ClusterSCL: Cluster-Aware Supervised Contrastive Learning on Graphs

Yanling Wang[1,2], Jing Zhang[1,2,*], Haoyang Li[1,2], Yuxiao Dong[3]
Hongzhi Yin[4], Cuiping Li[1,2], Hong Chen[1,2]
[1] School of Information, Renmin University of China, Beijing, China
[2] Key Laboratory of Data Engineering and Knowledge Engineering of Ministry of Education, Renmin University of China, Beijing, China
[3] Department of Computer Science and Technology, Tsinghua University, Beijing, China
[4] School of Information Technology and Electrical Engineering, The University of Queensland, Brisbane, Australia
{wangyanling,zhang-jing,lihaoyang.cs,licuiping,chong}@ruc.edu.cn,yuxiaod@tsinghua.edu.cn,h.yin1@uq.edu.au

ABSTRACT

We study the problem of supervised contrastive (SupCon) learning on graphs. The SupCon loss has been recently proposed for classification tasks by pulling data points in the same class closer than those of different classes. However, it could be difficult for SupCon to handle datasets with large intra-class variances and high inter-class similarities. This issue is also challenging when it couples with graph structures. To address this, we present the cluster-aware supervised contrastive learning loss (ClusterSCL[1]) for graph learning tasks. The main idea of ClusterSCL is to retain the structural and attribute properties of a graph in the form of nodes' cluster distributions during supervised contrastive learning. Specifically, ClusterSCL introduces the strategy of cluster-aware data augmentation and integrates it with the SupCon loss. Extensive experiments on several widely adopted graph benchmarks demonstrate the superiority of ClusterSCL over the cross-entropy, SupCon, and other graph contrastive objectives.

CCS CONCEPTS

• **Computing methodologies** → **Supervised learning by classification**; **Probabilistic reasoning**.

KEYWORDS

Supervised contrastive learning, Clustering, Data augmentation

ACM Reference Format:
Yanling Wang[1,2], Jing Zhang[1,2,*], Haoyang Li[1,2], Yuxiao Dong[3], Hongzhi Yin[4], Cuiping Li[1,2], Hong Chen[1,2]. 2022. ClusterSCL: Cluster-Aware Supervised Contrastive Learning on Graphs. In *Proceedings of the ACM Web Conference 2022 (WWW '22), April 25–29, 2022, Virtual Event, Lyon, France.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3485447.3512207

[1]The code is available at https://github.com/wyl7/ClusterSCL or https://github.com/RUCKBReasoning/ClusterSCL.

* Corresponding author.

WWW '22, April 25–29, 2022, Virtual Event, Lyon, France

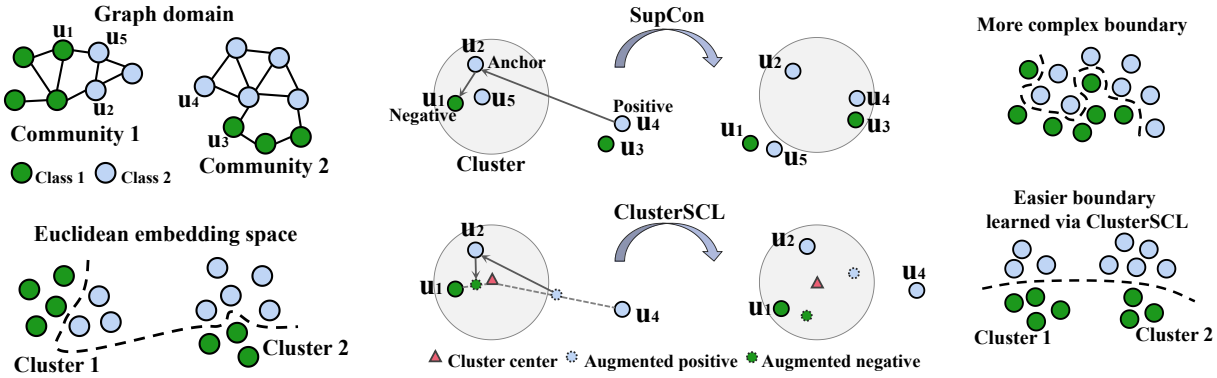
ACM ISBN 978-1-4503-9096-5/22/04…$15.00
https://doi.org/10.1145/3485447.3512207

1 INTRODUCTION

Graph data, which has been broadly studied in a variety of research domains, such as social science [2, 13, 52, 60, 62], computational biology [39, 40, 56] and knowledge graph [18, 49], provides powerful machinery for effectively capturing the rich correlations among data points. In this work, we focus on node classification — a widely studied task on graphs. Inspired by the advances of graph neural networks (GNNs) [11, 23, 28, 47, 55], the most popular architecture for performing node classification includes two key components: a GNN encoder for encoding node representations and a linear or MLP classifier for predicting node labels.

Extensive researches promote the quality of classification via studying promising GNN encoders for better representing nodes [11, 16, 28, 47, 55]. The cross-entropy loss is the most widely used loss function for supervised learning of the GNN encoder and the classifier. Recently, supervised contrastive learning (SupCon) [21] which extends the unsupervised contrastive learning (CL) [5] to the fully supervised setting, has shown advantages over cross-entropy loss on the large-scale ImageNet classification tasks [21]. SupCon pulls representations of the same class closer than those of different classes, reducing the difficulty of discovering the classification boundaries between classes. This motivates us to adopt the competitive SupCon loss in the tasks of node classification.

Despite the success of SupCon, intra-class variances and inter-class similarities in the real-world graph data make the SupCon loss sub-optimal. The top of Figure 1(a) shows an example of node classification in a social network, where (u_1, u_3) or (u_2, u_4) belong to the same class but have distinct interests and live in different graph communities (intra-class variances), while (u_1, u_2) or (u_3, u_4) come from different classes but share similar graph patterns (inter-class similarities). Such nodes result in a complex decision boundary in the Euclidean embedding space (the bottom of Figure 1(a)). When performing SupCon for the anchor node u_2, as illustrated in the top of the Figure 1(b), for the positive sample pairs that belong to the same class but locate in distinct clusters such as (u_2, u_4), simply pulling them together in the embedding space could indirectly pull together the nodes of different classes such as (u_2, u_3), since u_3 is structurally similar to u_4. Meanwhile, for the negative sample pairs that belong to different classes but locate in the same cluster such as (u_2, u_1), simply pushing them away could indirectly push away the nodes of the same class such as (u_2, u_5), since u_5 is structurally similar to u_1. In other words, contrasting the nodes with



(a) Node classification with intra-class variances and inter-class similarities.

(b) Comparison between SupCon and ClusterSCL.

(c) Decision boundaries learned by different loss functions.

