

Two-Stage Training of Graph Neural Networks for Graph Classification

Manh Tuan Do¹, Noseong Park², Kijung Shin³

¹Graduate School of AI, KAIST

²Department of AI, Yonsei University

³Graduate School of AI & School of Electrical Engineering, KAIST

manh.it97@kaist.ac.kr, noseong@yonsei.ac.kr, kijungs@kaist.ac.kr

Abstract

Graph neural networks (GNNs) have received massive attention in the field of machine learning on graphs. Inspired by the success of neural networks, a line of research has been conducted to train GNNs to deal with various tasks, such as node classification, graph classification, and link prediction. In this work, our task of interest is graph classification. Several GNN models have been proposed and shown great accuracy in this task. However, the question is whether usual training methods fully realize the capacity of the GNN models.

In this work, we propose a two-stage training framework based on triplet loss. In the first stage, GNN is trained to map each graph to a Euclidean-space vector so that graphs of the same class are close while those of different classes are mapped far apart. Once graphs are well-separated based on labels, a classifier is trained to distinguish between different classes. This method is generic in the sense that it is compatible with any GNN model. By adapting five GNN models to our method, we demonstrate the consistent improvement in accuracy and utilization of each GNN’s allocated capacity over the original training method of each model up to 5.4% points in 12 datasets.

1 Introduction

With the pervasiveness of graph-structured data, graph representation learning has become an increasingly important task. Its goal is to learn meaningful embeddings (i.e., vector representations) of nodes and/or (sub)graphs. These embeddings can be used in various downstream tasks, such as node classification, link prediction, and graph classification.

Metric learning is about learning distance between objects in a metric space. While it remains a difficult task to properly define an efficient metric measure directly based on graph topology, a common approach is to map the graphs into vectors in the Euclidean space and measure the distance between those vectors. In addition to satisfying the basic properties of metrics, this mapping is also expected to separate graphs of different classes to distinguishable clusters.

Table 1: Comparison of our method 2STG+ with [Hu *et al.*, 2020]. While both need a pre-training step before graph classification, our method consistently improves accuracy of several GNNs even in datasets of a far domain while requiring shorter training time and no additional rich dataset.

	2STG+ (Proposed)	[Hu <i>et al.</i> , 2020]
Accuracy improvement	5 out of 5 GNNs All datasets	3 out of 4 GNNs Only within domains
Required datasets	No additional set	Large ($\approx 400K$ graphs)
Total training time	Short (≈ 1 hour)	Long (≈ 1 day)

Graph neural networks (GNNs) have received a lot of attention in the graph mining literature. Despite the challenge of applying the message-passing mechanism of neural networks to the graph structure, GNNs have proved successful in dealing with graph learning problems, including node classification [Veličković *et al.*, 2018; Kipf and Welling, 2017], link prediction [Schlichtkrull *et al.*, 2018] and graph classification [Zhang *et al.*, 2018; Dai *et al.*, 2016; Duvenaud *et al.*, 2015]. The common approach is to start from node features, allow information to flow between neighboring nodes and finalize the meaningful node embeddings. GNN models differ by the information-passing method and the objectives of the final embeddings.

Graph classification involves separating graph instances of different classes and predicting the label of an unknown graph. This task requires a graph representation vector distinctive enough to distinguish graphs of different classes. The subtlety is how to combine the node embeddings into an expressive graph representation vector, and a number of approaches have been proposed.

Although GNNs are shown to achieve high accuracy of graph classification, we observe that, with usual end-to-end training methods, they cannot realize their full potential. Thus, we propose 2STG+, a new training method with two stages. The first stage is metric learning with triplet loss, and the second stage is training a classifier. We observed that 2STG+ significantly improves the accuracy of five different GNN models, compared to their original training methods.

Our training method is, to some extent, similar to [Hu *et al.*, 2020] in the sense that GNNs are pre-trained on a task before being used for graph classification. However, [Hu *et al.*, 2020] does transfer learning by pre-training GNNs on a different massive graph, either in chemistry or biology domain,

with numerous tasks, on both node and graph levels. On the other hand, 2STG+ pre-trains GNNs on the same training dataset with only one graph-level task as the first stage. As highlighted in Table 1, 2STG+ is faster without requiring pre-training on rich and massive datasets, and it consistently achieves improved accuracy of more GNN models in more datasets than [Hu *et al.*, 2020].

In short, the contributions of our paper are three-fold.

- **Observation:** In the graph classification task, GNNs often fail to exhibit their full power. Using a proper training method, their expressiveness can be further utilized.
- **Method Design:** We propose a two-stage learning method with pre-training based on triplet loss. With this method, up to 5.4% points in accuracy can be increased. Moreover, our method also utilizes the capacity of each GNN better by producing embeddings with higher intrinsic dimension and weaker correlation between embedding dimensions.
- **Extensive Experiments:** We conducted comprehensive experiments with 5 different GNN models and 12 datasets to illustrate the consistent improvement in accuracy and capacity utilization by our two-stage training method. We also compare our method with a strong graph transfer-learning framework to highlight its competency of our method.

2 Related Works

2.1 Graph neural networks

Graph neural networks (GNNs) attempt to learn embeddings (i.e., vector representations) of nodes and/or graphs, utilizing the mechanisms of neural networks adapted to the topology of graphs. The core idea of GNNs is to allow messages to pass between neighbors so that the representation of each node can incorporate the information from its neighborhood and thus to enable the GNNs to indirectly learn the graph structures. Numerous novel architectures for GNNs have been proposed and tested, which differ by the information-passing mechanisms. Among the most recent architectures are graph convolutions [Kipf and Welling, 2017], attention mechanisms [Veličković *et al.*, 2018], and those inspired by convolutional neural networks [Gao and Ji, 2019; Niepert *et al.*, 2016; Defferrard *et al.*, 2016]. The final embeddings obtained from GNNs can be utilized for various graph mining tasks, such as node classification [Kipf and Welling, 2017], link prediction [Schlichtkrull *et al.*, 2018], and graph classification [Zhang *et al.*, 2018; Dai *et al.*, 2016].

