Grouping Matrix Based Graph Pooling with Adaptive Number of Clusters

Sung Moon Ko[1], Sungjun Cho[1], Dae-Woong Jeong[1], Sehui Han[1], Moontae Lee[1,2], Honglak Lee[1]

[1]LG AI Research
[2]University of Illinois Chicago
{sungmoon.ko, sungjun.cho, dw.jeong, hansse.han, moontae.lee, honglak}@lgresearch.ai

Abstract

Graph pooling is a crucial operation for encoding hierarchical structures within graphs. Most existing graph pooling approaches formulate the problem as a node clustering task which effectively captures the graph topology. Conventional methods ask users to specify an appropriate number of clusters as a hyperparameter, then assume that all input graphs share the same number of clusters. In inductive settings where the number of clusters can vary, however, the model should be able to represent this variation in its pooling layers in order to learn suitable clusters. Thus we propose GMPOOL, a novel differentiable graph pooling architecture that automatically determines the appropriate number of clusters based on the input data. The main intuition involves a *grouping matrix* defined as a quadratic form of the pooling operator, which induces use of binary classification probabilities of pairwise combinations of nodes. GMPOOL obtains the pooling operator by first computing the grouping matrix, then decomposing it. Extensive evaluations on molecular property prediction tasks demonstrate that our method outperforms conventional methods.

Introduction

Graph Neural Networks (GNNs) learn representations of individual nodes based on the connectivity structure of an input graph. For graph-level prediction tasks, the standard procedure globally pools all the node features into a single graph representation without weight difference, then feeds the representation to a final prediction layer. This process implies that information only propagates through node-to-node edges, rendering the model unable to hierarchically aggregate information efficiently beyond local convolution.

However, a hierarchical structure can encode the global topology of graphs that is useful for effective learning of long range interactions. Therefore, designing a pooling architecture which respects the graph structure is crucial for downstream tasks such as social network analyses (Kipf and Welling 2017; Hamilton, Ying, and Leskovec 2017) and molecule property predictions (Duvenaud et al. 2015; Defferrard, Bresson, and Vandergheynst 2016; Bruna et al. 2013; Coley et al. 2017; Scarselli et al. 2009; Jin et al. 2018).

As an alternative to global pooling, DiffPool first proposed an end-to-end differentiable pooling by soft-classifying each node into a smaller number of clusters (Ying et al. 2018). Later gPool (Gao and Ji 2019) and SAGPool (Lee, Lee, and Kang 2019) incorporated the attention mechanism into pooling, while MinCutPool proposed grouping the nodes into clusters by minimizing the relaxed K-way normalized minimum cut objective (Bianchi, Grattarola, and Alippi 2019).

In most inductive setups, there is no single number of clusters that is suitable across all graphs in the dataset. Particularly in molecular graphs, the number of functional groups often determines useful chemical characteristics and behaviors, while varying significantly across different molecules. Nonetheless, existing pooling methods require the number of clusters as a hyperparameter and operate under the assumption that all graphs share the same number of clusters (Ranjan, Sanyal, and Talukdar 2020). This is undesirable as it not only requires hyperparameter tuning, but also imposes a strong inductive bias that deteriorates downstream performance.

To overcome this challenge, we propose GMPOOL, a general pooling framework that does not require an universal number of clusters as a user hyperparameter. Figure 1 depicts the overall framework of GMPOOL. The core intuition is that the product of a pooling matrix with itself forms a grouping matrix, where each (i, j)-th entry indicates the pairwise *clustering similarity*: whether the nodes i and j are pooled to the same clusters. For each graph, GMPOOL parameterizes the clustering similarities in its grouping matrix via a classification layer. Finally, we perform SVD on the grouping matrix to obtain the pooling matrix such that the overall rank represents the suitable number of clusters. We also test a single-pooling variant NGMPOOL that does not perform any decomposition, but rather uses the grouping matrix as is. In real-world molecular property prediction tasks, we show that our approach outperforms previous baselines, while successfully learning suitable clusters.

The main contributions of this paper are as follows:

- We design a grouping matrix based pooling operator that does not require users to specify the number of clusters a priori.
- We propose GMPOOL and NGMPOOL. GMPOOL performs SVD on the grouping matrix to obtain the pooling matrix, whereas NGMPOOL utilizes the grouping matrix as is.
- We demonstrate the power of our methods both quantita-

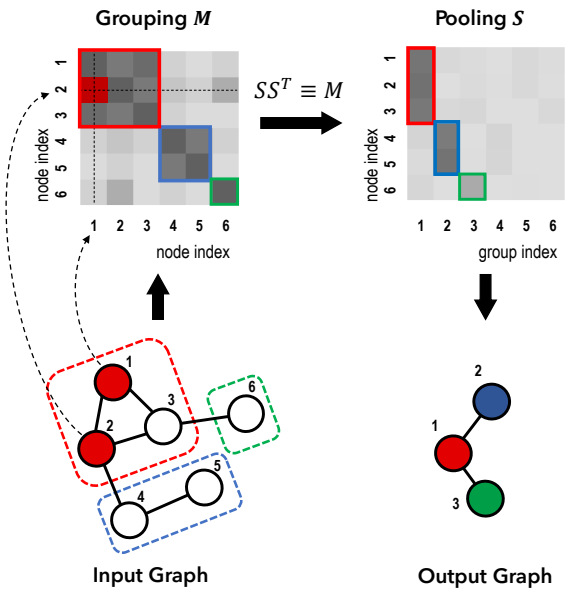


Figure 1: An illustration of our framework. Our method first forms a grouping matrix that encodes clustering similarities between each pair of nodes and then acquires the pooling matrix that coarsens the graph by decomposing the grouping matrix.

tively and qualitatively on a wide range of real molecular property prediction tasks.