Figure 1: Motivation of our work (best seen in color). Different colors of nodes represent different classes. The gray area in (b) denotes the cluster derived based on intrinsic data property. (a) Node classification with intra-class variances and inter-class similarities. (u_1, u_3) or (u_2, u_4) are of the same class but locate in distinct graph communities. (u_1, u_2) or (u_3, u_4) are of different classes but share similar structural patterns. (b) Comparison between SupCon and ClusterSCL in the embedding space. SupCon could break the intrinsic cluster distributions, while ClusterSCL performs node contrast as well as retaining the cluster distributions. (c) Decision boundaries learned by different loss functions. ClusterSCL enables an easier decision boundary.

large intra-class variances and high inter-class similarities could misinterpret the true graph property, resulting in a more complex decision boundary shown in the top of Figure 1(c).

To address this problem, **we propose to learn the intrinsic graph property expressed by nodes' cluster distributions and retain it when performing SupCon learning**, expecting to learn an easier decision boundary as illustrated in the bottom of Figure 1(c). To be specific, we conduct clustering in the embedding space to learn the cluster distribution for each node. In order to retain the cluster distributions during SupCon learning, we can straightforwardly perform SupCon within the same cluster. However, this solution ignores the contrast between some potentially useful sample pairs such as (u_1, u_3) and $(u2, u4)$, since they are partitioned into different clusters. Empirical studies in Section 5.3 verify the limitation. Alternatively, we design a cluster-aware data augmentation (CDA) module to generate the augmented positives and negatives for each anchor. The bottom of Figure 1(b) illustrates the basic idea. The augmented samples stay in or closer to the cluster of the anchor to narrow the space for SupCon learning. By doing this, the pulling strength of u_4 to its anchor u_2 and the pushing strength of u_1 apart from its anchor u_2 can be indirectly weakened (compared with SupCon) to help retain the nodes' cluster distributions.

Formally, we propose a simple and effective contrastive learning scheme called **Cluster**-Aware **S**upervised **C**ontrastive **L**earning (ClusterSCL), to unify the node clustering, the CDA, and the SupCon. ClusterSCL is a probabilistic model, where we introduce a latent variable to represent which cluster an anchor node should belong to and formalize the supervised contrast as a discrimination task with a latent variable. To optimize the feature encoder via maximizing the log conditional likelihood, we develop a variant of the Variational Expectation-Maximization (Variational EM) algorithm for learning and inference of ClusterSCL.

Overall, our main contributions are summarized as follows:

- We design a cluster-aware data augmentation (CDA) method to softly constrain the SupCon learning by node clusters, so that the negative impact induced by intra-class variances and inter-class similarities can be alleviated.
- ClusterSCL unifies the clustering, the cluster-aware data augmentation, and the supervised contrastive learning via a probabilistic model to make use of their interactions.
- We conduct extensive experiments on the widely adopted graph benchmarks for node classification to evaluate our model. ClusterSCL is proposed for node classification, but the main thought is not restricted to this task.

2 RELATED WORK

The work is closely related to contrastive representation learning and data augmentation.

2.1 Contrastive Representation Learning

Contrastive learning (CL) has become one of the most popular approaches for unsupervised representation learning. The core idea behind is to pull together an anchor and its positive samples in the embedding space, while push apart the anchor from its negative samples. Triplet loss [36], lifted structured loss [42] and N-pair loss [41] are classic contrastive loss functions. Recently, the agreement within a positive sample pair is formalized by mutual information (MI) maximization. To facilitate the computation of MI, Jensen-Shannon divergence [8, 29] and NCE/InfoNCE loss [10, 14, 46] are adopted in practice. In the graph domain, representation learning methods such as DGI [48], DCI [53], InfoGraph [43], MVGRL [12], GCC [31], GraphCL [58], and JOAO [57], are inspired by CL a lot and achieve good performance on many graph mining tasks. Earlier attempts of unsupervised graph representation learning such

as GAE [22], node2vec [9], deepwalk [30], and LINE [44], can be viewed as a case of CL, which treats structurally nearby nodes as the positive pairs and encourages them to have similar representations.

On a parallel note, the thought of CL is applied to the supervised setting. CLIP [32] maximizes the agreement between the vision representation and the language representation of the same thing. Currently, SupCon [21] extends the batch unsupervised contrastive learning [5] to a fully supervised setting, where each positive sample pair consists of samples from the same class. Specially, SupCon resembles the soft-nearest neighbors loss [35, 54]. Please refer to [21] for the detailed comparison between SupCon and soft-nearest neighbors loss. In this work, we inherit SupCon to perform supervised learning of the node classification models. Instead of directly using SupCon, we propose ClusterSCL to alleviate the negative impacts induced by the intra-class variances and the inter-class similarities.

2.2 Data Augmentation

Data augmentation is an important technique employed by CL. It changes the pattern of a sample but remains the semantic information unchanged, so that the augmented data can be viewed as another version (i.e., a positive sample) of the original sample. In computer vision, there are many well-studied methods for data augmentation, such as random rotation, random cropping, and random Gaussian blur [5]. Some studies learn augmentation strategies via searching for the combination of different augmentations [6, 7, 17]. In the graph domain, graph augmentation also raises significant interest. For the node-level augmentation, ego-network sampling [31] is a good choice so far. Feature transformation [58], structure transformation [12, 31, 58] and subgraph-sampling [31, 58] for the graph-level augmentation, also receive lots of attentions. Currently, JOAO [57] is proposed to automatically select augmentations on the specific graph data.

Besides generating positive samples for contrastive learning, data augmentation can be used to enlarge the training set to improve the generalization ability of neural networks. Adversarial learning-based methods [4, 24, 50] train a generator to augment the training examples. The mixup-based method which is simple yet effective is another popular solution. *mixup* [61] generates new training instances by mixing up samples across different classes. Cutmix [59] improves the model robustness by cutting and pasting patches among training images. MoCHi [19] facilitates better and faster learning via generating hard negatives with the mixing strategy. The proposed cluster-aware data augmentation (CDA) also follows the mixing strategy. Differently, CDA aims to refine the positive and negative samples for SupCon, and CDA is integrated into the learning process, instead of being an independent module.

3 PROBLEM FORMALIZATION

In this section, we formalize the problem of node classification and introduce the popular GNN-based node classification model, then describe two training settings.

Let $G = (V, E, X)$ be an attributed undirected graph, where $V = \{v_1, v_2, \cdots, v_{|V|}\}$ denotes the set of nodes, and $E = \{e_1, e_2, \cdots, e_{|E|}\}$ denotes the set of edges. $X \in \mathbb{R}^{|V| \times d_0}$ is the matrix of node attributes, where d_0 is the dimension of an attribute vector, and the i-th row of X denotes the attribute vector of node v_i. The adjacency matrix of G is denoted by $A \in \mathbb{R}^{|V| \times |V|}$, where $a_{ij} = 1$ if $(v_i, v_j) \in E$ and 0 otherwise.

In this paper, we consider the problem of classifying nodes in a graph. Formally, the problem with K classes is defined as follows:

Problem 1. ***Node Classification in a Graph.*** *Given a partially labeled graph $G = \left(V, E, X, Y^L\right)$, where Y^L denotes the available node labels in G, the objective is to learn a mapping function:*

$$\mathcal{F} : G = \left(V, E, X, Y^L\right) \rightarrow Y \tag{1}$$

where $Y \in \mathbb{R}^{|V| \times K}$ includes Y^L and Y^U, and Y^U is the unavailable node labels which are to be predicted. Y^L is used as supervision for optimizing $\mathcal{F}$. In particular, each row of Y is a one-hot or multi-hot vector to represent the label of the corresponding node. In this paper, we only consider the one-hot case.