2.2 Graph classification by GNN

In graph classification, GNNs are tasked with predicting the label of an unseen graph. While node embeddings can be updated within a graph, the elusive step here is how to combine them into a vector representation of the entire graph that can distinguish among different labels. Two of the most common approaches are global pooling [Duvenaud *et al.*, 2015] and hierarchical pooling [Ying *et al.*, 2018; Ma *et al.*, 2019; Lee *et al.*, 2019]. The simplest ways for global pooling are global mean and global max of the final node embeddings. In contrast, hierarchical pooling iteratively reduces the number of nodes either by merging similar nodes into supern-

odes [Ying *et al.*, 2018; Ma *et al.*, 2019] or selecting most significant nodes [Lee *et al.*, 2019] until reaching a final supernode whose embedding is used to represent the whole graph.

2.3 Transfer learning for graphs

While most existing methods attempt to train GNNs as an end-to-end classification system, some studies considered transfer learning in which the GNN is trained on a large dataset before being applied to the task of interest, often in a much smaller dataset. [Hu *et al.*, 2020] succeeded in improving 3 (out of 4 attempted) existing GNNs by transfer learning from other tasks. Rather than training a GNN to classify a dataset right away, the authors pre-trained that GNN on another massive dataset (up to 456K graphs); then they added a classifier and trained the whole architecture on the graph classification task. However, transfer learning for graph remains a major challenge, as [Ching *et al.*, 2018; Wang *et al.*, 2019] pointed out that considerable domain knowledge is required to design the appropriate pre-training procedure.

2.4 Metric learning

Metric learning aims to approximate a real-valued distance between two objects, and the most common objectives are contrastive loss and triplet loss. Some work has focused on metric learning on graphs [Ktena *et al.*, 2018; Liu *et al.*, 2019; Ling *et al.*, 2020]. Some of them [Ktena *et al.*, 2018; Liu *et al.*, 2019] employ a Siamese network structure, in which a twin network sharing the same weights is applied on a pair of graphs, and the two output vectors acting as representation of the two graphs are passed through a distance measure.

In computer vision, [Schroff *et al.*, 2015] learns metric on triplet of images, where two (anchor and positive) share the same label and one (negative) has a different label. The model aims to minimize the distance between the anchor and the positive, while maximizing the distance between the anchor and the negative. This inspired our interest in learning graph metrics with a triplet loss.

3 Proposed Method

In this section, we first define our task of interest: graph classification. We then describe each component of our proposed training method of GNNs for graph classification.

Problem definition

We tackle the task of graph classification. Given $\mathcal{D} = \{(G_1, y_1), (G_2, y_2), \dots\}$ where $y_i \in \mathcal{Y}$ is the class label of the graph $G_i \in \mathcal{G}$, the goal of graph classification is to learn a mapping $f : \mathcal{G} \rightarrow \mathcal{Y}$ that maps graphs to the set of class labels and predicts the class labels of unknown graphs.

Outline of our method

Our method combines the advantages of both GNN and metric learning. Specifically, to facilitate a better accuracy of the classifier, our method first maps input graphs into vectors in the Euclidean space such that their corresponding vectors are well-separated based on classes. Below, we first briefly introduce GNNs and a learning scheme on triplet loss. We then describe the two stages of our method: pre-training a GNN and training a classifier.

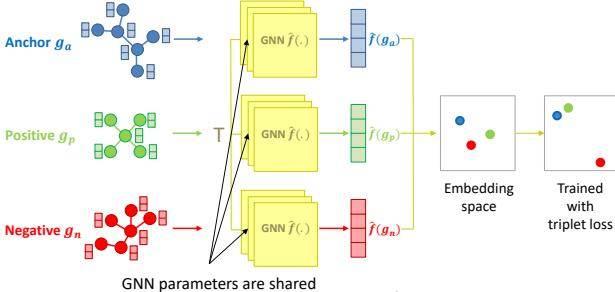


Figure 1: First training stage for a GNN. $\hat{f}(\cdot)$ is trained to differentiate graphs of different classes via the triplet loss.

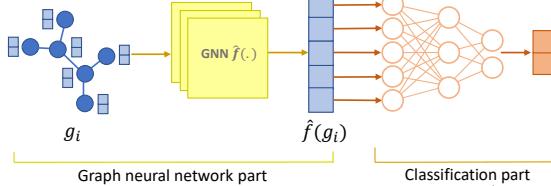


Figure 2: Second training stage for a classifier. After $\hat{f}(\cdot)$ maps g_i to $\hat{f}(g_i)$, a classifier is added to map $\hat{f}(g_i)$ to the class probability prediction. At this step, either the classifier is trained independently (2STG) or the whole architecture is trained together (2STG+).

3.1 Graph neural networks

Various GNNs have been proposed and proven effective in approximating such a function f . Starting from a graph $G = (V, E)$ with node features $X_G = \{x_v | v \in V\}$, GNNs obtain final embeddings of nodes $Z_G = \{z_v | v \in V\}$ and a final embedding h_G of the graph after K layers of aggregation. Specifically, at each $(k+1)$ -th layer, the embedding $h_v^{(k+1)}$ of each node v incorporates the embeddings of itself and its neighboring nodes $N(v)$ at the k -th layer as follows:

$$h_v^{(k+1)} = \text{MERGE}\left(h_v^{(k)}, \text{AGGR}\left(\{h_u^{(k)} | u \in N(v)\}\right)\right)$$

The embedding h_G of the graph G is then obtained by pooling all node embeddings into a single vector as follows:

$$h_G = \text{POOL}\left(\{h_v^{(i)} | v \in V; i = 1, \dots, K\}\right)$$

GNNs differ by how the incorporating function MERGE , the aggregating function AGGR , and the final pooling function POOL are implemented.