Related Work

Graph Neural Networks. GNN architectures have shown great performance in various fields such as social network data, authorship/citation networks, and molecular data that can naturally be interpreted as graphs. For graph convolution, several work have utilized the graph Laplacian in the spectral domain. However, sheer convolution in the spectral domain suffers from the non-locality problem, and various approaches have been introduced to overcome this limitation. (Hamilton, Ying, and Leskovec 2017; Xu et al. 2019; Veličković et al. 2018; Gilmer et al. 2017) One stream of work has embedded the attention architecture into GNN, inferring the interaction between nodes without using a diffusion-like picture. (Veličković et al. 2018) Another line of work considered message passing networks, which ensures the signal to be localized and non-linearly weighted. (Gilmer et al. 2017) This architecture has been proven to be highly effective in molecular property prediction fields. (Yang et al. 2019)

Graph Pooling. Graph pooling aims to utilize the hierarchical nature of graphs. Early work mainly focused on fixed axiomatic pooling methods such as minimum cut, k-means, and spectral clustering without any gradient-based optimization. (Bruna et al. 2013; Coates and Ng 2011; Kushnir, Galun, and Brandt 2006; Von Luxburg 2007; Dhillon, Guan, and Kulis 2007) Although these pooling methods are effective on graphs without noise, the same heuristic often fails to work well on real datasets and tasks, especially due to a lack of differentiability that prohibits training under supervised signals. Since node representations and pooling strategies mutually affect each other during training, simultaneous optimization of whole components is crucial for avoiding local minima. Among many solutions, DIFFPOOL (Ying et al. 2018) is the first to propose an end-to-end learnable pooling mechanism that learns an assignment matrix in which each entry represents the probability of a node being assigned to a cluster.

GPOOL (Gao and Ji 2019) and SAGPOOL (Lee, Lee, and Kang 2019) are ranking-based pooling methods that coarsen the input graph by ranking and downsampling a small subset of nodes. MINCUTPOOL (Bianchi, Grattarola, and Alippi 2019) leverages a continuous relaxation of the minimum-cut objective, enabling spectral clustering under full differentiability.

However, the pooling methods above all share a common limitation: the number of clusters must be predefined for each layer as hyperparameters. This limitation is especially detrimental in inductive settings such as molecular property prediction, where each graph can have varying numbers of useful sub-structures. (Kanakaveti et al. 2017; Ertl, Altmann, and McKenna 2020a; Guvench 2016) Allowing the model to pool towards varying number of clusters based on data is expected to enhance performance, and our proposed GMPOOL allows such variation through the rank of the grouping matrix. To the best of our knowledge, GMPOOL is the first to achieve high performance without the need to manually adjust the number of clusters through additional hyperparameter tuning.

Proposed Method

In this section, we propose a novel differentiable pooling layer, GMPOOL, which obtains the pooling matrix by first building a grouping matrix that contains clustering similarities of pairwise nodes and then decomposing the matrix to its square-root form. We start the section with preliminary information, then outline the details of GMPOOL in later sections.

Preliminaries

Problem setting. We assume an inductive graph-level prediction setting where our aim is to learn a function $f_\theta : \mathcal{G} \rightarrow \mathcal{Y}$ that maps a graph $G \in \mathcal{G}$ to a property label $y \in \mathcal{Y}$. Each graph G with n nodes is represented as a triplet $G = (A, X, E)$ with graph adjacency $A \in \{0,1\}^{n \times n}$, node features $X \in \mathbb{R}^{n \times d_n}$, and edge features $E \in \mathbb{R}^{n \times n \times d_e}$. We use X_i and E_{ij} to denote the features of node i and edge (i, j), respectively.

Directed MPNN and pooling. As our backbone GNN, we adopt the Directed Message Passing Neural Network (DMPNN) (Yang et al. 2019) which aggregates messages through directed edges. Note that while we chose DMPNN due to its superior performance over GNN architectures, our pooling layer is module-agnostic and can be combined with any GNN as long as node representations are returned as output. Given a graph, DMPNN first initializes the hidden state of each edge (i, j) based on its feature E_{ij} and the source-node's feature X_i. At each timestep t, each directional

edge gathers hidden states from incident edges into a message m_{ij}^{t+1} and updates its own hidden state to h_{ij}^{t+1} as follows

$$m_{ij}^{t+1} = \sum_{k \in \mathcal{N}(i) \setminus j} h_{ki}^{t} \tag{1}$$

$$h_{ij}^{t+1} = \texttt{ReLU}(h_{ij}^{0} + W_e m_{ij}^{t+1}) \tag{2}$$

Here, $\mathcal{N}(i)$ denotes the set of neighboring nodes of node i and W_e a learnable weight. The hidden states of nodes are updated by aggregating the hidden states of incident edges into message m_i^{t+1}, and passing its concatenation with the node feature X_i into a linear layer followed by ReLU non-linearity

$$m_i^{t+1} = \sum_{j \in \mathcal{N}(i)} h_{ij}^{t} \tag{3}$$

$$h_i^{t+1} = \texttt{ReLU}(W_n \texttt{concat}(X_i, m_i^{t+1})) \tag{4}$$

Similarly, W_n denotes a learnable weight. Assuming DMPNN runs for T timesteps, we use $(X_{out}, E_{out}) = \texttt{GNN}(A, X, E)$ to denote the output representation matrices containing hidden states of all nodes and edges, respectively (i.e., $X_{out,i} = h_i^T$ and $E_{out,ij} = h_{ij}^T$).

