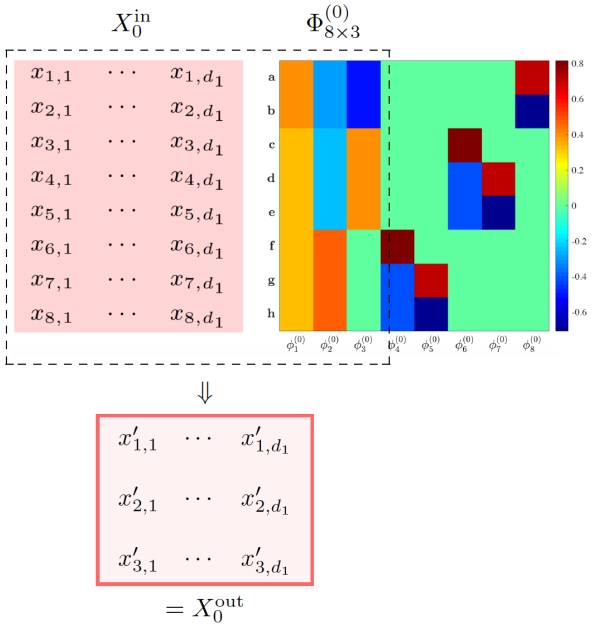
(a) First HaarPool Layer for $\mathcal{G}_0 \to \mathcal{G}_1$.



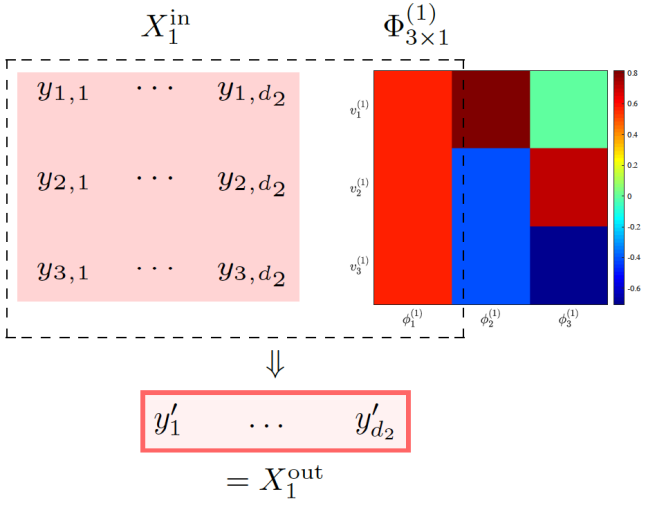
(b) Second HaarPool Layer for $\mathcal{G}_1 \to \mathcal{G}_2$.

- In the first layer, the input X_1^{in} of size $8 \times d_1$ is transformed by the compressive Haar basis matrix $\Phi_{8\times3}^{(0)}$ that consists of the first 3 column vectors of the full Haar basis $\Phi_{8\times8}^{(0)}$ in (a), and the output is a $3 \times d_1$ matrix X_1^{out}.
- In the second layer, the input X_2^{in} of size $3 \times d_2$ (usually X_1^{out} followed by convolution) is transformed by the compressive Haar matrix $\Phi_{3\times1}^{(1)}$, which is the first column vector of the full Haar basis matrix $\Phi_{3\times3}^{(1)}$ in (b).

Computing Strategy of HaarPool (Continued)



(a) First HaarPool Layer for $\mathcal{G}_0 \to \mathcal{G}_1$.



(b) Second HaarPool Layer for $\mathcal{G}_1 \to \mathcal{G}_2$.

- By the construction of the Haar basis in relation to the chain, each of the first three column vectors $\phi_1^{(0)}, \phi_2^{(0)}$ and $\phi_3^{(0)}$ of $\Phi_{8\times 3}^{(0)}$ has only up to three different values. This bound is precisely the number of nodes of $\mathcal{G}_1$. For each column of $\phi_\ell^{(0)}$, all nodes with the same parent take the same value. Similarly, the 3×1 vector $\phi_1^{(1)}$ is constant.
- This example shows that the HaarPool amalgamates the node feature by adding the same weight to the nodes that are in the same cluster of the coarser layer, and in this way, pools the feature using the graph clustering information.

Compressive Haar Transforms

We utilize adjoint Haar transforms for graph signal f to compute HaarPool. For layer j of the chain, $N_j > N_{j+1}$,

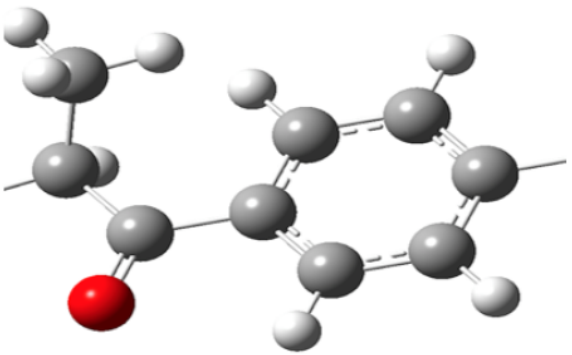
$$(\widetilde{\Phi}_j)^T f = \left(\sum_{v \in V_j} \phi_1^{(j)}(v) f(v), \ldots, \sum_{v \in V_j} \phi_{N_j}^{(j)}(v) f(v) \right) \in \mathbb{R}^{N_j}$$

$$\Downarrow$$

$$(\Phi_j)^T f = \left(\sum_{v \in V_j} \phi_1^{(j)}(v) f(v), \ldots, \sum_{v \in V_j} \phi_{N_{j+1}}^{(j)}(v) f(v) \right) \in \mathbb{R}^{N_{j+1}}$$

- We preserve the main (approximation) information of the graph signal.
- Reduce the dimension of the graph each time we use compressive Haar transforms.
- Due to the sparsity of the Haar basis matrix and then fast Haar transforms, the HaarPool is computationally feasible.

Quantum Chemistry Graph Regression



- QM7 is a collection of $7,165$ molecules, train/test $= 4/1$.
- Each molecule contains ≤ 23 atoms (including C, O, N, S), atoms are connected by bonds, molecular structure varies (e.g. double/triple bonds, cycles, carboxy, cyanide ...).
- Molecule is a graph, atoms are nodes, bonds are edges and Coulomb energy as weights, then Coulomb energy matrix is adjacency matrix.
- Task: to predict atomization energy of molecule given the molecular structure.

Methods and Architecture in Comparison

- In the experiment, we normalize the label value by subtracting the mean and scaling the standard deviation (Std Dev) to 1. We then need to convert the predicted output to the original label domain (by re-scaling and adding the mean back).
- Following *Gilmer et al. (2017).* , we use mean squared error (MSE) as the loss for training and mean absolute error (MAE) as the evaluation metric for validation and test.
- Here, the splitting percentages for training, validation, and test are 80%, 10%, and 10%, respectively. We set the hidden dimension of the GCN layer as 64, the Adam for optimization with the learning rate 5.0e-4, and the maximal epoch 50 with no early stop. We do not use dropout as it would slightly lower the performance. For better comparison, we repeat all experiments ten times with different random seeds.
- We use the same GNN architecture to test HaarPool and SAGPool *Lee et al., (2019).* : one GCN layer, one graph pooling layer, plus one 3-layer MLP.
- We compare the performance (test MAE) of the GCN-HaarPool against the GCN-SAGPool and other methods including Random Forest **RF** *Breiman, (2001).* , Multitask Networks **Multitask** *Ramsundar et al., (2015).* , Kernel Ridge Regression **KRR** *Cortes & Vapnik, (1995).* , Graph Convolutional models **GC** *Altae-Tran et al., (2017).* .