3.2 Metric learning based on triplet loss

Triplet loss was first introduced in [Schroff *et al.*, 2015]. The core idea is to enforce a margin between classes of samples. This results in embeddings of the same class mapped to a cluster distant apart from that of other classes. Specifically, given a mapping \bar{f} , we wish for a graph g_a (anchor) to be closer to another graph g_p (positive) of the same class than to a graph g_n (negative) of another class by at least a margin α , which is a hyperparameter, i.e.,

$$\|\bar{f}(g_a) - \bar{f}(g_p)\|_2^2 + \alpha < \|\bar{f}(g_a) - \bar{f}(g_n)\|_2^2.$$

The triplet loss for the whole dataset becomes:

$$\sum_{(a,p,n)} \max (\|\bar{f}(g_a) - \bar{f}(g_p)\|_2^2 - \|\bar{f}(g_a) - \bar{f}(g_n)\|_2^2 + \alpha, 0)$$

with the summation over all considered triplets.

Our two-stage method combines the power of both GNNs and the metric learning method, as described below.

3.3 First training stage (pre-training a GNN)

In the first training stage (depicted in Fig. 1), given a GNN architecture $\hat{f}(\cdot)$, its weights are shared among a triplet network T , which consists of three identical GNN architectures having the same weights as $\hat{f}(\cdot)$. The parameters of T are trained on each triplet of graphs (g_a, g_p, g_n) (anchor, positive, negative), in which the anchor and the positive graphs are of the same class while the negative graph is of another class. T maps a triplet of graphs to a triplet of real-valued vectors in the Euclidean space: $T(g_a, g_p, g_n) = (\hat{f}(g_a), \hat{f}(g_p), \hat{f}(g_n))$. Ideally, $\hat{f}(g_a)$ and $\hat{f}(g_p)$ should be close while $\hat{f}(g_n)$ is far from them both. The triplet loss for (g_a, g_p, g_n) is defined as:

$$\max \left(\|\hat{f}(g_a) - \hat{f}(g_p)\|_2^2 - \|\hat{f}(g_a) - \hat{f}(g_n)\|_2^2 + \alpha, 0 \right)$$

3.4 Second training stage (training a classifier)

In the second stage, a classifier is either trained independently, or added on top of the trained GNN and trained together on the graph classification task (see Fig. 2).

In summary, we propose two training methods for GNNs: 2STG and 2STG+, both consist of two stages.

- **2STG (Pre-training Setting):** In the first stage, the GNN maps each triplet of graphs to a triplet of Euclidean-space vectors, and the GNN is trained on triplet loss. In the second stage, a classifier is trained independently to classify the graph embeddings.
- **2STG+ (Fine-tuning setting):** It has the same structure as 2STG except that in the second stage, the classifier is plugged on top of the trained GNN, and then the whole architecture is trained together in an end-to-end manner.

Note that our methods are compatible to any GNN model that maps each graph to a representation vector. As shown in the next section, when applied to this method, each GNN model outperformed itself in the original setting.

4 Experiments

In this section, we describe the details of our experiments.

4.1 GNN architectures

In order to demonstrate that our two-stage method helps realize better accuracy of GNNs, for each of the following GNN model, we compare the accuracy obtained in the original setting versus that from our method:

- **GRAPH SAGE** [Hamilton *et al.*, 2017]: Global pooling is used to pool node embeddings into one graph embedding.
- **GAT** [Veličković *et al.*, 2018]: Global pooling is also applied on the node embeddings.
- **DIFFPOOL** [Ying *et al.*, 2018]: A hierarchical approach for pooling the node embeddings.
- **EIGENGCN** [Ma *et al.*, 2019]: A different design for hierarchical pooling.
- **SAGPOOL** [Lee *et al.*, 2019]: A hierarchical graph pooling with self-attention mechanisms.

In previous studies, these models were trained end-to-end, mapping each graph to a prediction of class probabilities. To further illustrate the competency of our method, we also compare it with a transfer-learning method [Hu *et al.*, 2020].

4.2 Datasets

To validate the claims, we apply our methods on 12 datasets. Some statistics of the datasets are summarized in Table. 2.

Benchmark datasets

They include some commonly tested binary-class datasets [Morris *et al.*, 2020] for graph classification: DD, MUTAG, MUTAGENICITY, PTC-FM, PROTEINS, and IMDB-B.

New York Taxi datasets

In addition, we also test our method on NYC Taxi datasets.¹. We extracted the taxi ridership data in 2019 from New York City (NYC) Taxi Commission. The areas in New York are represented as nodes, and each taxi trip is an edge connecting the source and destination nodes. All taxi trips in an 1-hour interval form a graph, and each dataset spans a month of taxi operations. We augmented the binary label for each graph as taxi trips in weekdays (Mon-Thu) vs. weekend (Fri-Sun). We considered two taxi operators (Yellow and Green) and processed data in January, February and March of 2019, making 6 datasets in total: JAN. G., FEB. G., MAR. G., JAN. Y., FEB. Y., and MAR. Y..

4.3 Experimental procedure

We test the ability of each GNN architecture to classify graphs in the following three settings:

- Original setting: The GNN with a final classifier outputs the estimated class probabilities, and the weights are updated by the cross-entropy loss with respect to the ground truth. We use the implementation provided by the authors. To enhance the capacity of the final classifier, we tune it by using up to three fully-connected layers and select the model based on validation sets.
- 2STG and 2STG+: See Section 3.

We also compare our two-stage method with the transfer-learning method in [Hu *et al.*, 2020], which also claims the effectiveness of a pre-training strategy. Out of the 5 GNN models investigated in our work, GRAPHsAGE and GAT are provided with trained weights by [Hu *et al.*, 2020], and they are compared with GRAPHsAGE/GAT trained in 2STG+.

Each dataset is randomly split into three sets: train (80%), validation (10%) and test (10%). Details about hyperparameter search are in Appendix C. The reported results are mean and standard deviation of test accuracy of five splits.

We initialize node features as learnable features that are optimized with GNN parameters during training. While input features are provided in some datasets, we empirically find that using learnable features often leads to better accuracy.

In 2STG and 2STG+, each graph can be anchor once, while the positive and negative graphs are chosen randomly. The classifier is a multi-layer perceptron (MLP).

¹<https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page>

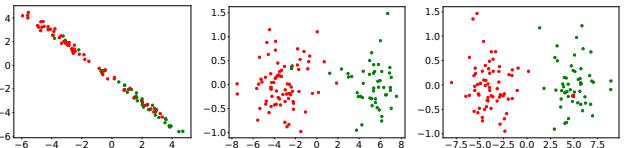


Figure 3: Visualization of the final embeddings in Original (left), 2STG (middle) and 2STG+ (right) settings (DD dataset). Instances of the two classes are separated better in 2STG and 2STG+ than in the original setting.