For graph-level prediction, the node representations after the final GNN layer are typically sum-pooled to obtain a single graph representation $h_G = \sum_i h_i$, which is then passed to a FFN prediction layer. Note that this approach only allows features to propagate locally and is hence unable to learn long-range dependencies and hierarchical structures within graphs.

Our goal is to learn a pooling operator to coarsen the input graph after the GNN in each hierarchical layer. In each hierarchical layer, the GNN constructs node representations and then the pooling layer forms a coarsened graph, which is used as input to the next hierarchical layer. More formally, given the representations from the l-th layer as $(X_{out}^{(l)}, E_{out}^{(l)}) = \texttt{GNN}(A^{(l)}, X^{(l)}, E^{(l)})$, the pooling layer yields an assignment matrix $S^{(l)} \in \mathbb{R}^{n_l \times n_{l+1}}$ pooling n_l nodes into n_{l+1} clusters. Then, the graph $G^{(l)} = (A^{(l)}, X^{(l)}, E^{(l)})$ is coarsened into $G^{(l+1)} = (A^{(l+1)}, X^{(l+1)}, E^{(l+1)}) = (S^{(l)^T} A^{(l)} S^{(l)}, S^{(l)^T} X_{out}^{(l)}, S^{(l)^T} E_{out}^{(l)} S^{(l)})$. This hierarchical process can be utilized iteratively depending on the task at hand.

Differentiable Pooling via Grouping Decomposition

When looking into the relation between pairs of nodes, the grouping task becomes rather simple. While most previous models focus on classifying each node to a predefined number of clusters, our idea simplifies the task into classifying whether each pair of nodes is in the same group. Thus, setting the number of clusters a priori becomes unnecessary. This classification will go through every pair of combinations of nodes to ensure permutation invariance.

$$M_{ij}^{(l)} = \text{Softmax}(\text{Clf}(f(X_i, X_j))) \qquad \forall\ i, j \in \#\text{ of Nodes} \tag{5}$$

where Clf is a classifier, $M^{(l)} \in \mathbb{R}^{n \times n}$ and f is a commutative function

$$f : X \oplus X \to Y \qquad \text{where } X, Y \in \mathbb{R}^n \tag{6}$$

that maps two input vectors into one output vector. While there exist many available choices for f, we use Euclidean distance between input vectors to simplify the classification task. Each matrix index corresponds to the node number and each element contains probability values for each pair of nodes whether they are in the same group.

As an illustrative example, consider a set of disjoint clusters with no overlapping nodes. In such case, the grouping matrix not only contains $0, 1$ as its elements, but also can be reformed into a block diagonal form. The number of blocks corresponds to the number of groups after pooling and nodes assigned to the same blocks corresponds to a same group. For instance, if there are three different groups and each group size are k_1, k_2, k_3,

$$M^{(l)} = \begin{bmatrix} \boxed{1_{k_1 \times k_1}} & 0 & 0 \\ 0 & \boxed{1_{k_2 \times k_2}} & 0 \\ 0 & 0 & \boxed{1_{k_3 \times k_3}} \end{bmatrix} \tag{7}$$

One can easily see that the corresponding pooling operator is as follows

$$S^{(l)} = \begin{bmatrix} \boxed{1_{k_1 \times 1}} & 0 & 0 & \cdots & 0 \\ 0 & \boxed{1_{k_2 \times 1}} & 0 & \cdots & 0 \\ 0 & 0 & \boxed{1_{k_3 \times 1}} & \cdots & 0 \end{bmatrix} \tag{8}$$

In general, each element of the grouping matrix (in eq. 7) is a continuous number within $[0, 1]$, which allows soft-clustering with overlapping nodes. For detailed computation, see appendix.

However, the grouping matrix itself has a limited role in pooling operations. Therefore, extracting pooling operators from the grouping matrix is crucial. Our strategy to form a pooling operator is rather simple. It can be acquired by decomposing a grouping matrix into square-root form. There are numerous known methods which can be utilized, yet we will introduce two representative methods in the following subsection.

Decomposition Schemes

While the grouping matrix cannot be used for pooling as is, it encodes how similarly each pair of nodes are pooled as it equals the product of the pooling operator with its transpose. The (i, j)-th entry of the grouping matrix equals $\langle S_i^{(l)}, S_j^{(l)} \rangle = 1$ if the nodes are exactly pooled to the same clusters, $\langle S_i^{(l)}, S_j^{(l)} \rangle = 0$ if they are pooled orthogonally to different clusters. Therefore, if we can decompose the grouping matrix into square-root form, it can be interpreted as a pooling operator for the model.

$$S^{(l)} S^{(l)T} = M^{(l)} \tag{9}$$

The pooling operator $S \in \mathbb{R}^{n_l \times n_{l+1}}$ is a matrix where $n_{l+1} \leq n_l$. Note that by multiplying pooling operator S in reverse order, the degree matrix $D \in \mathbb{R}^{n_{l+1} \times n_{l+1}}$ of pooling space can be obtained.

$$S^{(l)T} S^{(l)} = D^{(l)} \tag{10}$$

From eq. 9, it is obvious that the pooling operator completely reconstructs grouping matrix by interacting pooling indices. Moreover, S can be interpreted as a weighted matrix for each node to form appropriate sub-structures.

