GRASSMANN GRAPH EMBEDDING

Anonymous authors
Paper under double-blind review

ABSTRACT

Geometric deep learning that employs the geometric and topological features of data has attracted increasing attention in deep neural networks. Learning the intrinsic structure property of data is a crucial step for dimensionality reduction and effective feature extraction. This paper develops *Grassmann graph embedding*, which combines graph convolutions to capture the main components within graphs' hidden representations. Each set of featured graph nodes is mapped to a point on a Grassmann matrix manifold through Singular Value Decomposition, which is then embedded into a symmetric matrix space that approximates denoised second-order feature information. The view of treating nodes as a set could inspire many potential applications. In particular, we propose Grassmann (global graph) pooling that can connect with any graph convolution for graph neural networks. The Grassmann pooling achieves state-of-the-art performance on a variety of graph prediction benchmarks.

1 INTRODUCTION

Graphs are prevalent objects in machine learning and data science. Graph representation learning has recently attracted increasing attention with graph neural networks (GNNs) emerging as one of the most prominent avenues in deep learning (Bronstein et al., 2017; Hamilton, 2020). One critical ingredient of graph representation like GNNs is to effectively distill key features of graph data. Graph convolution (Bruna et al., 2014) to GNN, or more generally, graph neural message passing (Gilmer et al., 2017), provides an efficient way to extract core information of a graph. In node prediction tasks, graph convolution is an excellent learner that aggregates graph features and uses node relation to predict node labels. On the contrary, graph-level learning tasks are usually defined on graph data with varying size and structure, in which case dimensionality reduction (for graph size) is often an indispensable necessity. To this end, a graph pooling scheme is critical to obtaining scaled-down graphs or graph-level embeddings. For example, TOPKPOOL (Cătălina et al., 2018) exploits a node selection criterion by a learned score and drops all but the top-ranked nodes; SAGPOOL (Lee et al., 2019) calculates a score with an extra MLP in the TOPK framework to enhance the selection efficacy; and ATTENTIONPOOL (Li et al., 2016) defines a soft scoring mechanism that concatenates node state and annotation.

One basic property of a pooling or embedding operation is node permutation invariance, as the node order of an undirected graph must not incline the network to focus on specific information. The permutation invariance is intuitive in heuristic pooling operations like summation, averaging, and maximization, where the aggregation rules do not rely on the order information of nodes. Indeed we can define such operations on subsets of a graph. The view of treating nodes as a set in pooling can be seen in the recent work of Kolouri et al. (2021), where a node set finds its Wasserstein Embedding within a linear optimal transport framework (Mémoli, 2011). The underlying idea is to match two sets of nodes by a proper Wasserstein metric.

When graph nodes come with features, we may view the set of graph nodes from the perspective of manifold learning. The set of feature vectors naturally generates a subspace as a new representation. In other words, the node information of a graph can be abstracted into a lower-dimensional subspace, and a learning task over graphs of different size and structure can thus be converted into a learning task on a Grassmann manifold (Absil et al., 2008). We call this process of mapping a graph onto the Grassmann manifold *Grassmann (graph) embedding*. The acquired Grassmann embeddings from graphs can be pipelined into a Grassmann learning algorithm, such as Deep Grassmann Networks (Huang et al., 2018) and Grassmann clustering (Wang et al., 2014; 2017).

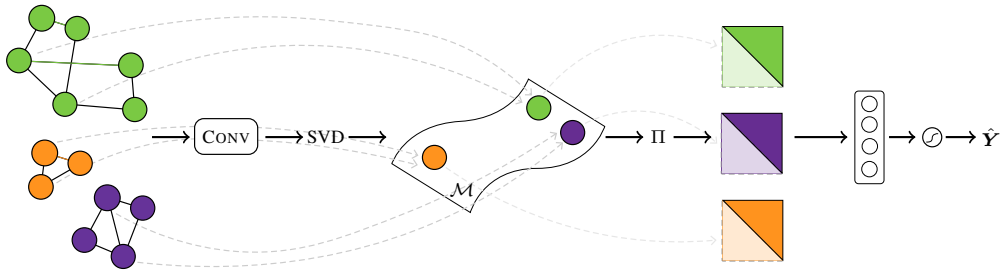


Figure 1: Computational principle of the proposed *Grassmann Graph Embedding*. A complete description is provided in Section 1. This work implements the strategy and constructs Grassmann pooling (GRPOOL).

As a proof of concept, this work develops a simple manifold embedding strategy (Figure 1). An input graph is first passed through Graph Convolutional layers to extract the hidden representation of its node information, which is then embedded as a Grassmann point with a subspace dimension by Singular Value Decomposition (SVD). All Grassmann points that represent corresponding graphs are on the same Grassmann manifold, and they can be projected to Euclidean space in the form of symmetric matrices (Absil et al., 2008), where all graphs acquire the same size representation. The vectorized representations can be sent to a conventional learning model for any learning task such as graph classification or clustering. The architecture design is one practice of exploiting Grassmann representation as a pooling or embedding component. Based on this, we propose Grassmann Graph Pooling (GRPOOL), a feasible graph pooling mechanism for graph data in varying size and structure. The strategy can also inspire other learning tasks, such as graph coarsening and graph clustering.