Graph neural networks (GNNs) [11, 23, 28, 47, 55] have been widely adopted to solve the defined problem. Concretely, the mapping function $\mathcal{F}$ consists of a GNN encoder g_θ and a linear or MLP classifier f_ϕ, where g_θ encodes the node representations, then f_ϕ predicts node labels based on the node representations output by g_θ. More precisely,

$$\hat{Y} = \mathcal{F}(G) = f_\phi(g_\theta(G)) \tag{2}$$

where $\hat{Y} \in \mathbb{R}^{|V| \times K}$ is the predicted node labels. θ and ϕ are the trainable parameters. Now we introduce two supervised settings to train g_θ and f_ϕ.

End-to-end Training with Cross-entropy Loss. In general, the node labels are predicted in an end-to-end manner. Cross-entropy is the most popular loss function to simultaneously train g_θ and f_ϕ.

Two-stage Training with Supervised Contrastive Loss. Supervised contrastive (SupCon) loss encourages samples of the same class to have similar representations, while pushes apart samples of different classes in the embedding space. The first stage computes node representations via $H = g_\theta(G)$ and trains g_θ via the SupCon loss. Then based on the learned g_θ, the second stage predicts the node labels via $\hat{Y} = f_\phi(g_\theta(G))$ and trains f_ϕ via cross-entropy loss. We suggest to simultaneously train g_θ and f_ϕ at the second stage.

SupCon has shown great success in image classification tasks [21], so we employ this competitive loss function for supervised node classification. Different from the original SupCon, we directly sample instances of the same class to construct positive sample pairs without performing extra data augmentation. Note that CDA is conducted on each constructed sample pair.

4 METHOD

In this section, we briefly review the base loss function SupCon and then propose ClusterSCL. Furthermore, we discuss the relations between ClusterSCL and some popular research topics.

4.1 Base CL Scheme: SupCon

ClusterSCL follows the thought of supervised contrastive (SupCon) learning [21] to learn node representations, where each positive sample pair consists of samples from the same class, while each negative sample pair is formed by samples randomly chosen from

the batch. Given N randomly sampled nodes, we assign each node a positive sample by randomly sampling from the other nodes in the corresponding class. Thus, a batch B contains $2N$ nodes. Since each node can share class label with multiple nodes, the node could have multiple positive samples in the batch.

For the positive samples of node v_i within the batch, their indices are denoted as set S_i. Specifically, $s_i \in S_i$ is a surrogate label of v_i, which denotes the index of a positive sample of v_i. The SupCon loss function is formalized as follows:

$$\mathcal{L}_{\text{SupCon}} = -\sum_{v_i \in B} \frac{1}{|S_i|} \sum_{s_i \in S_i} \log \frac{\exp\left(\mathbf{h}_i^\top \mathbf{h}_{s_i}/\tau\right)}{\sum_{v_j \in B \setminus \{v_i\}} \exp\left(\mathbf{h}_i^\top \mathbf{h}_j/\tau\right)} \tag{3}$$

where $\mathbf{h}$ denotes ℓ_2-normalized representations, and τ is the temperature parameter.

Intuitively, if nodes of the same class share the similar patterns, pulling the same-labeled nodes together in the embedding space via SupCon agrees with the true data property. However, nodes of the same class may have diverse patterns (i.e., intra-class variances), and nodes of different classes could be similar to each other (i.e, inter-class similarities). In such a case, representations learned by SupCon could misinterpret the true data property. In view of this, a question is raised: **is there a better way to construct the positive sample pairs and negative sample pairs for SupCon learning?**

4.2 Proposed CL Scheme: ClusterSCL

ClusterSCL answers the above question by a cluster-aware data augmentation (CDA) module. Specially, ClusterSCL unifies the clustering, the CDA and the SupCon learning via a probabilistic model.

Cluster-aware Data Augmentation (CDA). Assume that there exist M latent clusters which need to be retained during SupCon learning, we introduce the latent variable $c_i \in \{1, 2, \ldots, M\}$ to indicate which cluster node v_i should be clustered to. Given an anchor node v_i and a node v_j, CDA constructs an augmented version of v_j at the feature level for SupCon learning via the following linear interpolation:

$$\tilde{\mathbf{h}}_j = \alpha \mathbf{h}_j + (1-\alpha)\, \mathbf{w}_{c_i} \tag{4}$$

where $\mathbf{w} = \{\mathbf{w}_m\}_{m=1}^M$ represents the cluster prototypes and is shared across different batches. $\tilde{\mathbf{h}}_j$ includes information from v_j and stays closer to the cluster that the anchor node v_i belongs to. These virtual augmentations narrow the feature space for SupCon learning, such that the pulling strength between a distant positive sample pair and the pushing strength between a close negative sample pair can be indirectly weakened to help retain the nodes' cluster distributions.

Essentially, α controls the strength of pulling (pushing) the original sample pair (v_i, v_j). We aim to adjust the value of α for each sample pair automatically. The main idea to do this is illustrated in Figure 2. If the anchor node v_i and the contrasted sample v_j already stay close to each other in the embedding space, we tend to directly contrast between them. So we use a larger α to include more information from v_j into the augmented sample. Conversely, if v_i and v_j are far away from each other in the embedding space, we use a smaller α to decay the information from v_j to guarantee that the anchor and the augmented sample are not too far away from each other. Considering that Eq. 3 explicitly models the pulling between each positive sample pair rather than the pushing between each negative sample pair, we design the principle of adjusting α from the perspective of positive sample pairs. We apply the same principle to the negative sample pairs, and CDA takes effect demonstrated by the experimental results. Refining the principle for negative sample pairs deserves further studies in the future work. Here, we calculate the weight α by:

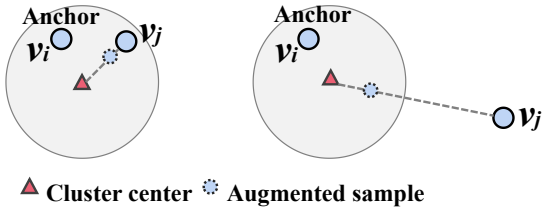


Figure 2: An illustration of cluster-aware data augmentation (CDA). The weight of linear interpolation between the cluster center and the node embedding of v_j is determined by their distances (or similarities) to the anchor v_i.

$$\alpha = \frac{\exp\left(\mathbf{h}_i^\top \mathbf{h}_j\right)}{\exp\left(\mathbf{h}_i^\top \mathbf{h}_j\right) + \exp\left(\mathbf{h}_i^\top \mathbf{w}_{c_i}\right)} \tag{5}$$

Since $\mathbf{h}_i$ and $\mathbf{h}_j$ lie on the surface of a hypersphere of radius 1, we have $||\mathbf{h}_i - \mathbf{h}_j||^2 = 2 - 2\mathbf{h}_i^\top \mathbf{h}_j$, so a large inner product is equivalent to a small squared Euclidean distance.