Eigendecomposition Based Method Eigendecomposition is one of the basic decomposition schemes one can consider. It is widely used to decompose a given matrix into orthonormal basis $O \in \mathbb{R}^{n_l \times n_l}$ and eigenvalue $\Lambda \in \mathbb{R}^{n_l \times n_l}$.

$$M^{(l)} = O\Lambda O^T \tag{11}$$

This particular decomposition scheme always works unless the determinant of a given matrix is equal to 0. From eq. 11, one can rearrange RHS of the equation to become a square form of pooling operator if we set $n_{l+1} = n_l$.

$$M^{(l)} = O\sqrt{\Lambda}\sqrt{\Lambda}O^T \equiv S^{(l)}S^{(l)T} \tag{12}$$

The pooling operator S is a square matrix with size of $n_l \times n_l$, yet the eigenvalue Λ suppresses useless ranks in the matrix by multiplying 0 to each column of orthonormal basis. Also, eigendecomposition works for any matrix with non-zero determinants, and so it performs perfectly fine in real world situations. Furthermore, any symmetric and real matrix are guaranteed to have real eigenvalues as well as vectors. Therefore, the square-root of the grouping matrix is ensured to be interpreted as a transformation operator forming sub-groups from nodes. These continuous real valued elements have the advantage that nodes can be soft-clustered to sub-groups. In conventional clustering, it is hard to cluster these structures properly. However, since soft clustering is naturally embedded in the algorithm, linker structures can be dealt with ease.

After acquiring the pooling operator, the pooling process becomes obvious. Nodes are in fundamental representation while edge features and adjacency matrix are in adjoint representation. Which leads to the following transformation rules.

$$X^{(l+1)} = S^{(l)}X^{(l)} \tag{13}$$

$$E^{(l+1)} = S^{(l)}E^{(l)}S^{(l)T} \tag{14}$$

$$A^{(l+1)} = S^{(l)}A^{(l)}S^{(l)T} \tag{15}$$

where $S^{(l)} \in \mathbb{R}^{n_l \times n_l}$, $X^{(l)} \in \mathbb{R}^{n_l}$, $E^{(l)} \in \mathbb{R}^{n_l \times n_l}$ and $A^{(l)} \in \mathbb{R}^{n_l \times n_l}$.

If grouping is properly done, 0 (or close to 0) components will appear in the decomposed eigenvalue matrix. These zero eigenvalues arise naturally and play a role in disregarding group information; those are ineffective towards prediction. However, zero elements in the eigenvalues causes a major problem in the decomposition process since the matrix might carry a singular determinant. Eigendecomposition is based on an iterative approximation algorithm which includes unbounded terms if there exists a small eigengap. This can cause numerical instability during backpropagation, as we can see in the equation below showing the gradient of the loss with respect to the adjacency matrix A (Ionescu, Vantzos, and Sminchisescu 2015):

$$\frac{\partial l}{\partial A} = U\left(K^T \odot \left(U^T\frac{\partial l}{\partial U}\right) + \left(\frac{\partial l}{\partial \Lambda}\right)_{\text{diag}}\right)(U^T) \tag{16}$$

Here, $\odot$ denotes element-wise product. Off-diagonal components of $K = 1/(\lambda_i - \lambda_j)$ causes the problem, since the value blows up to the infinity if any two eigenvalues are close or very small. However, there are some solutions for this matter by approximating gradient in different ways (Wang et al. 2019, 2021; Song, Sebe, and Wang 2021). Those methods are developed further to achieve higher speed in the calculation (Song, Sebe, and Wang 2022). They claim that the method is noticeably faster, over 8 times, than the standard SVD which has the time complexity $\mathcal{O}(n^3)$. Thus, we utilized this method in our work to stabilize and accelerate the learning process. However, since the algorithm achieves the higher speed by approximating gradients, the error compared to standard SVD grows bigger as the size of the matrix grows. Therefore, this method might not be valid with large sized graph data.

Pooling without Decomposing The Grouping Matrix Another decomposition scheme we are introducing has a rather different approach. Since computing the square root of a given matrix is not an easy task, here we focus on the square of the pooling operator, which is nothing but the grouping matrix itself, and formulate a pooling-like effect by multiplying the grouping matrix. The key idea is to retain pooling depth to one and use a weighted aggregation vector in pooling space as an aggregation basis. The weighted aggregation vector is transformed Euclidean one vector by acting a pooling matrix obtained by decomposing the grouping matrix.

$$1_i^{(l+1)} = S^{(l)}1_i^{(l)} \tag{17}$$

where 1_i is an one vector.

The final form of the transformation can be expressed as follows.

$$X^{(l+1)} \sim M^{(l)}X^{(l)} \tag{18}$$

$$E^{(l+1)} \sim M^{(l)}E^{(l)}M^{(l)} \tag{19}$$

This pooling scheme is simpler to use and more scalable (with $\mathcal{O}(n^2)$ cost) than GMPOOL since the method circumvents SVD computation. Yet there are two mathematical ambiguities. One is that it is only valid for single depth pooling cases. If one tries to perform multiple sequential pooling operations, the pooling operators are no more available to be reduced into the grouping matrix, since two different pooling operators do not commute. The other ambiguity is that most activation functions commonly used are not equivariant with pooling operators. However, since many of them are based on element-wise operations with monotonic functions, we can presume that the anomaly are not dominant in most cases. We find that this approach performs comparably to GMPOOL in our experiment setup. Furthermore, unlike GMPOOL, NGMPOOL bypasses decomposition process, hence, the computational burden is much less compared to the GMPOOL. Therefore, for small sized molecules where a single pooling depth suffice, NGMPOOL takes advantages over GMPOOL.

