ECAI 2023
K. Gal et al. (Eds.)

doi:10.3233/FAIA230624

Structure-Aware Group Discrimination with Adaptive-View Graph Encoder: A Fast Graph Contrastive Learning Framework

Zhenshuo Zhang[a], Yun Zhu[a], Haizhou Shi[b] and Siliang Tang[a;*]

[a]Zhejiang University
[b]Rutgers University

Abstract. lbeit having gained significant progress lately, large-scale graph representation learning remains expensive to train and deploy for two main reasons: (i) the repetitive computation of multi-hop message passing and non-linearity in graph neural networks (GNNs); (ii) the computational cost of complex pairwise contrastive learning loss. Two main contributions are made in this paper targeting this twofold challenge: we first propose an adaptive-view graph neural encoder (AVGE) with a limited number of message passing to accelerate the forward pass computation, and then we propose a structure-aware group discrimination (SAGD) loss in our framework which avoids inefficient pairwise loss computing in most common GCL and improves the performance of the simple group discrimination. By the framework proposed, we manage to bring down the training and inference cost on various large-scale datasets by a significant margin (250x faster inference time) without loss of the downstream-task performance.

1 Introduction

Graph Neural Networks (GNNs) have shown superiority in dealing with graph-structured data, such as social networks [5], traffic networks [4], and molecular graphs [26]. In real-world scenarios, however, large-scale graph data often lack human-annotated labels, which creates a huge barrier for the traditional supervised learning paradigm. To conquer this limitation, self-supervised graph representation learning methods have been widely studied, among which Graph Contrastive Learning (GCL) is dominant due to its ability to learn robust and generalizable representations for the downstream tasks [24, 8, 19]. In GCL, the graph data encoder is trained to produce the representation space that minimizes the distance between the semantically invariant perturbed instances, e.g., sub-graphs created with mild augmentation, and maximizes the distance between irrelevant instances, e.g., randomly sampled sub-graphs.

Although proven to be effective, the existing GCL methods have limitations in real-world large-scale graph data applications: since they typically require large amounts of time and computational resources to deploy. For one thing, the most common GNN encoders utilize multi-hop information in graphs by multi-layer message passing in every calculation step, which leads to large computational costs for both training and inference. And for another, the predominant pairwise constrictive loss is not efficient enough and takes lots of time until convergence. In the supervised setting, there are several works addressing the first problem by reducing the number of parameterized message passing [25] or distilling the trained GNN to Multi-Layer Perceptron (MLP) to improve inference speed [21]. As for the second problem, various techniques have been studied such as simplifying positive and negative sample construction process [13], and removing the negative sample generation process [19, 7, 18]. However, those works didn't explore the application of a more efficient encoder in GCL and discuss the relationship between encoder and pretext tasks.

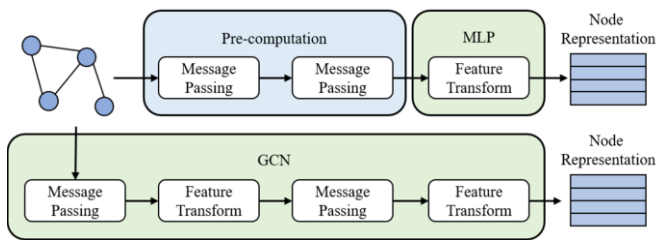

Figure 1: The architecture of separating 2-hop message passing and feature transform (above), compare to 2-layer GCN architecture (below). GCN will degrade to MLP if feature message passing is predcomputed and removed in each GCN layer.

In this paper, we propose a novel GCL framework (AVGE-SAGD) to tackle the aforementioned two challenges. It contains an adaptive-view graph encoder (AVGE) that achieves higher training and inference speed than the GNN counterparts, and a structure-aware group discrimination (SAGD) module that increases the speed of the pretext contrastive task training. In the AVGE, instead of using GNN, we first perform a limited number of message-passing to generate a multi-view feature vector that consists of multi-hop features. During training, the multi-view features are adaptively input to the encoder. And then we use an MLP encoder to further learn high-level representations for the pretext task. This encoder is significantly more efficient since it separates message passing and feature encoding and strictly controls the number of both operations. In the SAGD module, we first introduce the group discrimination loss to avoid inefficient pairwise contrastive loss computation [32]. Considering that the AVGE views the input multi-hop vector as a collection of independent features, it will lose the structural information of the original graph. Therefore to empower AVGE in the scenario of self-supervised representation

* Corresponding Author. Email: siliang@zju.edu.cn

Please check ArXiv or contact the authors for any appendices or supplementary material mentioned in the paper.

learning, a novel structure prediction loss is added. It requires the encoded feature to further divide the graph into meaningful groups. By this extra topological constraint, we manage to prevent performance degradation and even achieve performance improvement.

Overall, our framework offers a simple and efficient approach to graph self-supervised learning by incorporating an adaptive-weight encoder. We design novel components for accelerating and preserving structure information and combine them in a non-trivial way.

By reducing computation time while maintaining optimal performance (e.g., 300x inference speed up in OGBN-Products), our framework provides a practical solution to graph self-supervised learning algorithms' computational challenges in real-world scenarios. To summarize, our contributions are as follows:

- We propose a graph encoder that adaptively utilizes multi-hop neighbor information, and separates the message passing from the encoder calculation procedure to save the repeated message passing calculation steps in the traditional GNN encoders, thereby improving the training speed and inference speed of our framework.
- We propose a novel structure-aware group discrimination (SAGD) module for GCL. It is built on graph group discrimination and further requires the encoder to subdivide the group into topology-based mini-groups so that the pre-trained model preserves more structural information and achieves better generalization ability for downstream tasks.
- Experiments on various node classification datasets showcase the effectiveness of our framework in terms of training and inference efficiency and downstream-task performance. Especially on large-scale graph data, our method achieves comparable performance with less training time and 250x faster inference time.

2 Related Work

Our framework involves two aspects: GNN encoder architecture and GCL methods. In this chapter, we introduce several previous works, discuss their limitations in GCL and propose our ideas for improvement.