2 GRASSMANN GRAPH EMBEDDING

We can embed multiple graphs to one Grassmann manifold using their graph convolutional network representation. For a given set of N graphs $\mathbb{G} = \{\mathcal{G}_1, \ldots, \mathcal{G}_N\}$, a sequence of graph convolutional layers like GCN (Kipf & Welling, 2017) and GIN (Xu et al., 2019) maps them to a set of hidden representations $\mathbb{H} = \{\boldsymbol{H}_1, \ldots, \boldsymbol{H}_N\}$ where $\boldsymbol{H}_i \in \mathbb{R}^{n_i \times m}$ is with respect to n_i nodes in $\mathcal{G}_i$, and m denotes the number of hidden units in the last graph convolutional layer. To conduct graph-level learning tasks, one needs to find a uni-dimensional graph representation that is irrelevant to the node size. We accomplish the mission by transforming each graph representation of $\mathbb{H}$ to a Grassmann point, which can later be projected back to its Euclidean form for regular output layers.

Grassmann manifold Grassmann manifold $\mathcal{M}(m, d)$ is the space of all d-dimensional linear subspaces of $\mathbb{R}^m$ for $0 \leq d \leq m$. A point on $\mathcal{M}(m, d)$ is a d-dimensional subspace of $\mathbb{R}^m$, which can be represented by any orthonormal basis $\boldsymbol{U} = [\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_d] \in \mathbb{R}^{m \times d}$. The chosen orthonormal basis is called a representative of its subspace. Grassmann manifold is an abstract quotient manifold, which one can represent in many ways. In this paper, we embed Grassmann manifold into the space of symmetric matrices $\mathrm{Sym}(m)$ by

$$\Pi : \mathcal{M}(m, d) \longrightarrow \mathrm{Sym}(m), \quad \Pi(\boldsymbol{U}) = \boldsymbol{U}\boldsymbol{U}^\top. \tag{1}$$

This method is called *Grassmann embedding*, which we now introduce.

Rectified manifold embedding Suppose we have obtained a graph hidden representation $\boldsymbol{H} \in \mathbb{H} \subset \mathbb{R}^{n \times m}$ and its row-generated subspace $\mathrm{span}(\boldsymbol{H}^\top)$, which as mentioned can be achieved by applying one or multiple layers of graph convolution like GCN. We then go to find a better Grassmann representation for the subspace to extract the most important features of the data. Such representation is characterized by an orthogonal basis of the subspace, which can be rectified in several ways. A typical method as demonstrated in Huang et al. (2018) is QR decomposition of matrix $\boldsymbol{H}^\top$. Our preliminary goal is to bring representation towards a unique subspace dimension k. That is, we need to find out the most representative basis $\boldsymbol{U} = [\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_d]$, thus $\mathcal{U} = [\boldsymbol{U}]$, an equivalence class of $\boldsymbol{U}$, is a point of an underlying Grassmann manifold $\mathcal{M}$. To this end, we employ singular value

Table 1: Performance comparison for graph property prediction. **QM7** is a regression task in MSE; **ogbg-molhiv** is a classification task in ROC-AUC in percentage; others are for classification in test accuracy in percentage. The value after $\pm$ is standard deviation.

		PROTEINS	**D&D**	**NCI1**	**Mutagenicity**	**COLLAB**	**ogbg-molhiv**	**QM7**
GCN Conv	TopKPool	73.48$\pm$3.57	74.87$\pm$4.12	75.11$\pm$3.45	79.84$\pm$2.46	81.18$\pm$0.89	77.11$\pm$1.27	175.41$\pm$3.16
	Attention	73.93$\pm$5.37	77.48$\pm$2.65	74.04$\pm$1.27	80.25$\pm$2.22	81.58$\pm$1.72	77.44$\pm$1.27	177.99$\pm$2.22
	SAGPool	75.89$\pm$2.91	74.96$\pm$3.60	76.30$\pm$1.53	79.86$\pm$2.36	79.26$\pm$5.37	75.36$\pm$1.82	**41.93**$\pm$1.14
	SUM	74.91$\pm$4.08	78.91$\pm$3.37	76.96$\pm$1.70	80.69$\pm$3.26	80.76$\pm$1.56	74.88$\pm$2.64	42.09$\pm$0.91
	AVG	73.13$\pm$3.18	76.89$\pm$2.23	73.70$\pm$2.55	80.37$\pm$2.44	81.24$\pm$1.34	**77.69**$\pm$1.17	177.49$\pm$4.69
	MAX	73.57$\pm$3.94	75.80$\pm$4.11	75.96$\pm$1.82	78.83$\pm$1.70	82.28$\pm$2.10	76.95$\pm$0.94	177.48$\pm$4.70
	GrPool	**77.79**$\pm$2.16	**79.10**$\pm$2.98	**77.80**$\pm$2.01	**81.01**$\pm$1.28	**82.94**$\pm$1.06	76.60$\pm$1.10	172.35$\pm$3.50
GIN Conv	TopKPool	73.66$\pm$6.00	76.40$\pm$2.32	77.06$\pm$0.90	78.30$\pm$1.36	81.40$\pm$0.94	78.14$\pm$0.62	42.68$\pm$0.50
	Attention	75.63$\pm$1.13	71.76$\pm$3.26	78.22$\pm$1.32	78.54$\pm$5.37	83.22$\pm$0.30	74.44$\pm$2.12	42.67$\pm$0.80
	SAGPool	75.95$\pm$4.52	68.94$\pm$7.62	76.97$\pm$2.94	78.86$\pm$1.58	81.76$\pm$1.57	75.26$\pm$2.29	43.02$\pm$1.54
	SUM	78.04$\pm$2.30	78.57$\pm$1.26	78.83$\pm$1.49	81.31$\pm$1.10	82.64$\pm$0.85	77.41$\pm$1.16	42.41$\pm$3.36
	AVG	71.70$\pm$2.08	74.37$\pm$1.32	76.55$\pm$1.72	80.97$\pm$1.18	**83.30**$\pm$0.77	**78.21**$\pm$0.90	43.16$\pm$4.77
	MAX	76.70$\pm$1.57	77.31$\pm$2.06	79.27$\pm$1.38	80.28$\pm$0.83	80.94$\pm$0.72	78.16$\pm$1.33	42.54$\pm$5.17
	GrPool	**79.80**$\pm$1.09	**81.18**$\pm$1.14	**81.31**$\pm$1.55	**82.53**$\pm$0.72	81.32$\pm$0.68	77.82$\pm$0.90	**42.36**$\pm$3.39