At the high level, both *mixup* [61] and CDA adopt the operation of linear interpolation to generate virtual data points. Here, **we'd like to clarify how CDA differs from *mixup*:**

- Objective-wise, *mixup* is proposed to enlarge the training set for enhancing the generalization ability of neural networks, while CDA aims to handle the problem of intra-class variances and inter-class similarities in SupCon learning.
- Technique-wise, *mixup* performs linear interpolations between two samples, while CDA performs linear interpolations between a sample and a cluster. Specifically, we interpolate the representations of an anchor's cluster center and the anchor's positive (negative) sample.
- Learning-wise, *mixup* is independent from the learning process, while CDA in ClusterSCL is integrated into the learning process to take advantages of the learnable parameters.

Integrating Clustering and CDA into SupCon Learning. Based on the augmentations derived by CDA, we model the following instance discrimination task:

$$\begin{aligned} p\left(s_i | v_i, c_i\right) &= \frac{\exp\left(\mathbf{h}_i^\top \tilde{\mathbf{h}}_{s_i}/\tau\right)}{\sum_{v_j \in V \setminus \{v_i\}} \exp\left(\mathbf{h}_i^\top \tilde{\mathbf{h}}_j/\tau\right)} \\ &= \frac{\exp\left(\mathbf{h}_i^\top \left(\alpha \mathbf{h}_{s_i} + (1-\alpha)\, \mathbf{w}_{c_i}\right)/\tau\right)}{\sum_{v_j \in V \setminus \{v_i\}} \exp\left(\mathbf{h}_i^\top \left(\alpha \mathbf{h}_j + (1-\alpha)\, \mathbf{w}_{c_i}\right)/\tau\right)} \end{aligned} \tag{6}$$

Before performing CDA, we need to know which cluster the anchor node v_i belongs to via:

$$p\left(c_i|v_i\right) = \frac{\exp\left(\mathbf{h}_i^\top \mathbf{w}_{c_i}/\kappa\right)}{\sum_{m=1}^{M}\exp\left(\mathbf{h}_i^\top \mathbf{w}_m/\kappa\right)} \tag{7}$$

where κ is the temperature parameter for adjusting the softness of the predicted cluster distributions, and $p\left(c_i|v_i\right)$ can be viewed as a prototype-based soft clustering module.

Since we have modeled the soft clustering module $p\left(c_i|v_i\right)$ and the cluster-aware discriminator $p\left(s_i|v_i, c_i\right)$, ClusterSCL can be modeled as the following instance discrimination task:

$$p\left(s_i|v_i\right) = \int p\left(c_i|v_i\right) p\left(s_i|v_i, c_i\right) dc_i \tag{8}$$

Please see Figure 3 for an overview of ClusterSCL .

Inference and Learning. In effect, it is non-trivial to maximize the log likelihood over the whole training data due to the summation within the log operation. We can adopt EM algorithm to solve this problem, where we need to calculate the posterior distribution:

$$p\left(c_i|v_i, s_i\right) = \frac{p\left(c_i|v_i\right) p\left(s_i|v_i, c_i\right)}{\sum_{m=1}^{M} p\left(m|v_i\right) p\left(s_i|v_i, m\right)} \tag{9}$$

However, it is prohibitive to compute the posterior distribution due to the summation over the entire nodes $\sum_{v_j \in V\setminus\{v_i\}} \exp\left(\mathbf{h}_i^\top \tilde{\mathbf{h}}_j/\tau\right)$. Alternatively, we maximize the evidence lower bound (ELBO) of $\log p\left(s_i|v_i\right)$ given by:

$$\begin{aligned}\log p\left(s_i|v_i\right) &\geq \mathcal{L}_{\text{ELBO}}\left(\boldsymbol{\theta}, \mathbf{w}; v_i, s_i\right) \\ &:= \mathbb{E}_{q\left(c_i|v_i,s_i\right)}\left[\log p\left(s_i|v_i, c_i\right)\right] \\ &\quad - \text{KL}\left(q\left(c_i|v_i, s_i\right)||p\left(c_i|v_i\right)\right)\end{aligned} \tag{10}$$

where $q\left(c_i|v_i, s_i\right)$ is a variational distribution to approximate the posterior $p\left(c_i|v_i, s_i\right)$. The derivation of the ELBO is provided in Appendix A. Here, we formalize the variational distribution by:

$$q\left(c_i|v_i, s_i\right) = \frac{p\left(c_i|v_i\right)\tilde{p}\left(s_i|v_i, c_i\right)}{\sum_{m=1}^{M} p\left(m|v_i\right)\tilde{p}\left(s_i|v_i, m\right)} \tag{11}$$

where $\tilde{p}\left(s_i|v_i, c_i\right) = \exp\left(\mathbf{h}_i^\top \tilde{\mathbf{h}}_{s_i}/\tau\right)/\sum_{v_j\in B\setminus\{v_i\}}\exp\left(\mathbf{h}_i^\top \tilde{\mathbf{h}}_j/\tau\right)$ is calculated within a batch B. Note that both v_i and v_{s_i} are in the batch. Moreover, we apply $\tilde{p}\left(s_i|v_i, c_i\right)$ to estimate $p\left(s_i|v_i, c_i\right)$ in Eq. 10, and give the explanation in Appendix B.

We optimize the model parameters via a variation of Variational EM algorithm, where we infer $q\left(c_i|v_i, s_i\right)$ at the E-step, then optimize the ELBO at the M-step. Sampling a batch of nodes as described in Section 4.1, we maximize the following objective:

$$\mathcal{L}_{\text{ELBO}}\left(\boldsymbol{\theta}, \mathbf{w}; B\right) \approx \frac{1}{|B|}\sum_{v_i\in B}\frac{1}{|S_i|}\sum_{s_i\in S_i}\mathcal{L}_{\text{ELBO}}\left(\boldsymbol{\theta}, \mathbf{w}; v_i, s_i\right) \tag{12}$$

We observe that only using the stochastic update for the cluster prototypes can result in trivial solution, i.e., most instances are assigned to the same cluster. In order to alleviate this issue, we apply the following update after each training epoch:

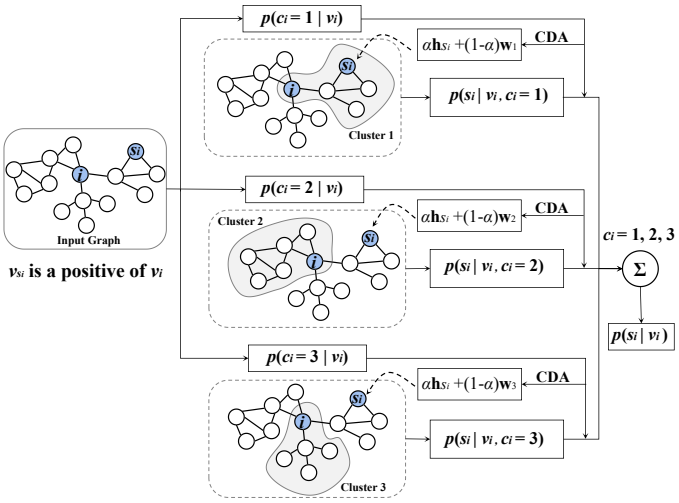


Figure 3: An overview of ClusterSCL with three latent clusters (best seen in color). ClusterSCL unifies the processes of clustering, CDA and SupCon learning.