2.1 GNN Architecture

Architecture design is an important part of GNN research. The most mainstream GNN structures are designed based on message passing, the most widely known of which is GCN [11]. There are several works that tried to accelerate the computing speed of GNN by separating the message passing phase and feature calculation as shown in Figure 1. The most known architecture is SGC [25], which removes the non-linearity calculation between GCN layers and simplified it to linear transform, showing that parameters-free linear message passing can achieve similar performance to GCNs. NAFS [31] present learning-free node-adaptive feature smoothing, assign fixed weights for features in different hops by computing the distance from the aggregated features to the extreme over-smoothed features, and combine the features in different hops by summation.

Some studies have shown that under a certain design, the joint use of different hop features can enhance expressive ability. ASGC [2] uses the linear regression method to fit the raw features by constructing a linear combination of different hop features, thereby solving the problem of heterophily graph node classification. GCN-PND [9] updates graph topology based on the similarity between the local neighborhood distribution of nodes and designing extensible aggregation from multi-hop neighbors.

These methods of separating message passing and feature computation are all applied in supervised scenarios, and a combination method such as summation is used for multi-hop information to keep data scale. We explore the separation of message passing and feature computation GNN architectures in self-supervised scenarios.

2.2 Self-Supervised Graph Representation Learning

Self-supervised learning was first proposed in the computer vision area and has quickly received widespread attention in the community of graph learning due to its excellent performance in scenarios with few labeled training data. There are three levels of contrastive learning in the GCL field: graph-graph level [29], node-graph level [24, 8], and node-node level [14, 34, 36]. Among those methods, we focus on the node-graph level since our framework is also a kind of node-graph level contrastive learning. DGI [24] obtains the graph-level representation by applying a readout function on the graph and maximizes the mutual information between the patch and the graph representation to perform node-graph level graph contrastive learning. MVGRL [8] uses multi-view constructiveness to extend the idea of DGI and borrow the idea from graph diffusion networks [12] to improve the performance. The training loss of DGI can be simplified into a binary classification loss which is empirically and theoretically proven in [32]. The training scheme in [32] is coined as *Group Discrimination* which can implement efficient training but neglect the inner group relations which can be used to divide the original group into multiple mini-groups. In order to overcome these obstacles, we design SAGD by dividing it into mini-groups according to the structure, which is more helpful to our encoder.

3 Method

In this chapter, we will introduce our framework AVGE-SAGD in detail. The overall processes are shown in Figure 2.

3.1 Problem Formulation

Given a graph $\mathcal{G} = (\mathbf{X}, \mathbf{A})$ with node attribute matrix $\mathbf{X} \in \mathbf{R}^{N\times d}$, where N is the number of nodes, d is node attribute dimension, and graph adjacency matrix $\mathbf{A} \in \mathbb{R}^{N\times N}$, where $A_{i,j} = 1$ if node i and j are connected, else $A_{i,j} = 0$. For message passing, we follow the setting in GCN, using normalized adjacency matrix $\tilde{\mathbf{A}} = \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$ where $\mathbf{D}$ is the diagonal degree matrix and $D_{ii} = d_i$ represent the number of degrees of node i. In our framework, we define the encoder as $f_\theta : \mathbb{R}^{N\times d} \to \mathbb{R}^{N\times d'}$ where d' is the dimension of node representations.

The goal of GCL is to train a generalized graph encoder f_θ by a pretext loss $\mathcal{L}$ without labels. For evaluating the pre-trained model on a specific downstream task (*e.g.*, node classification), we will obtain the node representations by the frozen encoder ($\mathbf{H} = f_\theta(\mathbf{A}, \mathbf{X})$). Then, we will train a linear classifier built on these node representations from the training set by a supervised loss (*e.g.*, cross-entropy loss). Lastly, we will use the test set to evaluate the performance of our pre-trained model with the linear classifier.

3.2 Generating Positive and Negative Samples

3.2.1 Data augmentation

We adopt data augmentation in generating positive samples. Node attribute masking is a popular technique and is widely used in GCL