Dataset	Description	Input Type	Mean Size	Task Type	Data #
PLQY	Fluorescent	SMILES	57.3	Regression	515
λ_{max} Solvents	Fluorescent	SMILES	55.2	Regression	1,070
λ_{max} Films	Fluorescent	SMILES	57.2	Regression	1,270
pIC50	Binding	SMILES	30.5	Regression	10,978
Tox21	Toxicity	SMILES	26.0	Classification	2,514

Table 1: Dataset Summary

Models	PLQY (RMSE ↓)	λ_{max} Solvents (RMSE ↓)	λ_{max} Films (RMSE ↓)	pIC50 (RMSE ↓)	Tox21 (ROC-AUC ↑)
GCN (Kipf and Welling 2017)	0.2619±0.0519	64.09±2.281	58.33±2.419	0.7390±0.0265	0.6902±0.0305
DMPNN (Yang et al. 2019)	0.2414±0.0093	46.22±0.523	35.10±1.205	0.7839±0.0076	0.7191±0.0138
TOP-K (Gao and Ji 2019)	0.2204±0.0143	47.06±2.086	50.60±0.765	0.7860±0.0107	0.7593±0.0160
SAGPOOL (Lee, Lee, and Kang 2019)	0.2123±0.0166	45.44±3.812	37.98±1.968	0.7956±0.0072	0.7155±0.0342
DIFFPOOL (Ying et al. 2018)	0.2386±0.0181	56.54±0.189	52.78±0.335	0.8091±0.0126	0.6619±0.0155
ASAPOOL (Ranjan, Sanyal, and Talukdar 2020)	0.2204±0.0211	53.19±3.548	47.39±2.434	0.8310±0.0173	0.6864±0.0466
MEMPOOL (Khasahmadi et al. 2020)	0.2188±0.0345	41.66±3.859	38.54±3.249	0.7910±0.0073	0.6926±0.0257
GMPOOL (ours)	**0.2091±0.0081**	44.48±2.010	**34.34±3.027**	**0.6951±0.0270**	0.7451±0.0197
NGMPOOL (ours)	0.2169±0.0340	**38.61±2.573**	39.82±0.702	0.7253±0.0198	**0.7617±0.0201**

Table 2: Test Results from Various Datasets

Experiments

Experimental Setup

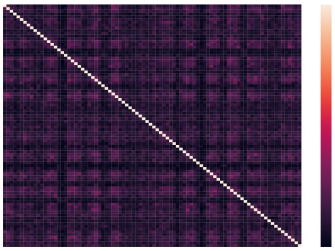
We arrange a total of five datasets to test our algorithms: two are open datasets collected from MoleculeNet (Ramsundar et al. 2019) and Binding DB (Chen X 2001, 2002; Chen and Gilson 2002; Liu T 2007; Gilson et al. 2015), three are manually collected and arranged from different literatures including scientific articles and patents.

- **PLQY** includes experimentally measured values of photoluminescence quantum yield (PLQY) for fluorescence molecules.
- λ_{max} **Solvents** contains measured λ_{max}, wavelength that shows maximum intensity for emission of a fluorescence molecule, under the solvent condition.
- λ_{max} **Films** consists of λ_{max} values measured after spin coating of fluorescence molecules on films doped with host materials.
- **pIC50** contains the negative log of the IC50 values for ATP receptor. IC50 implies minimum concentration of certain molecule needed for inhibiting half of activity of the target proteins. The IC50 values are optained from the BindingDB (https://www.bindingdb.org/bind/index.jsp).
- **Tox21** consists of results of 12 types of toxicity screening tests. We labeled a molecule 'toxic' if the molecule failed in any of screening type. Data were originated from Tox21 challenge (2014). Since there are molecules without graph structure information in the dataset, we selected $7,831$ molecules that have the graph structure information.

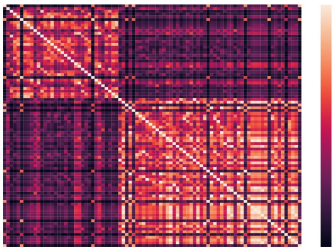
For proper evaluation of pooling approaches, each graph in the data must have at least two or more effective groups. However, Tox21 and pIC50 data contains molecules too small to contain multiple groups and thus we drop molecules with less than 20 nodes from the datasets. In addition, we drop molecules with more than 40 nodes from Tox21 and pIC50 datasets to accelerate the whole training process under dense matrix computations: the largest molecule in each respective dataset has 86 and 132 nodes, but the ratio of molecules with size over 40 in the dataset is only 3.4% and 3.6%. Especially for pIC50 dataset, the proportion of molecules with less than 20 nodes are 0.4%. Lastly, the Tox21 task has been simplified to a single classification task by setting a positive label if any of the 12 tasks are positive in the original dataset. Details can be found in Table 1 and appendix section. Every experiments are tested under five-fold settings with uniform sampling and 10% of dedicated test set to secure the results, and single RTX 3090 is used for the experiments.

Baselines

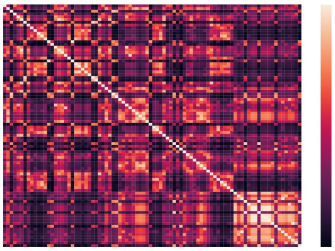
For empirical evaluation, we compare the performance of GMPOOL and NGMPOOL against that of five other pooling approaches. We run all experiments via a pipeline with a fixed DMPNN backbone, while exchanging the pooling layers only. Here we provide brief descriptions of each baselines used: TOP-K (Gao and Ji 2019) and SAGPOOL (Lee, Lee, and Kang 2019) retain nodes with the highest scoring based on the projections of node features and self-attention scores, respectively. DIFFPOOL (Ying et al. 2018) uses an additional GNN to learn soft-assignment matrices that mix nodes into clusters. ASAPOOL (Ranjan, Sanyal, and Talukdar 2020) clusters local subgraphs together through scoring and selection of clusters. MEMPOOL (Khasahmadi et al. 2020) incorporates memory layers that jointly coarsen and transform input node representations. Note that we reimplemented the DMPNN backbone, TOP-K pooling, and DIFFPOOL. Implementations of other pooling baselines are borrowed from the `pytorch-geometric` library.