$$\mathbf{w}_m = \frac{1}{|\overline{V}_m|}\sum_{v_i\in\overline{V}_m}\mathbf{h}_i, m = 1, 2, \cdots, M \tag{13}$$

where $\overline{V}_m$ denotes the set of nodes in the m-th graph community derived by METIS [20]. Before the training, we perform METIS to partition the whole graph G into M graph communities according to the interconnections between nodes. We use the communities to roughly describe the clusters, and average the node embeddings in each community to update the cluster prototypes after each training epoch. Note that ClusterSCL adopts Eq. 7 to derive a refined soft cluster distribution for each node. The hard cluster distributions output by METIS are only used for prototype update.

Besides, we observe that a fine-grained search on κ is needed, which is inefficient. Empirically, we use a small κ for deriving a relatively confident cluster prediction, and introduce an entropy term to smooth the predicted cluster distributions. By doing this, we can avoid the fine-grained search on κ. Finally, the ClusterSCL loss function is formalized as:

$$\mathcal{L}\left(\boldsymbol{\theta}, \mathbf{w}; B\right) = -\mathcal{L}_{\text{ELBO}}\left(\boldsymbol{\theta}, \mathbf{w}; B\right) + \frac{\eta}{|B|}\sum_{v_i\in B}\sum_{c_i=1}^{M} p\left(c_i|v_i\right)\log p\left(c_i|v_i\right) \tag{14}$$

where $\eta \in (0, 1]$ is the weight of the entropy term to control the strength of smoothing.

Algorithm 1 shows the pseudocode of training with ClusterSCL. We provide the complexity analysis of ClusterSCL in Appendix C.

4.3 Relations to Some Popular Research Topics

Deep Clustering. The most related topic is leveraging deep clustering to enhance contrastive representation learning. Existing attempts such as [3, 25, 26, 37, 45] follow the assumption that each

Algorithm 1 Pseudocode of training with ClusterSCL

Input: Graph $G = \left(V, E, X, Y^L\right)$, Number of clusters M
Output: Parameters of the optimized graph encoder g_θ and cluster prototypes $\mathbf{w} = \{\mathbf{w}_m\}_{m=1}^M$

```
Partition V into M graph communities via the METIS algorithm {V̄_m}_{m=1}^M = METIS(G);
Construct the batches {B_t}_{t=1}^T according to Y^L;
Initialize parameters θ and w;
while not MaxEpoch do
    for t = 1 to T do
        Calculate L(θ, w; B_t) with Eq. 14;
        θ, w = Adam(θ, w, L);
    end for
    H = g_θ(G);
    Update w based on H and {V̄_m}_{m=1}^M with Eq. 13;
end while
Return: θ and w
```

positive sample pair for contrastive learning share the similar cluster distributions. In contrast, due to the intra-class variances and inter-class similarities in reality, we consider the situation that each positive sample pair can have dissimilar cluster distributions. GS-TRS [1] proposed for fine-grained visual recognition also holds the same assumption about intra-class variances. Specifically, GS-TRS enforces instances of the same class within the same cluster to be closer to each other. We evaluate the idea which is similar to GS-TRS in Table 3, and the results demonstrate the superiority of ClusterSCL. The reason is that GS-TRS rigidly constrains the contrast within a cluster, so it overlooks other potentially useful positive sample pairs that locate across different clusters.

Hard Negative/Positive Mining. This work is somehow related to hard sample mining [19, 33, 36, 42, 51], which has been widely studied in contrastive representation learning. The hard negative samples are similar to the anchor, while the hard positive samples are dissimilar to the anchor. Hard samples can normally provide significant gradient information during training to benefit the model optimization. Turn to this work, intra-class variances and inter-class similarities naturally lead to hard positives and hard negatives. Despite the benefit of the hard samples, too large hardness may harm the representation learning and even result in the collapsed representations [33, 36, 42]. Thus, it is necessary to study the problem of intra-class variances and inter-class similarities.

Learning on Non-Homophilous Graphs. Most of the GNNs are designed based on the inductive bias of homophily, which assumes that the connected nodes tend to be similar in both the features and the labels. Recently, some studies [15, 27, 63] focus on a non-homophilous scenario, where the connected nodes are highly probably to be significantly dissimilar from each other in features and labels. A promising solution is to design a more powerful GNN encoder. Intra-class variances and inter-class similarities can result in some non-homophilous nodes. Contrasting nodes in a supervised way could alleviate the non-homophily issue in a global way instead of the local revision of the graph convolution operation. However, the proposed ClusterSCL still follows the homophily assumption. How to tackle more tough non-homophily issue by supervised contrastive learning will be left for further studies.

Table 1: Statistics of the used datasets.

Dataset	#Nodes	#Edges	#Features	#Classes
Cora	2,708	5,429	1,433	7
Pubmed	19,717	44,338	500	3
Citeseer	3,327	4,732	3,703	6
LastFM Asia	7,624	27,806	128	18
Amazon Computers	13,752	245,861	767	10

5 EXPERIMENTS

In this section, we conduct extensive experiments to evaluate the proposed ClusterSCL. The experiments are unfolded by answering the following research questions (RQs): (1) How does ClusterSCL perform on node classification tasks? (2) Does CDA take effect? and (3) How does ClusterSCL perform under different sizes of labeled training data?

5.1 Experimental Protocol

Datasets. We conduct experiments on the widely adopted benchmarks for node classification, where each node has a class label. Among these datasets, **Cora**, **Pubmed** and **Citeseer**[2] [23] are citation networks, where nodes are documents and edges are the citations links. **LastFM Asia**[3] [34] is a social network, where nodes represent users from Asian countries and edges are friendships between them. **Amazon Computers**[4] [38] is a co-purchase network, where nodes represent products and edges represent two products are frequently bought together. LastFM Asia and Amazon Computers are preprocessed with pytorch-geometric[5]. Statistics of these datasets are summarized in Table 1.

Baselines. We compare ClusterSCL with **Cross-entropy** and **SupCon** to evaluate the effectiveness of ClusterSCL for supervised learning of graph encoder. We also compare with **DGI** [48] and **DCI** [53], which are unsupervised loss functions for optimizing the graph encoder. DCI is proposed upon DGI for anomalous user detection. Specially, DCI considers the inconsistency between users' behavior patterns and users' label semantics, which can correspond to the intra-class variances and the inter-class similarities discussed in this paper. The main idea behind DCI is to conduct local-semi-global contrast within a more concentrated feature space, such that the inconsistency can be reduced within such a space. Similar to SupCon and ClusterSCL, DGI and DCI are performed under the two-stage training scheme.