4.4 Results and discussion

Improvement by our methods

We draw some observations from the results of comparing 2STG and 2STG+ with the original setting (Tables 3 and 4):

- Pre-training using triplet loss (i.e., the first stage of 2STG and 2STG+) consistently enhances the graph classification accuracy of each GNN model by 0.9 – 5.4% points, compared to its original setting.
- Fine-tuning the weights of GNNs (i.e, the second stage of 2STG+) further improves the accuracy from 2STG in some cases by up to 1.3% points.

The improvement in accuracy by 2STG and 2STG+ suggests two possible explanations:

- The end-to-end training methods fail to realize the full potential of the GNN models. Even if the final classifier of is upgraded from a fully-connected layer to an MLP, the accuracy is not as high as in 2STG and 2STG+.
- Learning meaningful embeddings in between that are fairly separated based on classes (see Figure 3), for example through metric learning as in our methods, facilitates a better accuracy of the final classifier.

Comparison with a transfer learning method

We compare the pre-trained models of GRAPHsAGE/GAT in [Hu *et al.*, 2020] with GRAPHsAGE/GAT of the same architecture (5 layers, 300 dimensional hidden units and global mean pooling) trained in 2STG+. The pre-trained models are fine-tuned on the datasets. The mean and standard deviation of test accuracy of 5 splits are reported in Tables 5 and 6. Despite being pre-trained on a much smaller dataset, 2STG+ achieves better accuracy in 83% of the considered cases: up to 2% points in the benchmark datasets and up to 3% points in the NYC Taxi datasets. This validates our claims in Table 1.

Running time

For each hyperparameter setting in a dataset, the original setting takes up to half an hour to train a model. Due to having two stages, 2STG and 2STG+ take up to an hour for both stages. In [Hu *et al.*, 2020], the transfer-learning method was reported to take up to one day to pre-train on a rich dataset.

4.5 Further analysis

We study the final embeddings, generated by three training methods: original, 2STG, 2STG+. In Fig. 3, despite having 2 dimensions, the embeddings of the original setting can be reduced to 1 dimension. We suspect that given the potential capacity, specifically the dimension of the final embeddings, 2STG and 2STG+ utilize such expressive power better than

Table 2: Some statistics of the datasets considered in this paper.

Method	Dataset											
	DD	MUTAG	MUTAGENICITY	PTC-FM	PROTEINS	IMDB-B	JAN. G.	FEB. G.	MAR. G.	JAN. Y.	FEB. Y.	MAR. Y.
# Graphs	1,168	188	4,337	349	1,113	1,000	744	648	744	744	648	744
Avg. # Nodes	268.71	17.93	29.76	14.11	39.05	19.77	174.25	175.28	174.43	203.04	199.28	207.24
Avg. # Edges	676.21	19.79	30.76	14.48	72.81	96.53	497.35	502.96	480.33	1865.66	1868.28	1967.59
# Classes	2	2	2	2	2	2	2	2	2	2	2	2

Table 3: Average and standard deviation (in %) of graph classification accuracies in benchmark datasets in three settings: original, 2STG, and 2STG+. Pre-training GNNs in 2STG improves the classification accuracy compared to the original setting, and fine-tuning GNNs in 2STG+ further improves the accuracy. The average gain is in % points.

Method	Dataset						Average Gain (in % points)
	DD	MUTAG	MUTAGENICITY	PTC-FM	PROTEINS	IMDB-B	
GRAPH SAGE	69.24 ± 0.52	65.13 ± 0.87	75.44 ± 0.50	61.77 ± 1.11	71.25 ± 1.38	65.52 ± 0.96	-
GRAPH SAGE (2STG)	75.13 ± 0.82	80.86 ± 1.19	76.84 ± 0.54	62.75 ± 1.20	71.29 ± 0.41	68.37 ± 0.63	4.47
GRAPH SAGE (2STG+)	76.52 ± 1.47	81.14 ± 0.68	77.71 ± 0.41	62.65 ± 0.72	72.34 ± 0.56	68.24 ± 0.83	5.01
GAT	66.50 ± 1.24	65.18 ± 1.03	76.23 ± 0.67	60.65 ± 0.42	66.92 ± 0.75	67.13 ± 0.88	-
GAT (2STG)	72.95 ± 0.91	77.84 ± 0.63	76.34 ± 0.52	62.04 ± 1.16	70.17 ± 0.72	69.15 ± 0.87	4.11
GAT (2STG+)	74.13 ± 1.47	78.17 ± 1.41	76.49 ± 1.23	61.61 ± 0.53	72.64 ± 0.58	67.25 ± 0.89	5.37
DIFFPOOL	72.11 ± 0.42	86.32 ± 0.83	77.21 ± 1.16	61.15 ± 0.35	72.24 ± 0.67	64.93 ± 0.74	-
DIFFPOOL (2STG)	74.93 ± 0.53	86.14 ± 0.77	77.94 ± 1.28	62.03 ± 0.32	73.87 ± 0.64	65.22 ± 0.83	1.03
DIFFPOOL (2STG+)	78.84 ± 0.54	87.38 ± 0.62	77.08 ± 1.23	62.15 ± 0.68	73.07 ± 1.17	64.90 ± 0.81	1.07
EIGENGCN	75.62 ± 0.63	79.87 ± 0.66	76.65 ± 1.14	63.34 ± 1.23	75.63 ± 0.82	71.86 ± 0.55	-
EIGENGCN (2STG)	77.56 ± 0.48	80.21 ± 0.71	77.98 ± 0.62	64.13 ± 0.95	75.93 ± 0.56	72.66 ± 0.42	0.91
EIGENGCN (2STG+)	78.13 ± 0.51	81.42 ± 0.86	77.02 ± 1.72	63.52 ± 1.43	77.31 ± 1.46	72.04 ± 0.53	1.07
SAGPOOL	76.12 ± 0.79	78.34 ± 0.65	76.83 ± 1.27	63.27 ± 0.78	74.34 ± 1.25	71.23 ± 1.12	-
SAGPOOL (2STG)	78.32 ± 1.26	79.63 ± 0.95	78.03 ± 0.68	63.83 ± 0.83	77.52 ± 0.54	71.73 ± 0.81	1.48
SAGPOOL (2STG+)	78.22 ± 0.70	79.03 ± 0.89	77.03 ± 0.63	64.34 ± 0.86	76.23 ± 1.12	72.36 ± 0.73	1.24

the original setting by resulting in higher intrinsic dimension, weaker correlation between dimensions, and larger distance between embeddings [Chen *et al.*, 2020]. While measuring the distance did not provide clear evidence, the results of the other two measurements validate our hypothesis.