(a) Initial State



(b) 2-Groups



(c) Multi-Groups

Figure 2: Examples of Grouping Matrices. Both X and Y axes are node indices. (a) is the initial state of grouping matrix, (b) and (c) show a grouping matrix for example molecules from PLQY and λ_{max} datasets, respectively.

Hyperparameters

For the DMPNN backbone of the model, we use the same hidden size of 200 across all three independent layers: the initial edge features with dimension d_e and node features with dimension d_n are passed through layers of dimension $d_e \times 200$ and $d_n \times 200$, respectively with ReLU activation. The initial node and edge embeddings are determined by features generated in RDKit. The message passing module passes node embeddings through a linear layer with dimension 200×200, followed by ReLU activation and 0.15 dropout layer. For graph representation we use a global average pooling scheme. GMPool and NGMPool construct the grouping matrix via a 200×1 linear layer and sigmoid activation without any parameters related to cluster numbers or thresholds. We use a batch size of 80 and Adam optimizer for all model training.

For baseline pooling methods that require the cluster size as a hyperparameter, we perform grid search across candidates following previous work, and present best results. However, we fix the final pooling size to 10 as the average size of most common 40 functional groups in bioactive molecules is 4.25 (Ertl, Altmann, and McKenna 2020b), indicating that molecules under concern (statistics shown in Table 1) can have up to 10 clusters. The specific hyperparameter setups used for pooling baselines can be found in appendix.

Grouping Result

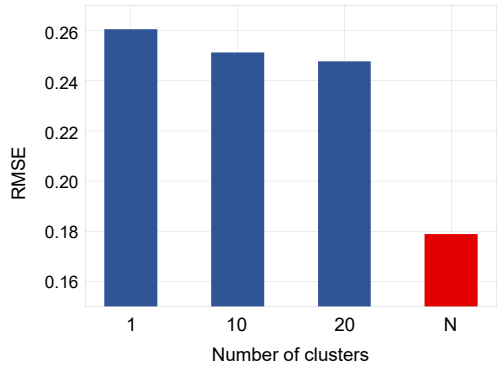
The grouping matrix starts from randomized initial state and is optimized to gather effective functional groups in the molecules (Figures 2b and 2c). Furthermore, since our algorithm fully enjoys the soft clustering concept, the result shows continuous weights for each group. This characteristic ensures the model can gather information from distant nodes if necessary. However, sometimes the grouping matrix shows unfamiliar forms, since the effective functional groups should vary due to the downstream task itself. For instance, for some simple tasks such as PLQY prediction, the grouping is rather intuitive as shown in Figure 2b, yet for complicated tasks like λ_{max} prediction, the effective functional groups are also complicated as in Figure 2c.

Main Result

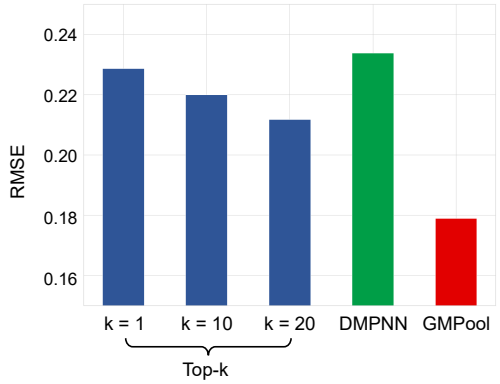
We tested various combinations of models and dataset to check the validity of our algorithm. We selected GCN, DMPNN, Top-k, SAGPool, DiffPool, ASAPool and MemPool algorithm to set a benchmark score to compare with. As it is shown in the table 2, majority of the cases, our models outperform conventional methods. However, for some tasks (i.e. λ_{max} datasets), our model is gaining only a small margin of the performance. This is caused by the underlying mechanism of the chemical effect. Since some tasks are strongly related to the effective groups of the molecule, yet others are not. In those cases, sub-structures are not intuitive and might appear in very complicated forms, as shown in Figure 2c. If the grouping becomes complicated, the rank of the pooling matrix should be larger to cover all degrees of freedom for the data. However, conventional models, which shared predefined numbers as universal grouping numbers, force to collect groups and reduce it to the low-rank form, which might not have enough degree of freedom. This will cause information loss or blend which compromises the prediction result. Therefore, one can check that in λ_{max} prediction test, conventional pooling algorithms show inferior result than simple massage passing scheme. Yet our model is not designed to reduce the physical rank of the matrix during the pooling process, and there is always enough degree of freedom to carry the information throughout learning. Hence, even for the λ_{max} case, our model outperforms the others. Furthermore, for other tasks, it is clear that our model improves performance by $5 \sim 10\%p$.