Experimental Settings. To ensure a fair comparison between different loss functions, we keep the graph encoder and the classifier unchanged. Specifically, we set a linear layer on the top of the GNN

[2] Cora, Pubmed and Citeseer are downloaded from https://github.com/tkipf/gcn/tree/master/gcn/data.
[3] LastFM Asia is downloaded from https://graphmining.ai/datasets/ptg.
[4] Amazon Computers is downloaded from https://github.com/shchur/gnn-benchmark/raw/master/data/npz.
[5] https://pytorch-geometric.readthedocs.io/en/latest

Table 2: Overall evaluation.The bold numbers are the best performance among all two-stage models.

		Cora	Pubmed	Citeseer	LastFM	Amazon
GCN-encoder	CE (E2E)	0.804	0.789	0.696	0.731	0.831
	DGI (Two-stage, Unsup)	0.801	0.796	**0.695**	0.749	0.838
	DCI (Two-stage, Unsup)	0.811	0.793	0.694	**0.757**	**0.844**
	SupCon (Two-stage, Sup)	0.793	0.788	0.687	0.756	0.831
	ClusterSCL (Two-stage, Sup)	**0.818**	**0.805**	0.692	0.752	0.834
GAT-encoder	CE (E2E)	0.799	0.786	0.691	0.772	0.828
	DGI (Two-stage, Unsup)	0.808	0.794	0.684	0.781	0.836
	DCI (Two-stage, Unsup)	0.821	0.790	0.695	**0.784**	0.836
	SupCon (Two-stage, Sup)	0.816	0.797	0.693	0.776	0.842
	ClusterSCL (Two-stage, Sup)	**0.826**	**0.811**	**0.706**	0.779	**0.849**

encoder as the classifier. Consider the assumption of homophily, i.e., connected nodes tend to be similar in both features and labels, we adopt GCN and GAT — two classic GNN encoders that follow the homophily principle, as the backbones.

More concretely, the graph encoders contain 2 convolution layers (64 hidden units and 0.5 dropout rate for each layer). For the GAT encoder, the number of attention heads is set to be 8. We use the Adam optimizer with weight decay 1e-4 for optimization, and decay the learning rate by 0.5 with step size 50. Following the setting in the source code of SupCon, we defaultly set the temperature parameter τ to be 0.07. The batch size for SupCon and ClusterSCL is set to be 32. Each dataset contains a visible set and a test set. For Cora, Pubmed, and Citeseer, the test set has been provided [23]. For LastFM Asia and Amazon Computers, we randomly sample 50% nodes as the test set. Node labels of the visible set are available during training. We randomly sample 20 nodes per class in the visible set to form the training set. The remaining nodes in the visible set form the validation set.

The hyper-parameters we tune on the validation set include: (1) the initial learning rate $\in$ {1e-1, 1e-2, 1e-3}, (2) the number of pre-train epochs for the two-stage models which ranges from 1 to 30, (3) the number of clusters which ranges from K to $2K$, (4) $\kappa \in \{0.05, 0.07, 0.09\}$ and (5) $\eta \in \{0.05, 0.1\}$. For each dataset, we tune the initial learning rate on the end-to-end (E2E) model, then apply the same initial learning rate to the two-stage models. When using the cross-entropy loss to simultaneously optimize the graph encoder and the classifier, early stopping criterion [48] is adopted to avoid over-fitting. We set a patience of 100 and a maximum of 10,000 epochs for early stopping. The patience is reset whenever the accuracy on the validation set increases.

Code Implementation. We implement all models with Pytorch. We adopt Deep Graph Library (DGL)[6] for easy implementation of the GNN encoders. METIS is implemented using an existing Python Wrapper [7]. DGI loss, DCI loss and SupCon loss are implemented using the source code released by their authors.

[6]https://docs.dgl.ai
[7]https://pypi.org/project/PyMetis

5.2 Overall Evaluation (RQ1)

We report the classification accuracy on test sets in Table 2. Here we abbreviate LastFM Asia and Amazon Computers as LastFM and Amazon. From the experimental results, we can summarize the following conclusions. **(1) The two-stage training scheme generally performs better than the end-to-end training scheme**. This can be explained by that training the graph encoder in advance enables a good initialization of the graph encoder. **(2) ClusterSCL outperforms SupCon on most datasets**. This demonstrates the necessity of retaining intrinsic cluster distributions during SupCon learning, and the proposed ClusterSCL is effective to retain such information. **(3) Unsupervised DGI and DCI also obtain good performance**. DGI and DCI do not use any class labels when training the graph encoder. For datasets with small intra-class variances and inter-class similarities, node properties summarized by DGI or DCI can already reflect the class semantics. SupCon that simply pulls the same-labeled nodes together may reduce the data diversity. For datasets with large intra-class variances and inter-class similarities, the performance of SupCon can be suppressed. Conversely, the unsupervised methods which are independent from the class labels, can be more robust. **(4) ClusterSCL combines the advantages of DCI and SupCon.** In essence, DCI contrasts based on the clusters, while SupCon contrasts based on the classes. The competitive performance of DCI implies the necessity of modeling cluster information. ClusterSCL improves upon SupCon, meanwhile, considers the cluster information. It is worth noting that DCI performs k-means over the whole dataset after every certain number of training epochs (set to be 1 in the experiments), which limits its applications on large-scale datasets. On the contrary, ClusterSCL performs METIS once before training.

Empirically, a confident cluster prediction is often beneficial to the performance of ClusterSCL. However, if a dataset has clear cluster boundaries, preferring too confident cluster predictions may amplify the negative impacts of false cluster predictions. Compared with other losses, DCI shows more obvious advantages on Cora, indicating that nodes in Cora could have more clear cluster boundaries. So we set a larger $\eta = 1.0$ on Cora to smooth the predicted cluster distributions.

Table 3: Comparison with the Group Sensitive SupCon (GS-SupCon).

	GCN-encoder			GAT-encoder		
	SupCon	GS-SupCon	ClusterSCL	SupCon	GS-SupCon	ClusterSCL
Cora	0.793	0.788	**0.818**	0.816	0.822	**0.826**
Pubmed	0.788	0.797	**0.805**	0.797	0.801	**0.811**
Citeseer	0.687	0.684	**0.692**	0.693	0.695	**0.706**
LastFM Asia	0.756	**0.769**	0.752	0.776	0.775	**0.779**
Amazon Computers	0.831	0.833	**0.834**	0.842	0.848	**0.849**

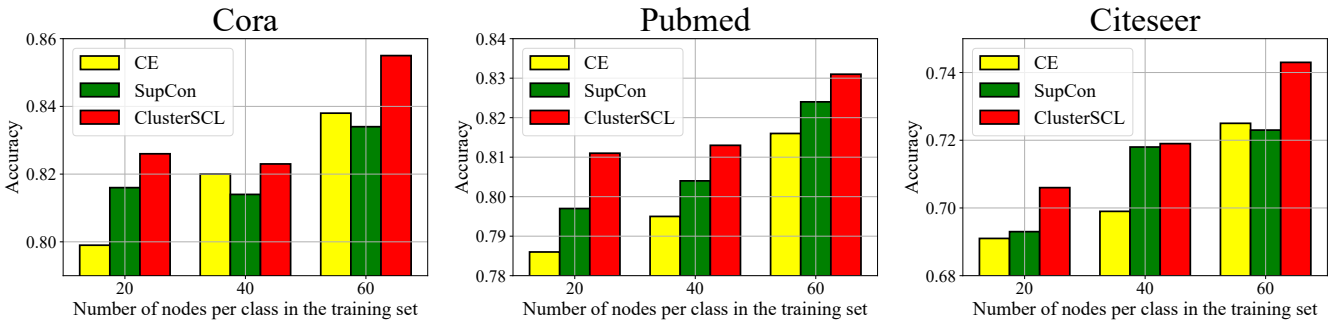


Figure 4: Study of ClusterSCL under different sizes of labeled training data.