decomposition (SVD) for the hidden feature

$$\boldsymbol{H}^\top = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^\top, \tag{2}$$

where $\boldsymbol{U} \in \mathbb{R}^{m\times k}$ with $\text{rank}(\boldsymbol{H}^\top) = k, 1 \leq k \leq \min\{m, n\}$ is an orthonormal basis that spans the column space of $\boldsymbol{H}^\top$. The diagonal $\boldsymbol{S} := \text{Diag}([\sigma_1, \ldots, \sigma_k]) \in \mathbb{R}^{k\times k}$ contains k singular values sorted in the descending order, where the lth singular value indicates the percentage importance of the lth basis vector. The corresponding singular vectors constitute $\boldsymbol{V} = [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k] \in \mathbb{R}^{n\times k}$.

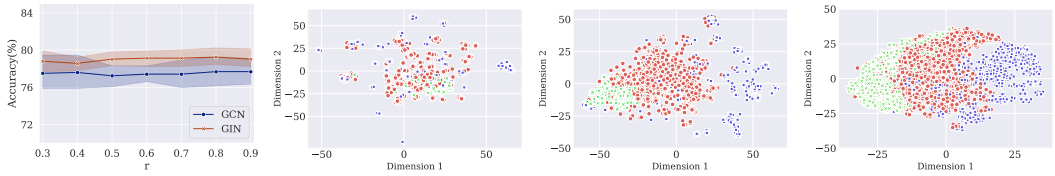
The first d-columns of $\boldsymbol{U}$ ($d \leq k$) from SVD include the d most important components in the original space of $\boldsymbol{H}$, and the subspace it spans naturally composes a Grassmann point $\mathcal{U} := [\boldsymbol{U}_d]$ (the equivalent class under the orthogonal group $\mathcal{O}(d)$) on the Grassmann manifold $\mathcal{M}(m, d)$. This embedding operation can be applied to any hidden representation from $\mathbb{H}$, which intuitively regards a graph with a set of n featured nodes as a d-dimensional subspace of the embedding space $\mathbb{R}^m$. Each graph feature is transformed to a hidden feature, and different graphs (possibly with varying node size) are taken as points on the (same) Grassmann manifold, where similar instances are also geodesically close to each other. Such similarity provides a criterion for clustering or classification. At this point, we have finished embedding graphs to the Grassmann manifold, and each graph is represented by a subspace of orthogonal basis $\boldsymbol{U}_d \in \mathbb{R}^{m\times d}$.

Projection mapping With the above Grassmann embedding, we have a set of embedded Grassmann points $\{\mathcal{U}_1, \ldots, \mathcal{U}_N\} \subset \mathcal{M}(m, d)$ with their relevant orthogonal matrix basis as representatives. Any follow-up learning can then be built on $\{\mathcal{U}_1, \ldots, \mathcal{U}_N\}$ by Grassmann geometry, see Absil et al. (2008). As pointed out in (1), the entire manifold $\mathcal{M}(m, d)$ can be embedded (mapped) into the Euclidean space $\text{Sym}(m)$. This embedding naturally transforms graph features to their Euclidean representation and thus conventional deep learning can be applied.

To be precise, we project the graph latent representation to the symmetric positive definite (SPD) matrix $\Pi(\boldsymbol{U}_d) = \boldsymbol{U}_d\boldsymbol{U}_d^\top$. The resulting SPD matrix representation, an analog to a bilinear mapping, captures a second-order statistics that can better reflect regional features (Tuzel et al., 2006). Moreover, the rectified representation $\boldsymbol{U}$ from (2) can reduce noise, and the approximated covariance matrix $\boldsymbol{U}_d\boldsymbol{U}_d^\top$ is thus more expressive. This projected Euclidean representation can be used for various tasks. For example, in graph property prediction, the Grassmann embedding can be vectorized and output as the readout for graph prediction. In subspace clustering, the $\Pi(\boldsymbol{U}_d)$ with respect to $\boldsymbol{H}$ is analogous to the affinity matrix and it can be used by clustering algorithms for graph segmentation.