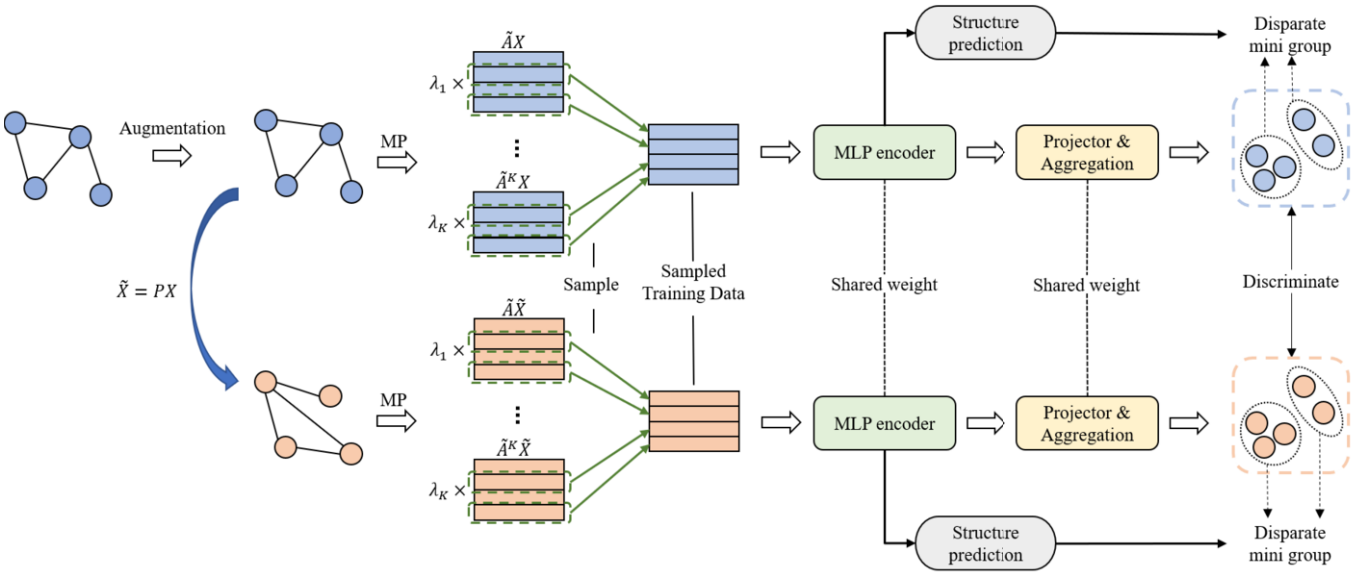

Figure 2: The architecture of AVGE-SAGD. Given a graph $\mathcal{G}$ and node attribute matrix $\mathbf{X}$, we first adopt an optional data augmentation and then generate negative samples by randomly permutating the node attributes matrix. Message passing is processed for both the positive sample and negative sample which will give K-views of features in K-hop. We sample N/K features in each view to keep the scale of training data. The training data will be fed to the MLP encoder. After projection and aggregation, the generated embeddings can be discriminated into the positive group and negative group.

methods (*e.g.*, GraphCL [29], GGD [32]). We adopt this data augmentation technique to enrich the features of positive samples. In practice, partial dimensions of node attributes will be masked with 0.

The augmented node attributes $\tilde{\mathbf{X}}$ is obtained by:

$$\tilde{\mathbf{X}} = \mathbf{X} \circ \mathbf{M}, \tag{1}$$

where $\mathbf{M} \in \mathbb{R}^{N \times D}$ is masking matrix and each row vector in $\mathbf{M}$ are equal (*i.e.*, $\mathbf{m}_i = \mathbf{m}_j, \forall i, j$), each element m_{ij} in $\mathbf{m}_i \in \{0, 1\}^D$ is is drawn from a Bernoulli distribution with probability p_m (*i.e.*, $m_{ij} \sim \mathcal{B}(1 - p_m)$). In order to keep the notation uncluttered, we use $\mathbf{X}$ to represent the augmented feature matrix in later sections.

3.2.2 Corruption

We adopt corruption to generate negative samples. We randomly permutate the node attributes matrix and keep the topology structure unchanged:

$$\check{g} = \{\check{\mathbf{X}}, \mathbf{A}\}, \check{\mathbf{X}} = \mathbf{P}\mathbf{X}, \tag{2}$$

where $\mathbf{P}$ is a permutation matrix.

This corruption technique

is widely used in node-graph level GCL frameworks (*e.g.*, DGI [24], MVGRL [8]) to encourage the representations including structural similarities of different nodes in the graph properly. In our framework, this corruption operation will mislead message passing (*e.g.*, $\mathbf{A}\check{\mathbf{X}}$) to generate erroneous node attributes as negative samples.

3.3 Adaptive-View Graph Encoder

3.3.1 Post-message-passing features as training data

With data augmentation and corruption, we can obtain multi-view features by parameters-free linear message passing: $[\tilde{\mathbf{A}}\mathbf{X}; \tilde{\mathbf{A}}^2\mathbf{X}; ...; \tilde{\mathbf{A}}^K\mathbf{X}]$ and $[\tilde{\mathbf{A}}\check{\mathbf{X}}; \tilde{\mathbf{A}}^2\check{\mathbf{X}}; ...; \tilde{\mathbf{A}}^K\check{\mathbf{X}}]$ which will be used to train the encoder f_θ. These features can be reused during training and inference which can save a lot of computational time.

Considering that we store K views of attributes for each node by parameter-free linear message passing, the scale of training data of a graph with N nodes increases from N to KN compared to standard GNN encoders, which is certainly contradictory to the goal of reducing computing time and memory. We use a simple sample method to make a trade-off between performance and computation cost. In each epoch, we randomly sample $\frac{N}{K}$ nodes to keep the input training data size as N, which is consistent with the standard GNN encoder training process.

Another alternative approach is to use the average or summation of features in different hops. Unfortunately, the information of different hops will be mixed up which leads to performance degeneration. However, our sample method can explicitly use features in more views that provide more distinct and useful information.

3.3.2 Adaptive weighted training

Different hop features will be fed to train the encoder, but some hops' information is redundant and high-order hop features may incur over-smoothing [3]. So, the contributions of each hop's feature should be disparate.

We assign individual adversarially learnable weight λ_i to each hop feature $\tilde{\mathbf{A}}^i\mathbf{X}$. The training data can be reformulated as $[\lambda_1\tilde{\mathbf{A}}\mathbf{X}, \lambda_2\tilde{\mathbf{A}}^2\mathbf{X}, ..., \lambda_K\tilde{\mathbf{A}}^K\mathbf{X}]$.

In order to avoid the training loss easily converging to 0 (*i.e.*, through minimizing training loss, discrepant high-order features will have large weights λ_i and weights of indiscernible low-order will easily collapse to zero), we use a two-step min-max optimization method to train the MLP encoder with adaptive weighted multiple receptive field features. This training method can be formulated as:

Algorithm 1 Adaptive weighted training algorithm

Input: initial model parameter $\theta^{(0)}$, adaptive weight $\lambda^{(0)}$, total training epoch E
Parameter: θ, λ
Output: Optimized model parameter $\theta^{(N)}$

1: **for** $e = 1$ to E **do**
2: Maximization: fix $\theta = \theta^{(e-1)}$ and calculate the gradient of $\lambda^{(e)}$
3: Minimization: fix $\lambda = \lambda^{(e)}$ and calculate the gradient of $\theta^{(e)}$
4: update θ and λ
5: **end for**
6: **return** Optimized parameter $\theta^{(E)}$

$$\min_{\theta} \max_{\lambda} \quad \mathcal{L}(\lambda_i \tilde{\mathbf{A}}^i \mathbf{X}, \lambda_i \tilde{\mathbf{A}}^i \check{\mathbf{X}}, \theta) \tag{3}$$

$$\text{s.t.} \quad \sum_{i}^{K} \lambda_i = 1, \forall \lambda_i \in [0, 1], \tag{4}$$

where λ is Xavier initialised [6] and $\mathcal{L}$ is our training loss which will be described in Section 3.4. At each training step, firstly, we optimize adaptive weights with frozen model parameters by maximizing the training loss. Then, we optimize the parameters of the encoder with fixed adaptive weights by minimizing training loss. The optimization algorithm is described in Algorithm 1.