Intrinsic dimension

We measure the *intrinsic dimension*, which we define as the dimension needed to retain 99% of the variance in the final embeddings. In the final embeddings of dimension d of the validation set, we apply Principal Component Analysis (PCA) and measure the cumulative explained variance $V(i)$ achieved by setting the final dimension as i for $i = 1, 2, \dots, d$. Assume that $V(j) \leq 0.99 \leq V(j+1)$, we linearly interpolate between j and $(j+1)$ to obtain the dimension with 99% explained variance. Such dimensions are reported in Table. 7. Given 64 dimensions as the capacity, 2STG and 2STG+ retain a higher intrinsic dimension, implying that they utilize this capacity better than the original setting.

Correlation between final embedding dimensions

We measure the average absolute correlation coefficients between final embedding dimensions. Specifically, given final embeddings of dimension five, the average of absolute values of $\binom{5}{2}$ pairwise correlation coefficients is calculated. As seen in Table. 8, in most of the cases, the original setting leads to stronger correlation than 2STG and 2STG+, and several obtained dimensions are too strongly correlated, making them redundant. This implies that 2STG and 2STG+ utilize the provided capacity (i.e., dimension of final embeddings) better than the original setting by giving fewer redundant dimen-

sions. As seen in Fig. 4, the average absolute correlation coefficients in the original setting quickly become close to 1. In 2STG and 2STG+, such correlation is much weaker.

5 Conclusion

Graph neural networks are powerful tools in dealing with the graph classification task. However, training them end-to-end to predict class probabilities often fails to realize their full capability. Thus, we apply GNN models in a triplet framework to learn discriminative embeddings first, and then train a classifier on those embeddings. Extensive experiments in 12 datasets lead to following observations:

- End-to-end training often fails to realize the full potential of GNN models. Our method consistently improves the accuracy of 5 (out of 5 tested) GNN models in 12 (out of 12 considered) datasets over the original setting.
- Despite not requiring any additional massive datasets or long training time, our two-stage method obtains better accuracy than a state-of-the-art pre-training method based on transfer-learning in 83% of the cases.
- Training each GNN in our method utilizes the given capacity better by producing embeddings with higher intrinsic dimension and weaker correlation between dimensions.

Source code

All source code used in the paper can be found at: <https://github.com/manhtuando97/two-stage-gnn>.

Table 4: Average and standard deviation (in %) of graph classification accuracies in NYC Taxi datasets in three settings: original, 2STG, and 2STG+. Similarly to the case of the benchmark datasets, 2STG and 2STG+ significantly outperform the original setting. G., Y. stand for GREEN, YELLOW. The average gain is in % points.

Method	Dataset						Average Gain (in % points)
	JAN. G.	FEB. G.	MAR. G.	JAN. Y.	FEB. Y.	MAR. Y.	
GRAPHSAGE	73.14 \pm 0.62	66.35 \pm 1.25	64.63 \pm 0.83	72.86 \pm 0.92	64.37 \pm 0.87	68.12 \pm 0.76	-
GRAPHSAGE (2STG)	76.14 \pm 0.93	66.67 \pm 1.31	67.13 \pm 0.85	75.24 \pm 1.16	65.43 \pm 0.68	70.15 \pm 0.64	1.88
GRAPHSAGE (2STG+)	76.63 \pm 0.82	67.74 \pm 0.88	68.95 \pm 1.41	75.21 \pm 1.70	67.64 \pm 0.73	70.23 \pm 1.25	2.82
GAT	71.26 \pm 1.51	67.82 \pm 0.77	66.13 \pm 0.72	72.64 \pm 0.54	64.76 \pm 0.73	67.51 \pm 1.69	-
GAT (2STG)	75.23 \pm 0.82	67.24 \pm 0.56	67.34 \pm 0.71	76.82 \pm 1.23	66.45 \pm 0.85	70.66 \pm 0.78	2.27
GAT (2STG+)	74.65 \pm 0.98	68.11 \pm 0.69	69.15 \pm 1.37	74.79 \pm 1.27	68.75 \pm 0.66	70.44 \pm 0.93	3.04
DIFFPOOL	78.43 \pm 0.74	73.12 \pm 0.42	71.39 \pm 1.56	72.52 \pm 1.23	67.43 \pm 0.87	74.34 \pm 0.77	-
DIFFPOOL (2STG)	80.28 \pm 1.16	75.69 \pm 1.21	73.79 \pm 0.81	75.09 \pm 0.72	68.19 \pm 0.50	74.87 \pm 0.83	1.78
DIFFPOOL (2STG+)	79.63 \pm 0.82	74.56 \pm 1.32	72.92 \pm 0.65	75.95 \pm 1.21	69.31 \pm 0.97	75.76 \pm 0.86	1.81
EIGENGCN	75.45 \pm 0.44	69.32 \pm 1.82	72.21 \pm 0.83	73.21 \pm 1.35	69.64 \pm 0.76	69.52 \pm 1.54	-
EIGENGCN (2STG)	77.14 \pm 0.81	70.03 \pm 0.62	74.12 \pm 1.34	74.36 \pm 1.65	69.72 \pm 0.97	70.03 \pm 0.86	1.02
EIGENGCN (2STG+)	76.73 \pm 1.21	71.27 \pm 1.33	73.37 \pm 1.85	75.33 \pm 1.14	71.65 \pm 1.67	71.84 \pm 0.62	1.80
SAGPOOL	73.23 \pm 0.59	67.46 \pm 0.73	72.78 \pm 1.34	72.65 \pm 0.72	68.83 \pm 1.25	69.68 \pm 1.35	-
SAGPOOL (2STG)	76.36 \pm 1.37	69.07 \pm 1.48	74.34 \pm 1.52	71.11 \pm 0.73	70.02 \pm 0.64	70.04 \pm 1.48	1.05
SAGPOOL (2STG+)	75.38 \pm 0.86	69.27 \pm 1.12	73.19 \pm 1.34	72.51 \pm 0.85	69.16 \pm 0.79	70.59 \pm 0.52	0.91