3 Experiments for Grassmann pooling

We evaluate the proposed GrPool on a set of graph classification tasks with different graph sizes, volume, and density. We also include a graph regression task with moderate data size.



(a) Sensitivity analysis (b) Raw t-SNE (c) GCN–GRPOOL (d) GIN–GRPOOL

Figure 2: (a) Sensitivity analysis for the threshold information ratio r on **PROTEINS**. The plots (b)-(d) present the t-SNE visualizations of the vectorized graph representations produced by our GRPOOL on **COLLAB** with GCN before training; with GCN after training; and with GIN after training, respectively.

We benchmark the performance on five binary classification, one multi-class classification and one regression tasks. Except for **ogbg-molhiv** (Hu et al., 2020), all datasets are provided by **TUDataset benchmarks** (Morris et al., 2020) with default graph attributes. The performance is compared against two hierarchical (TOPKPOOL (Gao & Ji, 2019; Cătălina et al., 2018), SAGPOOL (Lee et al., 2019)) and four global pooling methods (ATTENTIONPOOL (Li et al., 2016), SUM, AVG and MAX pooling) with two types of network architectures. For completeness, we report prediction results of all models with GCN (Kipf & Welling, 2017) and GIN (Xu et al., 2019) (under JKNET structure (Xu et al., 2018)) convolutional layers. We detail the model setup in Appendix C.

Table 1 compares the prediction performance of GRPOOL with baselines. We report mean test accuracy for the first five classification tasks; ROC-AUC score for **ogbg-molhiv** classification; and mean square error (MSE) for **QM7** regression. All the mean scores are averaged over 10 repetitions. Our GRPOOL achieves the top score with generally a lower volatility on all tasks with primary feature matrices. The advantage is more salient on the bottom half of the table, where we apply JKNET-typed structure with GIN convolution for GRPOOL. It provides a global pooling method and can express more information from its compressed graph representation. The performance of GRPOOL on **COLLAB** and **QM7** is less promising, which is likely due to the lack of non-structural feature information from the dataset. As a result, the abstracted subspaces provide more information compared to the baseline methods with common graph convolutional layers.

4 FURTHER INVESTIGATION ON GRASSMANN EMBEDDING

Real datasets could have a great number of graphs with a large variation on node sizes from a few to thousands. Consequently, defining an identical relatively small subspace dimension d for all graphs could potentially hurt the expressiveness of graph embedding. Instead, we let d of a Grassmann point $[\boldsymbol{U}_d] \in \mathcal{M}(m, d)$ be determined by $d = \sum_{i=1}^{k} \mathbb{1}\{\sigma_i > r\}$. The σ_i corresponds to the singular values from (2), and r denotes the global threshold of the percentage importance. However, all these Grassmannian points $\{\mathcal{U}\}$ can be naturally mapped to the embedded Euclidean space $\mathrm{Sym}(m)$ of the manifold $\mathcal{M}(m, d_{\text{max}})$, with d_{max} the highest d over all Grassmann points.

The r is a hyperparameter of the GRPOOL. However, the choice of r has little impact on the performance of the Grassmann embedding, as demonstrated by our sensitivity analysis. The experiment is conducted on **PROTEINS** with both GCN and GIN network architectures from Section 3. We report the mean test accuracy over 10 repetitions with seven different values of the threshold ratio r, ranging from 0.3 to 0.9 with step size 0.1. All other hyperparameters are tuned in the same way as Section 3. Figure 2a shows that the mean test accuracy is stable over all values of the threshold ratio r. This suggests that the hyperparameter r (within a wide range) has a negligible impact on the performance of our proposed GRPOOL, which indicates that a moderately high value of r (e.g., $r > 0.5$) is sufficient to retain the essential information of a graph in Grassmann embedding.

The rest three plots in Figure 2 exploit two-dimensional t-distributed Stochastic Neighbor Embedding (t-SNE) of flattened Euclidean graph embeddings on **COLLAB**, which is a 3-classification task. Each point denotes a graph representation after GRPOOL, and their color indicates the true labels. We randomly sampled a sufficient amount of $4,000$ instances. As the GIN-GRPOOL model employs JKNET structure, we aggregate all four pooling results as an approximated graph representation. Both Figures 2c and 2d suggest a clear clustering pattern of the pooled graphs.

REFERENCES

P.-A. Absil, R. Mahony, and R. Sepulchre. *Optimization algorithms on matrix manifolds*. Princeton University Press, 2008.

Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann Lecun. Spectral networks and locally connected networks on graphs. In *ICLR*, 2014.

Cangea Cătălina, Petar Veličković, Nikola Jovanović, Thomas Kipf, and Pietro Liò. Towards sparse hierarchical graph classifiers. In *NeurIPS Workshop on Relational Representation Learning*, 2018.

Hongyang Gao and Shuiwang Ji. Graph U-nets. In *ICML*, 2019.

Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *NeurIPS*, volume 70, pp. 1263–1272, 2017.

William L Hamilton. *Graph Representation Learning*. Morgan & Claypool Publishers, 2020.

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. In *NeurIPS*, 2020.

Zhiwu Huang and Luc Van Gool. A Riemannian network for SPD matrix learning. In *AAAI*, volume 31, 2017.

Zhiwu Huang, Jiqing Wu, and Luc Van Gool. Building deep networks on grassmann manifolds. In *AAAI*, volume 32, 2018.