From a more theoretical aspect, our motivation is given from the analysis of research on homophilous graphs and heterophilous graphs [33]. Homophily describes the similarity between adjacent nodes. The relevant studies [27] show that graph representation learning will benefit from message passing in a homophilous graph and the opposite in a heterophilous graph. The corruption operation disrupts the graph connection relationship, which will make the corrupted graph turn into a heterophilous graph. As the order of message passing hop increases, the node attributes in the positive group and negative group will be separated spontaneously. So the model without an adaptive weighted training method will take shortcuts by overly using high-order-hop features during pre-training, which will consequently cause the model to lose generalization ability.

3.4 Structure-Aware Group Discrimination

3.4.1 Group discrimination

It has been empirically proven that the contrastive learning task in DGI can be transformed into a binary classification task named group discrimination [32]. Following this method, we use a projector $g_\omega(\cdot)$ which consists of an MLP to map node representations into another latent space and then aggregate the projected representations. At last, we use binary cross entropy (BCE) loss to discriminate them into positive and negative groups, which are labeled as $y_i = 1$ and $y_i = 0$ respectively. The group discrimination loss can be formulated as:

$$\mathcal{L}_{\text{GD}} = -\frac{1}{2N} \sum_{i=1}^{2N} \left[y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i) \right], \tag{5}$$

where $\hat{y}_i = agg(g_\omega(\mathbf{h}_i))$, $agg(\cdot)$ is summation aggregation.

3.4.2 Preserving structure

Simple MLP encoder cannot preserve structure information [21] because the training data are all independent node attributes in disparate hops. To solve this problem, we design auxiliary classification tasks to preserve structural information and capture the inner group relations so that the discriminated groups will be implicitly divided into mini-groups. Figure 3 shows the procedure of SAGD. Considering that our goal is to speed up computation, we prefer simple and efficient modules to assist the encoder. The losses of these two modules are simple and consistent with the formula of group discrimination, which can make the convergence more stable.

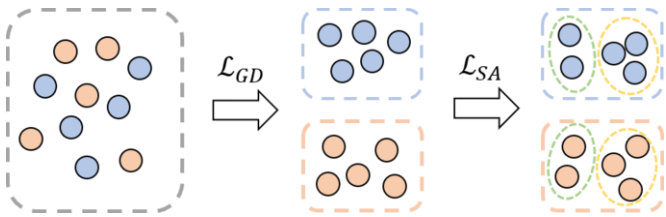

Figure 3: The schematic diagram of SAGD. $\mathcal{L}_{\text{GD}}$ means group discrimination loss and $\mathcal{L}_{\text{SA}}$ means structure aware loss. On the basis of $\mathcal{L}_{\text{GD}}$ distinguishing positive and negative samples, $\mathcal{L}_{\text{SA}}$ further distinguishes the mini-group according to the structure.

Here we introduce the concept of relative degree which evaluates the node degree compared to its neighbors' degrees. The definition of the relative degree of node v_i is:

$$\bar{r}_i = \frac{1}{d_i} \sum_{j \in \mathcal{N}_i} \sqrt{\frac{d_i}{d_j}}. \tag{6}$$

According to [27], the nodes with a high relative degree are more sensitive to homophily and heterophily. Positive sample nodes can be considered as having high homophily, while negative sample nodes can be considered as having low homophily. Nodes with high relative degrees will have features closer to the center of their respective classes after undergoing homophilous message passing, while they will be farther away from the center of their respective classes after undergoing heterophilous message passing. For an ideal and simple example, consider a graph with two classes of nodes, features of which class are μ and σ separately. The node representations of the first layer are given by $\mathbf{h}_i^{(1)} = \sum_{j \in \mathcal{N}_i} \frac{\mathbf{h}_j^{(0)}}{\sqrt{d_i}\sqrt{d_j}}$. In the extreme homophilous case, the neighbors of node i in class 1 are all in class 1, so after 1-layer message passing, $\mathbf{h}_i^{(1)} = \sum_{j \in \mathcal{N}_i} \sqrt{\frac{d_i}{d_j}} \mu$. After an ideal corruption, half of the neighbors of node i are in class 1, and the remains are in class 2. This corrupted message passing gives $\mathbf{h}_i^{(1)'} = \sum_{j \in \mathcal{N}_i} \sqrt{\frac{d_i}{d_j}} \frac{\mu+\sigma}{2}$. So $\|\mathbf{h}_i^{(1)} - \mathbf{h}_i^{(1)'}\| \propto \sqrt{\frac{d_i}{d_j}}$, which derives the idea.

In our case, the positive (high homophily) and negative (high homophily) samples generated by nodes with a high relative degree are highly discrepant. That is, the relative degree is a qualified graph structure indicator.

So encoders that can distinguish $\bar{r}$ preserve structural information and will therefore have stronger expressive power.

We hope that the formulation of the structure-preserving task can hold a consistent format of the discrimination task, which will be beneficial for model optimization.

Considering that relative degree is a continuous variable, we set 1 as the threshold to discriminate whether a node has a high relative degree, which means The relative degree loss can be written as:

$$\mathcal{L}_{\bar{r}_i} = -\frac{1}{2N}\sum_{i=1}^{2N}\left[y_{\bar{r}_i}\log(\hat{y}_{\bar{r}_i}) + (1-y_{\bar{r}_i})\log(1-\hat{y}_{\bar{r}_i})\right], \quad (7)$$

where $f_{\bar{r}} : \mathbb{R}^{D'} \to \mathbb{R}$ is a summation aggregation function and $\hat{y}_{\bar{r}}$ is the prediction result.