Table 5: 2STG+ outperforms transfer learning [Hu *et al.*, 2020] in terms of classification accuracy in most cases when the benchmark datasets are used. Note that 2STG+ has several advantages over [Hu *et al.*, 2020], as highlighted in Table 1.

Method	Dataset						Average Gain (in % points)
	DD	MUTAG	MUTAGENICITY	PTC-FM	PROTEINS	IMDB-B	
GAT (2STG+)	74.13 \pm 1.47	78.17 \pm 1.41	76.49 \pm 1.23	61.61 \pm 0.53	72.64 \pm 0.58	67.25 \pm 0.89	1.56
GAT [Hu <i>et al.</i> , 2020]	72.24 \pm 0.83	76.86 \pm 1.35	75.59 \pm 1.48	61.28 \pm 0.97	71.76 \pm 0.77	65.16 \pm 1.47	-
GRAPHSAGE (2STG+)	76.52 \pm 1.47	81.14 \pm 0.68	77.71 \pm 0.41	62.65 \pm 0.72	72.34 \pm 0.56	68.24 \pm 0.83	0.31
GRAPHSAGE [Hu <i>et al.</i> , 2020]	75.26 \pm 1.36	82.43 \pm 1.49	76.83 \pm 0.95	62.61 \pm 0.78	72.15 \pm 0.83	67.14 \pm 0.52	-

Table 6: 2STG+ leads to higher classification accuracy than transfer learning [Hu *et al.*, 2020] in most NYC Taxi datasets. Compared with Table 5, the gap is larger as the GNNs for transfer learning were pre-trained on either a biology or chemistry dataset, which is of a far domain.

Method	Dataset						Average Gain (in % points)
	JAN.G.	FEB. G.	MAR. G.	JAN. Y.	FEB. Y.	MAR. Y.	
GAT (2STG+)	74.65 \pm 0.98	68.11 \pm 0.69	69.15 \pm 1.37	74.79 \pm 1.27	68.75 \pm 0.66	70.44 \pm 0.93	1.97
GAT [Hu <i>et al.</i> , 2020]	73.87 \pm 1.13	65.89 \pm 1.04	67.25 \pm 1.27	71.87 \pm 1.35	66.24 \pm 0.92	68.95 \pm 1.25	-
GRAPHSAGE (2STG+)	76.63 \pm 0.82	67.74 \pm 0.88	68.95 \pm 1.41	75.21 \pm 1.70	67.64 \pm 0.73	70.23 \pm 1.25	0.53
GRAPHSAGE [Hu <i>et al.</i> , 2020]	75.19 \pm 0.98	67.82 \pm 0.43	68.03 \pm 0.66	73.66 \pm 1.27	68.24 \pm 0.79	70.25 \pm 0.73	-

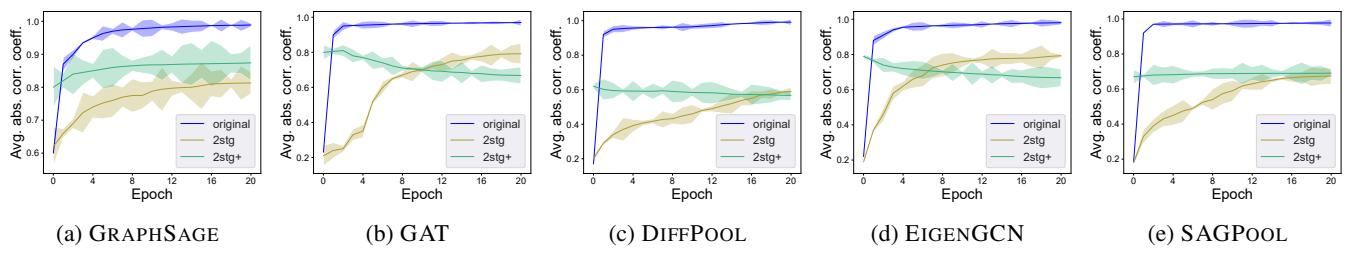


Figure 4: Average absolute correlation coefficients over epochs on the Jan. G. dataset. Means and standard deviations of 15 runs are plotted. Embedding dimensions quickly become strongly correlated in the orginal setting. Such correlation is much weaker in 2STG and 2STG+.

Table 7: Intrinsic dimension obtained by PCA on embeddings of dimension 64. Each entry is the mean of 5 runs. In most cases, the GNNs trained in 2STG and 2STG+ lead to higher intrinsic dimensions than that trained in the original setting.