Katsuhiko Ishiguro, Shin-ichi Maeda, and Masanori Koyama. Graph warp module: an auxiliary module for boosting the power of graph neural networks. *arXiv:1902.01020*, 2019.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

Soheil Kolouri, Navid Naderializadeh, Gustavo K. Rohde, and Heiko Hoffmann. Wasserstein embedding for graph learning. In *ICLR*, 2021.

Junhyun Lee, Inyeop Lee, and Jaewoo Kang. Self-attention graph pooling. In *ICML*, 2019.

Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. In *ICLR*, 2016.

Facundo Mémoli. Gromov-Wasserstein distances and the metric approach to object matching. *Foundations of Computational Mathematics*, 11:417–487, 2011.

Christopher Morris, Nils M Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. TUDataset: A collection of benchmark datasets for learning with graphs. In *ICML Workshop Graph Representation Learning and Beyond*, 2020.

Shayle R Searle and Andre I Khuri. *Matrix algebra useful for statistics*. John Wiley & Sons, 2017.

Oncel Tuzel, Fatih Porikli, and Peter Meer. Region covariance: A fast descriptor for detection and classification. In *ECCV*, pp. 589–600. Springer, 2006.

Boyue Wang, Yongli Hu, Junbin Gao, Yanfeng Sun, and Baocai Yin. Low rank representation on Grassmann manifolds. In *Asian Conference on Computer Vision (ACCV)*, 2014.

Boyue Wang, Yongli Hu, Junbin Gao, Yanfeng Sun, Haoran Chen, Muhammad Ali, and Baocai Yin. Locality preserving projections for Grassmann manifold. In *IJCAI*, pp. 2893–2900, 2017.

Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In *ICML*, 2018.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *ICLR*, 2019.

Xuebin Zheng, Bingxin Zhou, Ming Li, Yu Guang Wang, and Junbin Gao. MathNet: Haar-like wavelet multiresolution-analysis for graph representation and learning. *arXiv:2007.11202*, 2020.

Xuebin Zheng, Bingxin Zhou, Junbin Gao, Yu Guang Wang, Pietro Liò, Ming Li, and Guido Montúfar. How framelets enhance graph neural networks. *arXiv:2102.06986*, 2021.

A STABLE BACK-PROPAGATION FOR SVD

As Singular Value Decomposition (SVD) is one of the essential parts of our proposed *Grassmann graph embedding*, we present in this section the derivation for the back-propagation for SVD. This back-propagation is numerically stable, especially in the case when the input matrix has some extremely small singular values. The following materials are mainly based on the note of Townsend[1].

For a matrix $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ with rank $k \leq \min(m,n)$, we decompose $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^\top$ by using SVD, where $\boldsymbol{U}$ is a $m \times k$ matrix, $\boldsymbol{V}$ is in $n \times k$, and $\boldsymbol{S} = \mathrm{diag}(s_1, s_2, \ldots, s_k)$ is a $k \times k$ diagonal matrix which stores the singular values of $\boldsymbol{A}$. Here, we have the constraints that the matrices $\boldsymbol{U}$ and $\boldsymbol{V}$ are orthogonal. We denote the objective function by $f(\boldsymbol{A})$ whose argument is a square matrix $\boldsymbol{A}$. We aim to find the gradient of the objective function: $\overline{\boldsymbol{A}} := \nabla_{\boldsymbol{A}} f$. For simplicity, we employ the shorthand $\overline{\boldsymbol{A}} := \nabla_{\boldsymbol{A}} f$ for the derivation below. By the property of matrix algebra (Searle & Khuri, 2017), we reach the relation

$$\mathrm{d}f(\boldsymbol{A}) = \mathrm{tr}(\overline{\boldsymbol{A}}^\top \mathrm{d}\boldsymbol{A}) = \mathrm{tr}(\overline{\boldsymbol{U}}^\top \mathrm{d}\boldsymbol{U}) + \mathrm{tr}(\overline{\boldsymbol{S}}^\top \mathrm{d}\boldsymbol{S}) + \mathrm{tr}(\overline{\boldsymbol{V}}^\top \mathrm{d}\boldsymbol{V}). \tag{3}$$

To calculate $\overline{\boldsymbol{A}}$, we need the following three differentials

$$\mathrm{d}\boldsymbol{U} = \boldsymbol{U}\left(\boldsymbol{F} \circ \left[\boldsymbol{U}^\top \mathrm{d}\boldsymbol{A}\boldsymbol{V}\boldsymbol{S} + \boldsymbol{S}\boldsymbol{V}^\top \mathrm{d}\boldsymbol{A}^\top \boldsymbol{U}\right]\right) + \left(\boldsymbol{I}_m - \boldsymbol{U}\boldsymbol{U}^\top\right)\mathrm{d}\boldsymbol{A}\boldsymbol{V}\boldsymbol{S}^{-1} \tag{4}$$

$$\mathrm{d}\boldsymbol{S} = \boldsymbol{I}_k \circ \left[\boldsymbol{U}^\top \mathrm{d}\boldsymbol{A}\boldsymbol{V}\right] \tag{5}$$