Furthermore, we also use the hop order as the structural information that needs to be preserved. Similar to relative degree, we conduct a classification task to predict the order number of hop it belongs to through input features. The hop loss can be written as:

$$\begin{aligned}\mathcal{L}_{\text{hop}} = -\frac{1}{2N}\sum_{i=1}^{2N}\big[&y_{\text{hop}_i}\log(\hat{y}_{\text{hop}_i}) + \\ &(1-y_{\text{hop}_i})\log(1-\hat{y}_{\text{hop}_i}))\big],\end{aligned} \quad (8)$$

where $f_{\text{hop}} : \mathbb{R}^{D'} \to \mathbb{R}$ and $\hat{y}_{hop}$ is the prediction result.

3.4.3 Final SAGD loss

Our structure-aware group discrimination loss can be written as:

$$\mathcal{L} = \alpha\mathcal{L}_{\text{GD}} + \beta\mathcal{L}_{\text{hop}} + \gamma\mathcal{L}_{\bar{r}_i}, \quad (9)$$

where α, β, γ are hyper-parameters used for controlling the contributions of each loss. Empirically, we set α, β, γ as $1, 0.01, 0.05$ respectively in most cases.

3.5 Time Complexity Analysis

The overall time complexity of our method consists of three components: pre-computing, simple MLP encoder, and loss computation. Given a graph $\mathcal{G} = \{X \in \mathbb{R}^{N\times D}, A \in \mathbb{R}^{N\times N}\}$ in the sparse format with the number of edges E. The K-hop message passing takes $O(KED)$ for the pre-computing part. The time complexity of our encoder is $O(ND^2)$. And for the contrastive learning part, the 1-layer MLP projector network takes $O(ND^2)$, and aggregation takes $O(ND)$. The time complexity of basic discrimination loss is $O(N)$. For the structure-preserving module, two 1-layer MLP structural predictor network takes $O(ND^2)$, and the time complexity of structure predict loss is $O(N)$. In the training stage, suppose the training epoch is T, and the training complexity is $O(KED + TN(D^2 + D + 1))$. In the inference stage, the time complexity is $O(ND^2)$.

Method	Training complexity	Inference complexity
GRACE	$O(T(LED+$ LND$^2 + ND^2 + N^2D))$	$O(LED + LND^2)$
GGD	$O(T(LED+$ LND$^2 + ND^2 + ND + N))$	$O(LED + LND^2)$
Ours	$O(KED+$ T(ND$^2 + ND + N))$	$O(ND^2)$

Table 1: Time complexity comparison of different GCL methods

The comparison of time complexity of GRACE and GGD in Table 1. Note that E is ten of times greater than N is large-scale datasets. We can see that the result of speed-up (250x faster inference speed) mostly comes from our adaptive-views graph encoder. This module substantially declines the time cost in message passing during training and inference. And the adaptive weighted training method and structure-preserving module enhance the expression ability in graph structure. .

4 Experiments

In this section, we demonstrate that our framework AVGE-SAGD can achieve comparable performance in unsupervised representation learning for node classification with exceptional training and inference time. We evaluate the performance and computation time cost on various node classification datasets with the standard experiment settings.

4.1 Datasets

The datasets we use to evaluate our approach contain two types desperated by the data scale: small-scale datasets include Cora, CiteSeer, PubMed [16], Amazon Computers and Amazon Photo [17] and large-scale datasets include ogbn-arxiv and ogbn-products provided by Open Graph Benchmark[10]. Dataset statistics can be found in Appendix B.

In our implementation, we follow the standard data splits in [28]. And for Amazon Computers and Photos, we randomly allocate $10/10/80\%$ of data to training/validation/test set respectively.

4.2 Experimental Setup

4.2.1 Model

For the encoder f_θ, we use a 1-layer MLP for all datasets to save computing time. The projector g_ω is also a 1-layer MLP. $f_{\bar{r}}$ and f_{hop} are summation functions used for structure prediction.

4.2.2 Inference

During the inference phase, we freeze the trained MLP encoder f_θ and obtain final node representations $\mathbf{H}$ which can be used for downstream tasks with the processed input data. Since the input data comes from pre-processing, and the encoder is an MLP structure, the graph $\mathcal{G}$ is not needed in the inference stage, which saves a lot of computing resources in the message passing phase compared to the GNN encoder.

We utilize the last-hop features for inference solely as it encompasses multi-order information. Despite our efforts to use alternative features for inference such as the mean, sum or adaptive sum of all hop features, we have empirically discovered that the last-hop features yield the best performance, which is shown in Section 4.5. Different from the training step we use features in all hops as the training data, the final node representation is given from the features of the last hop only. In the training phase, we use the features of each hop. In order to maintain the consistency of the encoder input data, we only use one hop for inference in the inference stage. We choose the feature of the last hop because it contains the most neighbor information. Simple averaging of individual hop features will destroy high-order neighbor information. The final node representation is given by:

$$\mathbf{H} = f_\theta(\tilde{\mathbf{A}}^K\mathbf{X}), \quad (10)$$

where f_θ is the MLP encoder, $\tilde{A}$ is the normalized adjacency matrix of graph $\mathcal{G}$ and $\mathbf{X}$ is the original node attribute.