Ablation Study

One crucial hyperparameter to be explored in pooling models is the number of clusters. Even though our model does not require to fix the number of clusters in the first place, one can set the parameter by force. One can easily see in the Figure 3a



(a)



(b)

Figure 3: RMSE for PLQY prediction (a) over varying number of clusters in GMPOOL and (b) across different methods. For figure (a), GMPOOL does not require to set number of clusters, however to explore the effect, the number of clusters is set by force in the first place.

that the number of clusters can be set to the number of nodes without compromising performance of the model. Further, Figure 3b shows that our model outperforms TOP-K algorithms with various cluster numbers and original DMPNN as well. This is one of the powerful features of our model, since the model automatically splits groups and determines the appropriate number of sub-structures for each individual graph. One can also force the number of clusters and share through all graphs in an equal manner; however, it is not effective for the following reasons. In real world data, one can not esteem the exact number of clusters for individual graphs. This might be problematic if one sets the number of clusters less than it requires, the models' performance will be compromised due to the information loss. Another problem is caused by the mathematical structure of the decomposition scheme. Using SVD method will cause ambiguity since collecting only top k eigenvalues from the decomposed matrix might not reconstruct the original grouping matrix due to lack of information. It is even worse in the initial stage of the learning as the weight is almost in the random state and the top k eigenvalues are not precisely representing the appropriate clusters. Thus, as it is depicted in the above figure, it is best to leave the cluster number to be determined automatically by the model itself.

Conclusion

We have introduced a novel pooling architecture with adaptive number of clusters based on a second order pooling operator, namely the grouping matrix. The grouping matrix is based on clustering similarities between every possible pairs of nodes, ensuring permutation invariance. We have shown that our model is valid for chemical property prediction and outperforms conventional methods in real-world datasets.

While our model is useful and effective, there is still room for improvement. First of all, despite leveraging a method to decompose the grouping matrix with stable gradient computations, there exist corner cases with a small eigengap at which the model fails to converge. This event seldom happens (about 0.00018% in our experiments), but can be non-negligible when one needs to learn with a large number of data points. Hence, one future direction would be to impose proper constraints on the loss to avoid such gradient blowup in the grouping matrix.

Another future direction would be to enhance scalability of our methods to improve applicability to large-scale graphs. Since the grouping matrix decomposition step via SVD is the main computational bottleneck of GMPOOL, incorporating faster decomposition modules such as randomized approximation (Halko, Martinsson, and Tropp 2011; Vasudevan and Ramakrishna 2017) methods can lead to faster inference. However, this is likely to incur loss in predictive performance, and as the focus of this work lies in allowing variation in the number of clusters in small molecular graphs where scalability is not an issue, we defer improving the scalability to future work.

Lastly, generalizing the second order grouping matrix towards higher-order grouping tensors can allow further expressive power. We have introduced a pairwise structure; yet it is not obliged to be fixed into the pairwise form. If we consider higher-order form of node combinations, i.e. k-form where $k < N$ and N is total node number, the grouping matrix can be generalized into the higher rank tensor. Based on the tensor-form, the transformation rule can be written as

$$\tilde{M}_{\mu_1 \cdots \mu_k} = S_{\mu_1}{}^{\nu_1} \cdots S_{\mu_k}{}^{\nu_k} M_{\nu_1 \cdots \nu_k} \tag{20}$$

Note that to satisfy the above transformation rule, the following conditions are required. One is that selecting nodes combination should be as same as selecting nodes *set* and size of the *set* should fixed into the number of nodes in a group. The other is that the classification result of the node set should be retained the same for any subset in the node set. This concept may have a connection to hypergraph configurations. However if we raise the nodes numbers above 2, required computation power increases by a huge amount, since the combination number grows exponentially until the number of nodes hits $N/2$. Therefore, practically it is a difficult task to test the higher rank version of our algorithm, yet it could be useful for learning datasets with higher order connections.

Acknowledgments

We would like to thank Taeyang Lee and Junho Lee in LG Display for helpful discussions and dataset preparation for main experiments.

References

Bianchi, F. M.; Grattarola, D.; and Alippi, C. 2019. Mincut pooling in Graph Neural Networks. *CoRR*, abs/1907.00481.

Bruna, J.; Zaremba, W.; Szlam, A.; and LeCun, Y. 2013. Spectral Networks and Locally Connected Networks on Graphs.

Chen, L. Y., X.; and Gilson, M. 2002. The Binding Database: Overview and User's Guide. *Biopolymers*, 61: 127–141.

Chen X, G. M., Liu M. 2001. indingDB: a web-accessible molecular recognition database. *Combinatorial Chemistry & High Throughput Screening*, 4(8): 719–725.

Chen X, L. M. G. M., Lin Y. 2002. The Binding Database: data management and interface design. *Bioinformatics*, 18(1): 130–139.

Coates, A.; and Ng, A. 2011. Selecting Receptive Fields in Deep Networks. In Shawe-Taylor, J.; Zemel, R.; Bartlett, P.; Pereira, F.; and Weinberger, K., eds., *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc.

Coley, C. W.; Barzilay, R.; Green, W. H.; Jaakkola, T. S.; and Jensen, K. F. 2017. Convolutional Embedding of Attributed Molecular Graphs for Physical Property Prediction. *Journal of Chemical Information and Modeling*, 57(8): 1757–1772. PMID: 28696688.

Defferrard, M.; Bresson, X.; and Vandergheynst, P. 2016. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. In Lee, D.; Sugiyama, M.; Luxburg, U.; Guyon, I.; and Garnett, R., eds., *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc.

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*, 29(11): 1944–1957.

Duvenaud, D. K.; Maclaurin, D.; Iparraguirre, J.; Bombarell, R.; Hirzel, T.; Aspuru-Guzik, A.; and Adams, R. P. 2015. Convolutional Networks on Graphs for Learning Molecular Fingerprints. In Cortes, C.; Lawrence, N.; Lee, D.; Sugiyama, M.; and Garnett, R., eds., *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc.

Ertl, P.; Altmann, E.; and McKenna, J. M. 2020a. The Most Common Functional Groups in Bioactive Molecules and How Their Popularity Has Evolved over Time. *Journal of Medicinal Chemistry*, 63(15): 8408–8418. PMID: 32663408.