Method	Dataset												
	DD	MUTAG	MUTAGENICITY	PTC-FM	PROTEINS	IMDB-B	JAN. G.	FEB. G.	MAR. G.	JAN. Y.	FEB. Y.	MAR. Y.	
GRAPH SAGE	37.01	8.73	10.23	8.76	33.67	24.41	26.46	30.49	32.21	31.02	21.24	23.41	
GRAPH SAGE (2STG)	38.83	9.82	17.21	19.98	36.13	24.18	35.98	36.03	36.98	38.95	38.61	36.64	
GRAPH SAGE (2STG+)	18.33	9.34	30.27	16.65	18.11	14.31	37.56	33.37	29.88	24.06	23.51	32.07	
GAT	8.38	6.59	12.26	11.07	33.72	26.46	35.17	34.44	33.61	32.01	36.95	31.72	
GAT (2STG)	34.53	10.78	20.47	18.73	31.13	15.33	23.57	25.24	38.01	39.02	24.34	23.33	
GAT (2STG+)	31.17	9.19	18.54	16.94	33.88	14.54	36.89	38.27	26.98	35.18	23.57	28.42	
DIFFPOOL	6.69	2.48	15.97	13.17	14.33	14.31	4.64	16.84	6.62	16.56	16.19	15.71	
DIFFPOOL (2STG)	37.16	8.04	22.64	20.02	18.25	19.39	33.03	32.77	26.78	36.01	33.09	33.93	
DIFFPOOL (2STG+)	15.11	8.23	25.19	7.02	10.21	11.21	32.57	33.14	26.22	30.93	20.84	30.35	
EIGENGCN	14.76	4.33	20.49	12.57	17.94	17.21	3.98	15.59	3.19	15.77	28.02	23.47	
EIGENGCN (2STG)	25.63	8.85	31.53	19.77	19.26	17.32	31.21	32.45	27.68	32.81	26.55	26.92	
EIGENGCN (2STG+)	22.76	5.53	28.75	19.62	19.57	11.74	32.26	33.98	18.87	30.87	31.01	26.47	
SAGPOOL	15.86	2.92	17.88	12.99	11.82	17.31	5.42	17.66	3.75	14.99	25.09	20.01	
SAGPOOL (2STG)	29.46	9.56	18.25	15.67	16.83	18.49	31.81	28.57	31.19	32.71	28.19	31.11	
SAGPOOL (2STG+)	25.83	7.42	18.37	14.74	15.87	11.75	33.18	32.49	28.16	21.31	27.56	29.11	

Table 8: Average absolute correlation coefficients between the five dimensions of the final embeddings. Each entry is the mean of 5 runs, and it ranges from 0 to 1. Higher values indicate stronger correlation between the dimensions. In most cases, the original setting leads to stronger correlation between the dimensions than 2STG and 2STG+.

Method	Dataset												
	DD	MUTAG	MUTAGENICITY	PTC-FM	PROTEINS	IMDB-B	JAN. G.	FEB. G.	MAR. G.	JAN. Y.	FEB. Y.	MAR. Y.	
GRAPH SAGE	0.94	0.77	0.91	0.56	0.96	0.55	0.92	0.87	0.95	0.89	0.89	0.94	
GRAPH SAGE (2STG)	0.67	0.66	0.26	0.28	0.43	0.31	0.41	0.66	0.66	0.68	0.61	0.84	
GRAPH SAGE (2STG+)	0.62	0.59	0.42	0.45	0.41	0.32	0.45	0.76	0.66	0.62	0.54	0.49	
GAT	0.92	0.98	0.87	0.62	0.82	0.84	0.71	0.77	0.91	0.86	0.78	0.86	
GAT (2STG)	0.72	0.45	0.25	0.26	0.46	0.53	0.54	0.54	0.61	0.41	0.52	0.52	
GAT (2STG+)	0.64	0.56	0.29	0.37	0.41	0.47	0.65	0.84	0.58	0.48	0.58	0.46	
DIFFPOOL	0.81	0.85	0.75	0.49	0.91	0.87	0.95	0.88	0.97	0.98	0.59	0.97	
DIFFPOOL (2STG)	0.36	0.52	0.51	0.35	0.38	0.39	0.66	0.51	0.61	0.76	0.83	0.88	
DIFFPOOL (2STG+)	0.32	0.44	0.47	0.31	0.42	0.33	0.57	0.92	0.59	0.44	0.39	0.53	
EIGENGCN	0.72	0.92	0.71	0.58	0.84	0.78	0.97	0.97	0.92	0.95	0.94	0.98	
EIGENGCN (2STG)	0.58	0.51	0.47	0.45	0.57	0.34	0.68	0.78	0.56	0.44	0.53	0.84	
EIGENGCN (2STG+)	0.54	0.75	0.42	0.37	0.43	0.32	0.45	0.71	0.51	0.44	0.48	0.63	
SAGPOOL	0.75	0.98	0.62	0.47	0.82	0.82	0.99	0.94	0.88	0.73	0.93	0.92	
SAGPOOL (2STG)	0.49	0.61	0.38	0.26	0.34	0.44	0.31	0.42	0.51	0.56	0.62	0.49	
SAGPOOL (2STG+)	0.41	0.55	0.36	0.31	0.32	0.41	0.38	0.41	0.57	0.92	0.57	0.53	

References

- [Chen *et al.*, 2020] Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, and Xu Sun. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020.
- [Ching *et al.*, 2018] Travers Ching, Daniel S Himmelstein, Brett K Beaulieu-Jones, Alexandr A Kalinin, Brian T Do, Gregory P Way, Enrico Ferrero, Paul-Michael Agapow, Michael Zietz, Michael M Hoffman, et al. Opportunities and obstacles for deep learning in biology and medicine. *Journal of The Royal Society Interface*, 15(141):20170387, 2018.
- [Dai *et al.*, 2016] Hanjun Dai, Bo Dai, and Le Song. Discriminative embeddings of latent variable models for structured data. In *Proceedings of the International Conference on Machine Learning*, 2016.
- [Defferrard *et al.*, 2016] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*, pages 3844–3852, 2016.
- [Duvenaud *et al.*, 2015] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in Neural Information Processing Systems*, pages 2224–2232, 2015.
- [Gao and Ji, 2019] Hongyang Gao and Shuiwang Ji. Graph u-nets. In *Proceedings of the International Conference on Machine Learning*, 2019.
- [Hamilton *et al.*, 2017] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pages 1024–1034, 2017.
- [Hu *et al.*, 2020] Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. Strategies for pre-training graph neural networks. In *Proceedings of the International Conference on Learning Representations*, 2020.
- [Kipf and Welling, 2017] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *Proceedings of the International Conference on Learning Representations*, 2017.
- [Ktena *et al.*, 2018] Sofia Ira Ktena, Sarah Parisot, Enzo Ferante, Martin Rajchl, Matthew Lee, Ben Glocker, and Daniel Rueckert. Metric learning with spectral graph con-