4.2.3 Evaluation

In our experiment, we evaluate the performance of our method by node classification tasks, following the most common GCL methods[24, 35, 20, 30, 15, 32]. In detail, we train a simple logistic regression classifier by using the final node representations $\mathbf{H}$

	Methods	Cora	CiteSeer	PubMed	Computers	Photo
Supervised	GCN	81.5	70.3	79.0	76.3 ± 0.5	87.3 ± 1.0
	GAT	83.0 ± 0.7	72.5 ± 0.7	79.0 ± 0.3	79.3 ± 1.1	86.2 ± 1.5
	SGC	81.0 ± 0.0	71.9 ± 0.1	78.9 ± 0.0	74.4 ± 0.1	86.4 ± 0.0
Self-supervised	DGI	81.7 ± 0.6	71.5 ± 0.7	77.3 ± 0.6	75.9 ± 0.6	83.1 ± 0.5
	GMI	82.7 ± 0.2	73.0 ± 0.3	80.1 ± 0.2	76.8 ± 0.1	85.1 ± 0.1
	MVGRL	82.9 ± 0.7	72.6 ± 0.7	79.4 ± 0.3	79.0 ± 0.6	87.3 ± 0.3
	GRACE	80.0 ± 0.4	71.7 ± 0.6	79.5 ± 1.1	71.8 ± 0.4	81.8 ± 1.0
	GraphCL	82.5 ± 0.2	72.8 ± 0.3	77.5 ± 0.2	OOM	79.5 ± 0.4
	BGRL	80.5 ± 1.0	71.0 ± 1.2	79.5 ± 0.6	89.2 ± 0.9	91.2 ± 0.8
	GBT	81.0 ± 0.5	72.8 ± 0.2	79.0 ± 0.1	88.5 ± 1.0	91.1 ± 0.7
	GGD	83.9 ± 0.4	73.0 ± 0.6	81.3 ± 0.8	90.1 ± 0.9	92.5 ± 0.6
	Ours	$\mathbf{84.2 \pm 0.5}$	$\mathbf{73.1 \pm 0.8}$	$\mathbf{81.6 \pm 0.2}$	$\mathbf{90.1 \pm 0.3}$	$\mathbf{93.5 \pm 0.3}$

Table 2: Experiments results for node classification task on small-scale datasets. We report accuracy(%) for all datasets. The best performance is in **bold**. OOM represents out-of-memory on NVIDIA GeForce RTX 3090 (24GB).

and test the performance on the various node classification datasets. We measure the model performance using the averaged classification accuracy with ten results.

The evaluation of computation efficiency on large-scale datasets contains two parts: training efficiency and inference efficiency. Training efficiency is measured by the time spent per training epoch and inference efficiency is measured by the time spent for node embedding generation. We do not measure the time spent for classifying embeddings because we keep the complexity of the classifier the same. Note that in our framework, the message passing step does not need back propagation so that it can be separated from the encoder training procedure. This calculation can be done on other servers in a distributed system. Therefore we do not measure the time required for message passing in ogbn-arxiv. However, in ogbn-products, even if other servers are used to calculate message passing, the calculation time is still very long. So we count the time required to calculate message passing locally.

4.2.4 Baselines

First, we compare our framework with supervised GNNs (i.e., GCN [11], GAT [23], SGC [25]). Then we compare with some classical GCL methods (i.e., DGI [24], MVGRL [8], GRACE [34], GMI [14], BGRL [19], GBT [1]). Finally, we compare a newly proposed efficient GCL method GGD [32]. The reported results of some baselines are from previous papers if available.

4.3 Results and Analysis

4.3.1 Results for small-scale datasets

Table 2 shows the classification results on five small-scale datasets, and we can draw some conclusions: (i) Experiment results show that our framework outperforms supervised GNNs and other state-of-the-art GCL baselines in all datasets, which shows the superiority of our AVGE-SAGD framework. (ii) Compared with GGD, our method surpasses it by a considerable margin (*e.g.*, 1% absolute improvement on Photo dataset) indicating the significance of structure-aware group discrimination. Our structure-aware group discrimination performs topology-based mini-group classification on the basis of graph group discrimination, which helps the model to learn more rich knowledge.

4.3.2 Results for large-scale datasets

We evaluate the classification accuracy and computational efficiency of our model on two large-scale datasets provided by OGB[10]: ogbn-arxiv and ogbn-products.

Experiment results in Table 4 and Table 5 show that our framework has faster training speed and faster inference speed than most GCL frameworks, as well as GGD, which also uses group discrimination instead of pairwise contrastive learning paradigm. Although the result of our method is slightly lower than GRACE and BRGL in ogbn-arxiv, it saves a lot of computing resources and is memory-friendly. For ogbn-arxiv, we are 266 $\times$ faster than GGD in inference time and for ogbn-products we are 301 $\times$ faster. Since our message passing process does not contain parameters, our framework is still faster than the other GCL frameworks using GCN encoder. Due to the addition of auxiliary modules and tasks in our framework, which increases the number of additional calculations, the training speed improvement is relatively limited. But in the inference stage, the size of our model is equivalent to a simple MLP. So the inference efficiency has been greatly improved.

On the other hand, it is observed that on the large-scale dataset provided by OGB, the performance of GCL is inferior to the basic supervised GCN. The reason is there are plenty of training data on these datasets while the main contribution of GCL is the scenario lacking training data, so it cannot performs better than supervised models on these datasets. In the small-scale datasets with very limited training data mentioned in the last paragraph, however, the overall performance of GCL is significantly improved compared with the supervised models.

4.4 Visualization

To visually assess the quality of our learned embeddings, we adopt t-SNE [22] to visualize the 2D projection of node embeddings on Cora dataset using raw features, random-init of AVGE-SAGD, GGD, and trained AVGE-SAGD in Figure 4, where nodes in different labels have different colors.

We can observe that the distribution of node embeddings in raw features and random-init are messy and intertwined. After training, node embeddings learned by AVGE-SAGD have a clear separation of clusters, which indicates the model can help learn representative features for downstream tasks. Compared to GGD, the margins of each cluster of node embeddings learned from AVGE-SAGD are much wider, which means higher quality.