$$\mathrm{d}\boldsymbol{V} = \boldsymbol{V}\left(\boldsymbol{F} \circ \left[\boldsymbol{S}\boldsymbol{U}^\top \mathrm{d}\boldsymbol{A}\boldsymbol{V} + \boldsymbol{V}^\top \mathrm{d}\boldsymbol{A}^\top \boldsymbol{U}\boldsymbol{S}\right]\right) + \left(\boldsymbol{I}_n - \boldsymbol{V}\boldsymbol{V}^\top\right)\mathrm{d}\boldsymbol{A}^\top \boldsymbol{U}\boldsymbol{S}^{-1}, \tag{6}$$

where $\boldsymbol{F}_{ij} = \frac{1}{s_j^2 - s_i^2} \cdot \mathbb{1}\{i \neq j\}$ which satisfies the identity $\boldsymbol{F}^\top = -\boldsymbol{F}$. The calculations of $\boldsymbol{F}$ and $\boldsymbol{S}^{-1}$ are often numerically unstable due to the possible near-zero singular values. To circumvent this difficulty, we employ the following improved strategy, as similarly used by Huang & Van Gool (2017):

$$\boldsymbol{S}_{i,i}^{\mathrm{new}} = \boldsymbol{S}_{i,i} \cdot \mathbb{1}\{\boldsymbol{S}_{i,i} > \epsilon\} + \epsilon \cdot \mathbb{1}\{\boldsymbol{S}_{i,i} \leq \epsilon\},$$

where ϵ is a small number and can usually be set to 10^{-12}. In practice, we can replace $\boldsymbol{S}$ by the modified matrix $\boldsymbol{S}^{\mathrm{new}}$ in the back-propagation formulas.

We evaluate the three terms in (3) respectively, as follows. By (5), we can write $\mathrm{tr}(\overline{\boldsymbol{S}}^\top \mathrm{d}\boldsymbol{S})$ as

$$\begin{aligned}
\mathrm{tr}(\overline{\boldsymbol{S}}^\top \mathrm{d}\boldsymbol{S}) &= \mathrm{tr}\left(\overline{\boldsymbol{S}}^\top \left(\boldsymbol{I}_k \circ \left[\boldsymbol{U}^\top \mathrm{d}\boldsymbol{A}\boldsymbol{V}\right]\right)\right) \\
&= \mathrm{tr}\left(\boldsymbol{U}^\top \mathrm{d}\boldsymbol{A}\boldsymbol{V}\left(\boldsymbol{I}_k \circ \overline{\boldsymbol{S}}\right)\right) \\
&= \mathrm{tr}\left(\boldsymbol{V}\left(\boldsymbol{I}_k \circ \overline{\boldsymbol{S}}\right)\boldsymbol{U}^\top \mathrm{d}\boldsymbol{A}\right).
\end{aligned} \tag{7}$$

Using (4) for the term $\mathrm{tr}(\overline{\boldsymbol{U}}^\top \mathrm{d}\boldsymbol{U})$ yields

$$\mathrm{tr}(\overline{\boldsymbol{U}}^\top \mathrm{d}\boldsymbol{U}) = \mathrm{tr}\left(\overline{\boldsymbol{U}}^\top \left[\boldsymbol{U}\left(\boldsymbol{F} \circ \left[\boldsymbol{U}^\top \mathrm{d}\boldsymbol{A}\boldsymbol{V}\boldsymbol{S} + \boldsymbol{S}\boldsymbol{V}^\top \mathrm{d}\boldsymbol{A}^\top \boldsymbol{U}\right]\right) + \left(\boldsymbol{I}_m - \boldsymbol{U}\boldsymbol{U}^\top\right)\mathrm{d}\boldsymbol{A}\boldsymbol{V}\boldsymbol{S}^{-1}\right]\right).$$

The above expression is a sum of two terms. By $\boldsymbol{F}^\top = -\boldsymbol{F}$, these two terms can be simplified as

$$\begin{aligned}
&\mathrm{tr}\left(\overline{\boldsymbol{U}}^\top \boldsymbol{U}\left(\boldsymbol{F} \circ \left[\boldsymbol{U}^\top \mathrm{d}\boldsymbol{A}\boldsymbol{V}\boldsymbol{S} + \boldsymbol{S}\boldsymbol{V}^\top \mathrm{d}\boldsymbol{A}^\top \boldsymbol{U}\right]\right)\right) \\
&= \mathrm{tr}\left(\left[\boldsymbol{U}^\top \mathrm{d}\boldsymbol{A}\boldsymbol{V}\boldsymbol{S} + \boldsymbol{S}\boldsymbol{V}^\top \mathrm{d}\boldsymbol{A}^\top \boldsymbol{U}\right]\left(\boldsymbol{F} \circ \boldsymbol{U}^\top \overline{\boldsymbol{U}}\right)\right) \\
&= \mathrm{tr}\left(\boldsymbol{V}\boldsymbol{S}\left(\boldsymbol{F} \circ \boldsymbol{U}^\top \overline{\boldsymbol{U}}\right)\boldsymbol{U}^\top \mathrm{d}\boldsymbol{A} - \boldsymbol{V}\boldsymbol{S}\left(\boldsymbol{F} \circ \overline{\boldsymbol{U}}^\top \boldsymbol{U}\right)\boldsymbol{U}^\top \mathrm{d}\boldsymbol{A}\right) \\
&= \mathrm{tr}\left(\boldsymbol{V}\boldsymbol{S}\left(\boldsymbol{F} \circ \left[\boldsymbol{U}^\top \overline{\boldsymbol{U}} - \overline{\boldsymbol{U}}^\top \boldsymbol{U}\right]\right)\boldsymbol{U}^\top \mathrm{d}\boldsymbol{A}\right)
\end{aligned}$$