5.3 Study of CDA (RQ2)

This subsection aims to verify the effectiveness of the cluster-aware data augmentation (CDA). To do this, we introduce a variant model called Group Sensitive SupCon (GS-SupCon) by firstly performing METIS on the graph to obtain several highly interconnected graph communities, then pulling nodes of the same class within the same graph community. GS-SupCon is similar to GS-TRS [1] proposed for fine-grained visual recognition. As reported in Table 3, GS-SupCon derives comparable or better performance compared with SupCon. This implies that modeling the intra-class variances and inter-class similarities is useful for supervised contrastive learning. For the proposed ClusterSCL, it outperforms GS-SupCon on most of the datasets. ClusterSCL considers the positive sample pairs that locate across different clusters, which are overlooked by GS-SupCon.

5.4 Study of ClusterSCL under Different Sizes of Labeled Training Data (RQ3)

In this subsection, we vary the number of nodes per class in the training set to analyze how ClusterSCL performs under different sizes of labeled training data. We set GAT-encoder as the backbone. Taking Cora, Pubmed and Citeseer as the examples, we report the experimental results in Figure 4. We can see that more labeled training data benefits the model performance. With the increase of the labeled training data, we observe that SupCon fails to outperform CE on Cora and Citeseer. The reason could be that more labeled training instances are likely to amplify the negative impacts induced by intra-class variances and inter-class similarities. ClusterSCL consistently outperforms CE and SupCon to further demonstrate its superiority.

6 CONCLUSION

This work piloted studies on supervised learning of graph neural networks for node classification. We propose a simple and effective contrastive learning scheme called **Cluster**-Aware **S**upervised **C**ontrastive **L**earning (ClusterSCL). ClusterSCL improves upon supervised contrastive (SupCon) learning and emphasizes the effectiveness of retaining intrinsic graph property during the SupCon learning, such that the negative impacts induced by intra-class variances and inter-class similarities can be reduced. ClusterSCL shows advantages over the popular cross-entropy, SupCon and other graph contrastive losses. We believe that the thought of ClusterSCL is not restricted to node classification on graphs, and could be inspiring for the research on representation learning.

ACKNOWLEDGMENTS

This work is supported by Beijing Natural Science Foundation (Grant No. 4212022), National Natural Science Foundation of China (Grant No. 62076245) and National Key Research & Develop Plan (Grant No. 2018YFB1004401). This work is partially supported by Australian Research Council Future Fellowship (Grant No. FT210100624) and Discovery Project (Grant No. DP190101985). We would like to thank the anonymous reviewers for their valuable comments.

REFERENCES

[1] Yan Bai, Feng Gao, Yihang Lou, Shiqi Wang, Tiejun Huang, and Ling-Yu Duan. 2017. Incorporating intra-class variance to fine-grained visual recognition. In *ICME*. 1452–1457.
[2] Smriti Bhagat, Graham Cormode, and S. Muthukrishnan. 2011. Node Classification in Social Networks. In *Social Network Data Analytics*. 115–148.