	Multi-View Weights	Structure Preserving	Cora	Citeseer	PubMed	Comp	Photo
Fixed Weights	$\lambda = [0, 0, ..., 1]$		83.5±0.3	71.7±0.7	80.9±0.5	89.9± 0.2	92.8±0.3
	$\lambda = [1, 1, ..., 1]$		83.4±0.4	71.7±0.4	81.0±0.4	89.6±0.4	92.8±0.4
	$\lambda = [1, 1, ..., 1]$	✔	83.5±0.7	71.8±0.4	81.2±0.5	90.0±0.2	93.2±0.1
Learnable Weights	$\min_\theta \mathcal{L}$		83.6±0.4	71.8±0.4	81.2±0.4	90.0±0.3	93.2±0.2
	$\min_\theta$-$\max_\lambda \mathcal{L}$		84.0±0.6	71.9±0.5	81.3±0.3	90.1±0.2	93.3±0.3
	$\min_\theta$-$\max_\lambda \mathcal{L}$	✔	**84.2±0.5**	**72.0±0.3**	**81.4±0.1**	**90.2±0.2**	**93.5±0.2**

Table 3: Ablation studies for AVGE-SAGD

(a) raw features (b) random-init (c) GGD (d) AVGE-SAGD

Figure 4: The t-SNE visualization result of node embeddings on Cora dataset. (a) is the raw features, (b) is the node embeddings from random initialized AVGE-SAGD, (c)is the learned representation of GGD, (d) is the learned representation of AVGE-SAGD.

Methods	Accuracy (%)	Training Time (s)	Inference Time (s)
GCN	71.7±0.3	-	-
MLP	55.5±0.2	-	-
Node2vec	70.1±0.1	-	-
DGI	70.3±0.2	/	/
GRACE	71.5±0.1	/	/
BGRL	71.6±0.1	/	/
GBT	70.1±0.2	6.19	0.13
GGD*	71.2±0.2	1.00	0.08
Ours	71.3±0.3	0.54	0.0003

Table 4: Accuracy on node classification task and speed test on the large-scale dataset ogbn-arxiv. 'Training Time' represents the average training time in each epoch. 'Inference Time' represents the time required from inputting data to computing the embedding. GGD* is the re-implementation on our devices with their official code. '/' means the method is OOM under a full-graph training setting.

Methods	Accuracy (%)	Training Time (s)	Inference Time (s)
GCN	75.6±0.2	-	-
MLP	61.1±0.0	-	-
Node2vec	68.8±0.0	-	-
BGRL	64.0±1.6	2267	265
GBT	70.5±0.4	1963	262
GGD*(1024)	75.6±0.2	779	718
GGD*(256)	73.3±0.4	555	301
Ours(256)	75.9±0.1	364	1

Table 5: Accuracy on node classification task and speed test on the large-scale dataset ogbn-products. In the method column, the number in the brackets means the dimension of embeddings. In the accuracy column, the number in the brackets means the training time with message passing. GGD* is the re-implementation on our devices with their official code.

4.5 Ablation Study

To prove the effectiveness of the design module of our framework, we conduct ablation experiments with different modules under the same hyperparameters on five small datasets. In Table 3, 'Multi-View Weights' includes different strategies for adopting weights on multi-view attributes by masking different components. The first three rows assign fixed weights to different hop attributes. $[0, 0, ..., 1]$ means we only use the attributes of the last hop to train the encoder. $[1, 1, ..., 1]$ means we keep the contributions of different hop attributes the same. The last three columns represent that we use learnable weights to adjust weights adaptively. 'min' represents that we optimize weights and model parameters by minimizing training loss. 'min-max' represents that we use a two-step adaptive weighted training method, 'Structure Preserving' means structure-aware module.

The results show that all of the modules we design are helpful for the performance of our framework. The two-step min-max adaptive weight training method is the most significant part in the framework since the performance degrades without it. And with structure-preserving module, SAGD outperforms GGD in our framework. Furthermore, we observe that using fixed multi-hop feature training performs worse than using the last-hop feature only, which underscores the importance of our adaptive weighted training approach.

5 Conclusion

In this paper, we approach to the challenge of increasing the training and inference efficiency of the graph contrastive representation learning frameworks. In terms of improving the encoder's efficiency, we separate the message passing from the embedding prediction and design a novel adversarially adaptive weights multi-hop features. As for the pre-training loss, we built a new structure-aware group discrimination loss that helps our fast encoder to preserve more structural information, which consequently improves its generalization ability on the downstream tasks. Extensive experiments conducted on both small-scale and large-scale datasets have shown the effectiveness of our framework regarding both downstream task performance and the training and inference speed.

Acknowledgements

This work has been supported in part by the Zhejiang NSF (LR21F020004), the NSFC (No. 62272411), Alibaba-Zhejiang University Joint Research Institute of Frontier Technologies, and Ant Group.

References

[1] Piotr Bielak, Tomasz Kajdanowicz, and Nitesh V Chawla, 'Graph barlow twins: A self-supervised representation learning framework for graphs', *Knowledge-Based Systems*, **256**, 109631, (2022).

[2] Sudhanshu Chanpuriya and Cameron Musco, 'Simplified graph convolution with heterophily', *arXiv preprint arXiv:2202.04139*, (2022).

[3] 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*, volume 34, pp. 3438–3445, (2020).

[4] Austin Derrow-Pinion, Jennifer She, David Wong, Oliver Lange, Todd Hester, Luis Perez, Marc Nunkesser, Seongjae Lee, Xueying Guo, Brett Wiltshire, et al., 'Eta prediction with graph neural networks in google maps', in *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pp. 3767–3776, (2021).

[5] Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin, 'Graph neural networks for social recommendation', in *The world wide web conference*, pp. 417–426, (2019).

[6] Xavier Glorot and Yoshua Bengio, 'Understanding the difficulty of training deep feedforward neural networks', in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256. JMLR Workshop and Conference Proceedings, (2010).

[7] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Guo, Mohammad Gheshlaghi Azar, et al., 'Bootstrap your own latent-a new approach to self-supervised learning', *Advances in neural information processing systems*, **33**, 21271–21284, (2020).

[8] Kaveh Hassani and Amir Hosein Khasahmadi, 'Contrastive multi-view representation learning on graphs', in *International Conference on Machine Learning*, pp. 4116–4126. PMLR, (2020).