- volutions on brain connectivity networks. *NeuroImage*, 169:431–442, 2018.
- [Lee *et al.*, 2019] Junhyun Lee, Inyeop Lee, and Jaewoo Kang. Self-attention graph pooling. In *Proceedings of the International Conference on Machine Learning*, 2019.
- [Ling *et al.*, 2020] Xiang Ling, Lingfei Wu, Saizhuo Wang, Tengfei Ma, Fangli Xu, Alex X Liu, Chunming Wu, and Shouling Ji. Hierarchical graph matching networks for deep graph similarity learning. *arXiv preprint arXiv:2007.04395*, 2020.
- [Liu *et al.*, 2019] Jiahao Liu, Guixiang Ma, Fei Jiang, Chun-Ta Lu, S Yu Philip, and Ann B Ragin. Community-preserving graph convolutions for structural and functional joint embedding of brain networks. In *Proceedings of the IEEE International Conference on Big Data*, 2019.
- [Ma *et al.*, 2019] Yao Ma, Suhang Wang, Charu C. Aggarwal, and Jiliang Tang. Graph convolutional networks with eigenpooling. In *Proceedings of the International Conference on Knowledge Discovery & Data Mining*, 2019.
- [Morris *et al.*, 2020] Christopher Morris, Nils M Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. Tudataset: A collection of benchmark datasets for learning with graphs. *arXiv preprint arXiv:2007.08663*, 2020.
- [Niepert *et al.*, 2016] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In *Proceedings of the International Conference on Machine Learning*, 2016.
- [Schlichtkrull *et al.*, 2018] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *Proceedings of the European Semantic Web Conference*, 2018.
- [Schroff *et al.*, 2015] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015.
- [Veličković *et al.*, 2018] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *Proceedings of the International Conference on Learning Representations*, 2018.
- [Wang *et al.*, 2019] Jingshu Wang, Divyansh Agarwal, Mo Huang, Gang Hu, Zilu Zhou, Chengzhong Ye, and Nancy R Zhang. Data denoising with transfer learning in single-cell transcriptomics. *Nature methods*, 16(9):875–878, 2019.
- [Ying *et al.*, 2018] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In *Advances in Neural Information Processing Systems*, pages 4800–4810. 2018.
- [Zhang *et al.*, 2018] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018.

A Datasets

We tested our training method using 12 datasets. Some statistics of the datasets are summarized in Table. 2

A.1 Benchmark datasets

These are the commonly tested binary-class datasets [Morris *et al.*, 2020] for the graph classification task: DD, MUTAG, MUTAGENICITY, PTC-FM, PROTEINS, and IMDB-B.

A.2 New York City Taxi datasets

We extracted the taxi ridership data in 2019 from New York City (NYC) Taxi Commission. The areas in New York are represented as nodes, and each taxi trip is an edge connecting the source and destination nodes. All taxi trips in an 1-hour interval form a graph, and each dataset spans a month of taxi operations. We augmented the binary label for each graph as taxi trips in weekdays (Mon-Thu) vs. weekend (Fri-Sun). We considered two taxi operators (Yellow and Green) and processed data in January, February and March of 2019, making 6 datasets in total: JAN. G., FEB. G., MAR. G., JAN. Y., FEB. Y., and MAR. Y..

B GNN architectures

In order to demonstrate that our two-stage training method helps realize a better performance of GNNs, for each GNN architecture, we compared the accuracy obtained in the original setting versus that from our method. The GNN architectures we considered in this work are:

- **GRAPH SAGE** [Hamilton *et al.*, 2017]: This is often used as a strong baseline in graph classification. After obtaining node embeddings, global mean/max pooling is applied to combine all node embeddings into one graph embedding.
- **GAT** [Veličković *et al.*, 2018]: Instead of uniformly passing neighbor information into a node embedding, [Veličković *et al.*, 2018] employs an attention mechanism for the importance of each neighboring node.
- **DIFFPOOL** [Ying *et al.*, 2018]: While using the same aggregation mechanism as [Hamilton *et al.*, 2017], [Ying *et al.*, 2018] proposes a hierarchical approach to pool the node embeddings. Rather than a “flat-pooling” step at the end, diff-pool repeatedly merges nodes into “supernodes” until there is only one supernode whose embedding is treated as the graph embedding.
- **EIGENGCN** [Ma *et al.*, 2019]: Attempting to implement hierarchical pooling like [Ying *et al.*, 2018], [Ma *et al.*, 2019] formulates a different way to combine nodes and their respective embeddings making use of the eigenvectors of the Laplacian matrix.
- **SAGPOOL** [Lee *et al.*, 2019]: Hierarchical graph pooling employing self-attention mechanisms.

C Hyperparameter search

For each GNN, the hyperparameters regarding the network architecture were tuned in the same search space for the three settings: original, 2STG, 2STG+. The search space for the dimensions of the input vector, hidden vector and output vector for all GNNs was $\{16, 32, 64, 96, 128\}$. For DIFFPOOL, we used three layers of graph convolution and one DIFFPOOL layer as described in the original paper. For EIGENGCN, we used three pooling operators as it was shown to achieve the best performance in the original paper. For SAGPOOL, we used three pooling layers as explained in the original paper. Other hyperparameters that are exclusive to each GNN architecture were set to the default values provided in each paper’s original code of each architecture’s authors.

In all three settings, the architecture of the final classifier was also tuned. The number of fully-connected layers was up to 3 while the search space for the hidden dimension was $\{2^h | 1 \leq h \leq \log_2(d)\}$, where d is the dimension of the output vector.

The two settings 2STG and 2STG+ require an additional hyperparameter α as the margin in the triplet loss. While [Schroff *et al.*, 2015] found $\alpha = 0.2$ to be effective, we empirically found that this value was too small to separate instances of different classes. Instead, the search space for α we used was $\{0.5, 1.0, 1.5, 2.0, 2.5\}$.

D Computing specifications & dependencies

All experiments were performed on Ubuntu 18.04 LTS running on a machine with four RTX2080Ti GPUs, each of which has 11GB memory.

The following libraries were used to run the code:

- torch 1.7.0,
- networkx 2.5,
- sklearn 0.23.2,
- numpy 1.16.4,
- torch-geometric 1.6.3.