and

$$\mathrm{tr}\left(\overline{\boldsymbol{U}}^\top \left(\boldsymbol{I}_m - \boldsymbol{U}\boldsymbol{U}^\top\right)\mathrm{d}\boldsymbol{A}\boldsymbol{V}\boldsymbol{S}^{-1}\right) = \mathrm{tr}\left(\boldsymbol{V}\boldsymbol{S}^{-1}\overline{\boldsymbol{U}}^\top \left(\boldsymbol{I}_m - \boldsymbol{U}\boldsymbol{U}^\top\right)\mathrm{d}\boldsymbol{A}\right).$$

Thus, an equivalent expression for the term $\mathrm{tr}(\overline{\boldsymbol{U}}^\top \mathrm{d}\boldsymbol{U})$ reads

$$\mathrm{tr}\left(\overline{\boldsymbol{U}}^\top \mathrm{d}\boldsymbol{U}\right) = \mathrm{tr}\left(\boldsymbol{V}\left[\boldsymbol{S}\left(\boldsymbol{F} \circ \left[\boldsymbol{U}^\top \overline{\boldsymbol{U}} - \overline{\boldsymbol{U}}^\top \boldsymbol{U}\right]\right)\boldsymbol{U}^\top + \boldsymbol{S}^{-1}\overline{\boldsymbol{U}}^\top \left(\boldsymbol{I}_m - \boldsymbol{U}\boldsymbol{U}^\top\right)\right]\mathrm{d}\boldsymbol{A}\right). \tag{8}$$

[1] https://j-towns.github.io/papers/svd-derivative.pdf

Table 2: Summary of the datasets for the graph property prediction tasks.

Datasets	**PROTEINS**	**D&D**	**NCI1**	**Mutagenicity**	**COLLAB**	**ogbg-molhiv**	**QM7**
# graphs	$1,113$	$1,178$	$4,110$	$4,337$	$5,000$	$41,127$	$7,165$
# classes	2	2	2	2	3	2	1 (R)
Min # nodes	4	30	3	30	32	2	4
Max # Nodes	620	$5,748$	111	417	492	222	23
Avg # nodes	39	284	30	30	74	26	15
Avg # edges	73	716	32	31	$2,458$	28	123
# Features	3	89	37	14	0	9	0

Similarly, the term $\mathrm{tr}(\overline{\boldsymbol{V}}^\top \mathrm{d}\boldsymbol{V})$ has the following form after some simple algebraic manipulations

$$\mathrm{tr}\left(\overline{\boldsymbol{V}}^\top \mathrm{d}\boldsymbol{V}\right) = \mathrm{tr}\left(\left[\boldsymbol{V}\left(\boldsymbol{F} \circ \left[\boldsymbol{V}^\top \overline{\boldsymbol{V}} - \overline{\boldsymbol{V}}^\top \boldsymbol{V}\right]\right)\boldsymbol{S} + \left(\boldsymbol{I}_m - \boldsymbol{V}\boldsymbol{V}^\top\right)\overline{\boldsymbol{V}}\boldsymbol{S}^{-1}\right]\boldsymbol{U}^\top \mathrm{d}\boldsymbol{A}\right). \tag{9}$$

Finally, we substitute (7)–(9) into (3) and then retrieve the expression for $\boldsymbol{B}$ to obtain the solution

$$\begin{aligned}\boldsymbol{B} = &\left[\boldsymbol{U}\left(\boldsymbol{F} \circ \left[\boldsymbol{U}^\top \overline{\boldsymbol{U}} - \overline{\boldsymbol{U}}^\top \boldsymbol{U}\right]\right)\boldsymbol{S} + \left(\boldsymbol{I}_m - \boldsymbol{U}\boldsymbol{U}^\top\right)\overline{\boldsymbol{U}}\boldsymbol{S}^{-1}\right]\boldsymbol{V}^\top + \\ &\boldsymbol{U}\left(\boldsymbol{I}_k \circ \overline{\boldsymbol{S}}\right)\boldsymbol{V}^\top + \boldsymbol{U}\left[\boldsymbol{S}\left(\boldsymbol{F} \circ \left[\boldsymbol{V}^\top \overline{\boldsymbol{V}} - \overline{\boldsymbol{V}}^\top \boldsymbol{V}\right]\right)\boldsymbol{V}^\top + \boldsymbol{S}^{-1}\overline{\boldsymbol{V}}^\top\left(\boldsymbol{I}_n - \boldsymbol{V}\boldsymbol{V}^\top\right)\right] = \overline{\boldsymbol{A}}.\end{aligned}$$

B Essential Requirements for Graph Pooling

In this section, we check the two essential requirements of a valid graph pooling method, which are *graph embedding with a unified size* and *permutation invariance*.