[9] Liancheng He, Liang Bai, and Jiye Liang, 'The impact of neighborhood distribution in graph convolutional networks'.

[10] 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', *Advances in neural information processing systems*, **33**, 22118–22133, (2020).

[11] Thomas N Kipf and Max Welling, 'Semi-supervised classification with graph convolutional networks', *arXiv preprint arXiv:1609.02907*, (2016).

[12] Johannes Klicpera, Stefan Weißenberger, and Stephan Günnemann, 'Diffusion improves graph learning', *arXiv preprint arXiv:1911.05485*, (2019).

[13] Yujie Mo, Liang Peng, Jie Xu, Xiaoshuang Shi, and Xiaofeng Zhu, 'Simple unsupervised graph representation learning'. AAAI, (2022).

[14] Zhen Peng, Wenbing Huang, Minnan Luo, Qinghua Zheng, Yu Rong, Tingyang Xu, and Junzhou Huang, 'Graph representation learning via graphical mutual information maximization', in *Proceedings of The Web Conference 2020*, pp. 259–270, (2020).

[15] Zhen Peng, Wenbing Huang, Minnan Luo, Qinghua Zheng, Yu Rong, Tingyang Xu, and Junzhou Huang, 'Graph representation learning via graphical mutual information maximization', in *Proceedings of The Web Conference 2020*, pp. 259–270, (2020).

[16] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad, 'Collective classification in network data', *AI magazine*, **29**(3), 93–93, (2008).

[17] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann, 'Pitfalls of graph neural network evaluation', *arXiv preprint arXiv:1811.05868*, (2018).

[18] Haizhou Shi, Dongliang Luo, Siliang Tang, Jian Wang, and Yueting Zhuang, 'Run away from your teacher: Understanding byol by a novel self-supervised approach', *arXiv preprint arXiv:2011.10944*, (2020).

[19] Shantanu Thakoor, Corentin Tallec, Mohammad Gheshlaghi Azar, Rémi Munos, Petar Veličković, and Michal Valko, 'Bootstrapped representation learning on graphs', in *ICLR 2021 Workshop on Geometrical and Topological Representation Learning*, (2021).

[20] Shantanu Thakoor, Corentin Tallec, Mohammad Gheshlaghi Azar, Rémi Munos, Petar Veličković, and Michal Valko, 'Bootstrapped representation learning on graphs', in *ICLR 2021 Workshop on Geometrical and Topological Representation Learning*, (2021).

[21] Yijun Tian, Chuxu Zhang, Zhichun Guo, Xiangliang Zhang, and Nitesh V Chawla, 'Nosmog: Learning noise-robust and structure-aware mlps on graphs', *arXiv preprint arXiv:2208.10010*, (2022).

[22] Laurens Van der Maaten and Geoffrey Hinton, 'Visualizing data using t-sne.', *Journal of machine learning research*, **9**(11), (2008).

[23] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio, 'Graph attention networks', *stat*, **1050**, 20, (2017).

[24] Petar Velickovic, William Fedus, William L Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm, 'Deep graph infomax.', *ICLR (Poster)*, **2**(3), 4, (2019).

[25] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger, 'Simplifying graph convolutional networks', in *International conference on machine learning*, pp. 6861–6871. PMLR, (2019).

[26] Yutong Xie, Chence Shi, Hao Zhou, Yuwei Yang, Weinan Zhang, Yong Yu, and Lei Li, 'Mars: Markov molecular sampling for multi-objective drug discovery', *arXiv preprint arXiv:2103.10432*, (2021).

[27] Yujun Yan, Milad Hashemi, Kevin Swersky, Yaoqing Yang, and Danai Koutra, 'Two sides of the same coin: Heterophily and oversmoothing in graph convolutional neural networks', *arXiv preprint arXiv:2102.06462*, (2021).

[28] Zhilin Yang, William Cohen, and Ruslan Salakhudinov, 'Revisiting semi-supervised learning with graph embeddings', in *International conference on machine learning*, pp. 40–48. PMLR, (2016).

[29] Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen, 'Graph contrastive learning with augmentations', *Advances in Neural Information Processing Systems*, **33**, 5812–5823, (2020).

[30] Jure Zbontar, Li Jing, Ishan Misra, Yann LeCun, and Stéphane Deny, 'Barlow twins: Self-supervised learning via redundancy reduction', in *International Conference on Machine Learning*, pp. 12310–12320. PMLR, (2021).

[31] Wentao Zhang, Zeang Sheng, Mingyu Yang, Yang Li, Yu Shen, Zhi Yang, and Bin Cui, 'Nafs: A simple yet tough-to-beat baseline for graph representation learning', in *International Conference on Machine Learning*, pp. 26467–26483. PMLR, (2022).

[32] Yizhen Zheng, Shirui Pan, Vincent Cs Lee, Yu Zheng, and Philip S Yu, 'Rethinking and scaling up graph contrastive learning: An extremely efficient approach with group discrimination', *arXiv preprint arXiv:2206.01535*, (2022).

[33] Jiong Zhu, Yujun Yan, Lingxiao Zhao, Mark Heimann, Leman Akoglu, and Danai Koutra, 'Beyond homophily in graph neural networks: Current limitations and effective designs', *Advances in Neural Information Processing Systems*, **33**, 7793–7804, (2020).

[34] Yanqiao Zhu, Yichen Xu, Feng Yu, Qiang Liu, Shu Wu, and Liang Wang, 'Deep graph contrastive representation learning', *arXiv preprint arXiv:2006.04131*, (2020).

[35] Yanqiao Zhu, Yichen Xu, Feng Yu, Qiang Liu, Shu Wu, and Liang Wang, 'Deep graph contrastive representation learning', *arXiv preprint arXiv:2006.04131*, (2020).

[36] Yun Zhu, Jianhao Guo, Fei Wu, and Siliang Tang, 'Rosa: A robust self-aligned framework for node-node graph contrastive learning', *arXiv preprint arXiv:2204.13846*, (2022).