Ertl, P.; Altmann, E.; and McKenna, J. M. 2020b. The most common functional groups in bioactive molecules and how their popularity has evolved over time. *Journal of medicinal chemistry*, 63(15): 8408–8418.

Gao, H.; and Ji, S. 2019. Graph U-Nets. *CoRR*, abs/1905.05178.

Gilmer, J.; Schoenholz, S. S.; Riley, P. F.; Vinyals, O.; and Dahl, G. E. 2017. Neural Message Passing for Quantum Chemistry. In Precup, D.; and Teh, Y. W., eds., *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, 1263–1272. PMLR.

Gilson, M. K.; Liu, T.; Baitaluk, M.; Nicola, G.; Hwang, L.; and Chong, J. 2015. BindingDB in 2015: A public database for medicinal chemistry, computational chemistry and systems pharmacology. *Nucleic Acids Research*, 44(D1): D1045–D1053.

Guvench, O. 2016. Computational functional group mapping for drug discovery. *Drug Discovery Today*, 21(12): 1928–1931.

Halko, N.; Martinsson, P.-G.; and Tropp, J. A. 2011. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review*, 53(2): 217–288.

Hamilton, W.; Ying, Z.; and Leskovec, J. 2017. Inductive Representation Learning on Large Graphs. In Guyon, I.; Luxburg, U. V.; Bengio, S.; Wallach, H.; Fergus, R.; Vishwanathan, S.; and Garnett, R., eds., *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Ionescu, C.; Vantzos, O.; and Sminchisescu, C. 2015. Training Deep Networks with Structured Layers by Matrix Backpropagation. *CoRR*, abs/1509.07838.

Jin, W.; Yang, K.; Barzilay, R.; and Jaakkola, T. 2018. Learning Multimodal Graph-to-Graph Translation for Molecular Optimization.

Kanakaveti, V.; Sakthivel, R.; Rayala, S. K.; and Gromiha, M. M. 2017. Importance of functional groups in predicting the activity of small molecule inhibitors for Bcl-2 and Bcl-xL. *Chemical Biology & Drug Design*, 90(2): 308–316.

Khasahmadi, A. H.; Hassani, K.; Moradi, P.; Lee, L.; and Morris, Q. 2020. Memory-Based Graph Networks. In *International Conference on Learning Representations*.

Kipf, T. N.; and Welling, M. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *International Conference on Learning Representations*.

Kushnir, D.; Galun, M.; and Brandt, A. 2006. Fast Multiscale Clustering and Manifold Identification. *Pattern Recogn.*, 39(10): 1876–1891.

Lee, J.; Lee, I.; and Kang, J. 2019. Self-Attention Graph Pooling. *CoRR*, abs/1904.08082.

Liu T, W. X. J. R. G. M., Lin Y. 2007. BindingDB: a web-accessible database of experimentally determined protein-ligand binding affinities. *Nucleic Acids Research*, 35: D198–D202.

Ramsundar, B.; Eastman, P.; Walters, P.; Pande, V.; Leswing, K.; and Wu, Z. 2019. *Deep Learning for the Life Sciences*. O'Reilly Media. https://www.amazon.com/Deep-Learning-Life-Sciences-Microscopy/dp/1492039837.

Ranjan, E.; Sanyal, S.; and Talukdar, P. 2020. Asap: Adaptive structure aware pooling for learning hierarchical graph representations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, 5470–5477.

Scarselli, F.; Gori, M.; Tsoi, A. C.; Hagenbuchner, M.; and Monfardini, G. 2009. The Graph Neural Network Model. *IEEE Transactions on Neural Networks*, 20(1): 61–80.

Song, Y.; Sebe, N.; and Wang, W. 2021. Why Approximate Matrix Square Root Outperforms Accurate SVD in Global Covariance Pooling? *CoRR*, abs/2105.02498.

Song, Y.; Sebe, N.; and Wang, W. 2022. Fast Differentiable Matrix Square Root. *CoRR*, abs/2201.08663.

Vasudevan, V.; and Ramakrishna, M. 2017. A Hierarchical Singular Value Decomposition Algorithm for Low Rank Matrices. *CoRR*, abs/1710.02812.

Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Liò, P.; and Bengio, Y. 2018. Graph Attention Networks. In *International Conference on Learning Representations*.

Von Luxburg, U. 2007. A tutorial on spectral clustering. *Statistics and computing*, 17: 395–416.

Wang, W.; Dang, Z.; Hu, Y.; Fua, P.; and Salzmann, M. 2019. Backpropagation-Friendly Eigendecomposition. *CoRR*, abs/1906.09023.

Wang, W.; Dang, Z.; Hu, Y.; Fua, P.; and Salzmann, M. 2021. Robust Differentiable SVD. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1–1.

Xu, K.; Hu, W.; Leskovec, J.; and Jegelka, S. 2019. How Powerful are Graph Neural Networks? In *International Conference on Learning Representations*.

Yang, K.; Swanson, K.; Jin, W.; Coley, C.; Eiden, P.; Gao, H.; Guzman-Perez, A.; Hopper, T.; Kelley, B.; Mathea, M.; Palmer, A.; Settels, V.; Jaakkola, T.; Jensen, K.; and Barzilay, R. 2019. Analyzing Learned Molecular Representations for Property Prediction. *Journal of Chemical Information and Modeling*, 59(8): 3370–3388. PMID: 31361484.

Ying, R.; You, J.; Morris, C.; Ren, X.; Hamilton, W. L.; and Leskovec, J. 2018. Hierarchical Graph Representation Learning with Differentiable Pooling. *CoRR*, abs/1806.08804.