Proposition 1. GrPool *always produces a graph embedding $g \in \mathbb{R}^{\frac{m(m+1)}{2}}$ for the node representation matrix $\boldsymbol{H} \in \mathbb{R}^{n\times m}$, regardless of the graph size n.*

Proof. Given the node representation matrix $\boldsymbol{H} \in \mathbb{R}^{n\times m}$ of a graph $\mathcal{G}$ with n nodes and m features, the Grassmann graph embedding gives the output $\boldsymbol{U}\boldsymbol{U}^\top$, where $\boldsymbol{U} \in \mathbb{R}^{m\times k}$ with $k = \mathrm{rank}(\boldsymbol{H}^\top)$. Then, the output of GrPool is the flattened representation of the upper triangular matrix of $\boldsymbol{U}\boldsymbol{U}^\top \in \mathbb{R}^{m\times m}$, which is independent of the graph size n. □

Proposition 2. GrPool *satisfies the requirement of permutation invariance so that it produces the same Grassmann graph embedding under row permutations of the input node representation matrix.*

Proof. Suppose $\boldsymbol{H}_1$ is the node representation matrix of a graph and let $\boldsymbol{H}_2 = \boldsymbol{P}\boldsymbol{H}_1$, where $\boldsymbol{P}$ is a permutation matrix. Then,

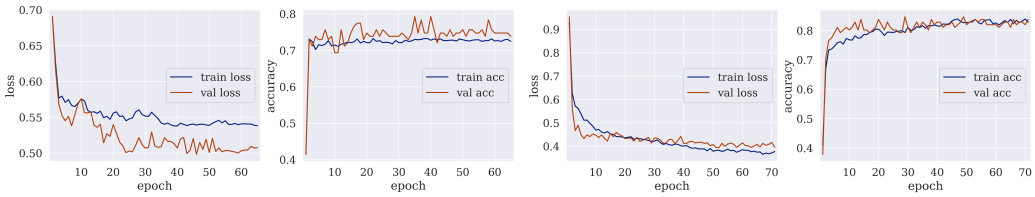
$$\boldsymbol{H}_1^\top = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^\top, \quad \boldsymbol{H}_2^\top = \boldsymbol{H}_1^\top \boldsymbol{P}^\top = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^\top \boldsymbol{P}^\top.$$

The Grassmann point for both $\boldsymbol{H}_1$ and $\boldsymbol{H}_2$ can be accessed by the same matrix $\boldsymbol{U}$. Hence, the proposed graph embedding method is permutation invariant. □

C Experiment

C.1 More on model training

We supplement below a summary of the datasets for graph property prediction tasks in Section 3. The 'R' in the bracket of the last row of **QM7** represents the regression task. For the two feature-less datasets **COLLAB** and **QM7**, we follow Zheng et al. (2020) to create the constant uninformative node features for **QM7**; and follow Xu et al. (2019) to apply one-hot encoding of node degrees for **COLLAB**. For **ogbg-molhiv**, virtual nodes (Ishiguro et al., 2019) are used in the GNNs.



(a) GCN Convolution (b) GIN Convolution

Figure 3: Training and Validation learning curves on **PROTEINS** with GRPOOL.

In the ablation, we consider two graph convolutions to be connected with pooling methods: GCN (Kipf & Welling, 2017) and GIN (Xu et al., 2019). For **TU Dataset**s, the first set connects two GCN convolutional layers with one pooling layer except for **NCI1**, where we add a third convolutional layer. We also study the performance of the second GNN architecture with four GIN convolutional layers and JKNET structure (Xu et al., 2018). For the **ogbg-molhiv**, both models use four convolutional layers. The hidden units of the one-layer MLP for baselines are set identical to that of the convolutional layer. For GRPOOL, it is set to 64 for the first layer and 16 for the second.

Table 3: Search space for hyperparameter tuning.

Hyperparameter	**Searching Space**
Learning rate	$5\mathrm{e}{-3}, 1\mathrm{e}{-3}, 5\mathrm{e}{-4}$
Weight decay (L_2)	$5\mathrm{e}{-3}, 1\mathrm{e}{-3}, 5\mathrm{e}{-4}$
Threshold ratio r	$0.5, 0.8$
Hidden units	$32, 64$
Dropout ratio	$0, 0.5$

Below we provide the searching space of the key hyperparameters that are fine-tuned with grid search, including learning rate, weight decay, hidden unit numbers of convolutional layers, and dropout ratio in MLP layers. Unless stated, models on **OGB** applies the same searching space. Other hyperparameters, if not specified, are set to default values.

For model training and evaluation, we split each dataset into 80% training, 10% validation, and 10% test samples. The stopping criterion is defined as the validation loss stops improving for 20 consecutive epochs, or the maximum 200 epochs is reached. In Table 1, except for those on **COLLAB**, the results of baselines with GCN on **TU Dataset**s are retrieved from Zheng et al. (2021).

C.2 LEARNING CURVE AND CONVERGENCE

The training and validation curves for loss and accuracy are shown in Figure 3. The results are retrieved from a (random) single run on **PROTEINS**. The hyperparameters and network architectures follow exactly the same routine as Section 3. We only display results from one repetition due to the employment of early stopping criteria, where each independent run would stop at an individual epoch. The minor volatility after epoch 5 is generally brought about by stochastic gradients. All four visualizations validate an efficient convergence of our GRPOOL, where the loss curve stabilizes quickly after a few epochs. The training process is slightly longer for GIN convolution, which is partly due to the more sophisticated network architecture for the model to fit.