[3] Mathilde Caron, Ishan Misra, Julien Mairal, Priya Goyal, Piotr Bojanowski, and Armand Joulin. 2020. Unsupervised Learning of Visual Features by Contrasting Cluster Assignments. In *NeurIPS*. 9912–9924.
[4] Tianlong Chen, Yu Cheng, Zhe Gan, Jianfeng Wang, Lijuan Wang, Zhangyang Wang, and Jingjing Liu. 2021. Adversarial Feature Augmentation and Normalization for Visual Recognition. *CoRR* (2021). arXiv:2103.12171
[5] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. 2020. A Simple Framework for Contrastive Learning of Visual Representations. In *ICML*. 1597–1607.
[6] Ekin D. Cubuk, Barret Zoph, Dandelion Mané, Vijay Vasudevan, and Quoc V. Le. 2019. AutoAugment: Learning Augmentation Strategies From Data. In *CVPR*. 113–123.
[7] Ekin D. Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V. Le. 2020. Randaugment: Practical automated data augmentation with a reduced search space. In *CVPR Workshops*. 3008–3017.
[8] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative Adversarial Nets. In *NeurIPS*.
[9] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable Feature Learning for Networks. In *KDD*. 855–864.
[10] Michael Gutmann and Aapo Hyvärinen. 2010. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *AISTATS*. 297–304.
[11] William L. Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *NeurIPS*. 1024–1034.
[12] Kaveh Hassani and Amir Hosein Khasahmadi. 2020. Contrastive Multi-View Representation Learning on Graphs. In *ICML*. 4116–4126.
[13] Jun He, Hongyan Liu, Yiqing Zheng, Shu Tang, Wei He, and Xiaoyong Du. 2020. Bi-Labeled LDA: Inferring Interest Tags for Non-famous Users in Social Network. *Data Science and Engineering* 5, 1 (2020), 27–47.
[14] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. 2020. Momentum Contrast for Unsupervised Visual Representation Learning. In *CVPR*. 9729–9738.
[15] Mingguo He, Zhewei Wei, Zengfeng Huang, and Hongteng Xu. 2021. BernNet: Learning Arbitrary Graph Spectral Filters via Bernstein Approximation. In *NeurIPS*.
[16] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang, and Meng Wang. 2020. LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation. In *SIGIR*. 639–648.
[17] Daniel Ho, Eric Liang, Xi Chen, Ion Stoica, and Pieter Abbeel. 2019. Population Based Augmentation: Efficient Learning of Augmentation Policy Schedules. In *ICML*. 2731–2741.
[18] Shaoxiong Ji, Shirui Pan, Erik Cambria, Pekka Marttinen, and Philip S. Yu. 2022. A Survey on Knowledge Graphs: Representation, Acquisition, and Applications. *IEEE Transactions on Neural Networks and Learning Systems* 33, 2 (2022), 494–514.
[19] Yannis Kalantidis, Mert Bulent Sariyildiz, Noe Pion, Philippe Weinzaepfel, and Diane Larlus. 2020. Hard Negative Mixing for Contrastive Learning. In *NeurIPS*. 21798–21809.
[20] George Karypis and Vipin Kumar. 1995. Multilevel Graph Partitioning Schemes. In *ICPP*. 113–122.
[21] Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. 2020. Supervised Contrastive Learning. In *NeurIPS*. 18661–18673.
[22] Thomas N. Kipf and Max Welling. 2016. Variational Graph Auto-Encoders. In *NIPS Workshop on Bayesian Deep Learning*.
[23] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*.
[24] Kezhi Kong, Guohao Li, Mucong Ding, Zuxuan Wu, Chen Zhu, Bernard Ghanem, Gavin Taylor, and Tom Goldstein. 2020. FLAG: Adversarial Data Augmentation for Graph Neural Networks. *CoRR* (2020). arXiv:2010.09891
[25] Junnan Li, Pan Zhou, Caiming Xiong, and Steven C. H. Hoi. 2021. Prototypical Contrastive Learning of Unsupervised Representations. In *ICLR*.
[26] Yunfan Li, Peng Hu, Zitao Liu, Dezhong Peng, Joey Tianyi Zhou, and Xi Peng. 2021. Contrastive Clustering. In *AAAI*. 8547–8555.
[27] Derek Lim, Xiuyu Li, Felix Hohne, and Ser-Nam Lim. 2021. New Benchmarks for Learning on Non-Homophilous Graphs. *CoRR* (2021). arXiv:2104.01404
[28] Ziqi Liu, Chaochao Chen, Longfei Li, Jun Zhou, Xiaolong Li, Le Song, and Yuan Qi. 2019. GeniePath: Graph Neural Networks with Adaptive Receptive Paths. In *AAAI*. 4424–4431.
[29] Sebastian Nowozin, Botond Cseke, and Ryota Tomioka. 2016. f-GAN: Training Generative Neural Samplers using Variational Divergence Minimization. In *NeurIPS*.
[30] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. DeepWalk: online learning of social representations. In *KDD*. 701–710.
[31] Jiezhong Qiu, Qibin Chen, Yuxiao Dong, Jing Zhang, Hongxia Yang, Ming Ding, Kuansan Wang, and Jie Tang. 2020. GCC: Graph Contrastive Coding for Graph Neural Network Pre-Training. In *KDD*. 1150–1160.
[32] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. 2021. Learning Transferable Visual Models From Natural Language Supervision. In *ICML*. 8748–8763.
[33] Joshua Robinson, Ching-Yao Chuang, Suvrit Sra, and Stefanie Jegelka. 2021. Contrastive Learning with Hard Negative Samples. In *ICLR*.
[34] Benedek Rozemberczki and Rik Sarkar. 2020. Characteristic Functions on Graphs: Birds of a Feather, from Statistical Descriptors to Parametric Models. In *CIKM*. 1325–1334.
[35] Ruslan Salakhutdinov and Geoffrey Hinton. 2007. Learning a Nonlinear Embedding by Preserving Class Neighbourhood Structure. In *AISTATS*. 412–419.
[36] Florian Schroff, Dmitry Kalenichenko, and James Philbin. 2015. FaceNet: A unified embedding for face recognition and clustering. In *CVPR*. 815–823.
[37] Vivek Sharma, Makarand Tapaswi, M. Saquib Sarfraz, and Rainer Stiefelhagen. 2020. Clustering based Contrastive Learning for Improving Face Representations. In *FG*. 109–116.
[38] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. 2018. Pitfalls of Graph Neural Network Evaluation. *CoRR* (2018). arXiv:1811.05868
[39] Chence Shi, Minkai Xu, Hongyu Guo, Ming Zhang, and Jian Tang. 2020. A Graph to Graphs Framework for Retrosynthesis Prediction. In *ICML*. 8818–8827.
[40] Chence Shi, Minkai Xu, Zhaocheng Zhu, Weinan Zhang, Ming Zhang, and Jian Tang. 2020. GraphAF: a Flow-based Autoregressive Model for Molecular Graph Generation. In *ICLR*.
[41] Kihyuk Sohn. 2016. Improved Deep Metric Learning with Multi-class N-pair Loss Objective. In *NeurIPS*.
[42] Hyun Oh Song, Yu Xiang, Stefanie Jegelka, and Silvio Savarese. 2016. Deep Metric Learning via Lifted Structured Feature Embedding. In *CVPR*. 4004–4012.
[43] Fan-Yun Sun, Jordan Hoffmann, Vikas Verma, and Jian Tang. 2020. InfoGraph: Unsupervised and Semi-supervised Graph-Level Representation Learning via Mutual Information Maximization. In *ICLR*.
[44] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. LINE: Large-scale Information Network Embedding. In *WWW*. 1067–1077.
[45] Tsung Wei Tsai, Chongxuan Li, and Jun Zhu. 2021. MiCE: Mixture of Contrastive Experts for Unsupervised Image Clustering. In *ICLR*.
[46] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. 2018. Representation Learning with Contrastive Predictive Coding. *CoRR* (2018). arXiv:1807.03748
[47] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *ICLR*.
[48] Petar Veličković, William Fedus, William L. Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. 2019. Deep Graph Infomax. In *ICLR*.
[49] Hongwei Wang, Fuzheng Zhang, Mengdi Zhang, Jure Leskovec, Miao Zhao, Wenjie Li, and Zhongyuan Wang. 2019. Knowledge-aware Graph Neural Networks with Label Smoothness Regularization for Recommender Systems. In *KDD*. 968–977.
[50] Qinyong Wang, Hongzhi Yin, Hao Wang, Quoc Viet Hung Nguyen, Zi Huang, and Lizhen Cui. 2019. Enhancing Collaborative Filtering with Generative Augmentation. In *KDD*. 548–556.
[51] Xiaolong Wang, Abhinav Shrivastava, and Abhinav Gupta. 2017. A-Fast-RCNN: Hard Positive Generation via Adversary for Object Detection. In *CVPR*. 2606–2615.
[52] Yanbang Wang, Pan Li, Chongyang Bai, and Jure Leskovec. 2021. TEDIC: Neural Modeling of Behavioral Patterns in Dynamic Social Interaction Networks. In *WWW*. 693–705.
[53] Yanling Wang, Jing Zhang, Shasha Guo, Hongzhi Yin, Cuiping Li, and Hong Chen. 2021. Decoupling Representation Learning and Classification for GNN-based Anomaly Detection. In *SIGIR*. 1239–1248.
[54] Zhirong Wu, Alexei A. Efros, and Stella X. Yu. 2018. Improving Generalization via Scalable Neighborhood Component Analysis. In *ECCV*. 685–701.
[55] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How Powerful are Graph Neural Networks?. In *ICLR*.
[56] Jiaxuan You, Bowen Liu, Zhitao Ying, Vijay Pande, and Jure Leskovec. 2018. Graph Convolutional Policy Network for Goal-Directed Molecular Graph Generation. In *NeurIPS*.
[57] Yuning You, Tianlong Chen, Yang Shen, and Zhangyang Wang. 2021. Graph Contrastive Learning Automated. In *ICML*. 12121–12132.
[58] Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. 2020. Graph Contrastive Learning with Augmentations. In *NeurIPS*. 5812–5823.
[59] Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. 2019. CutMix: Regularization Strategy to Train Strong Classifiers With Localizable Features. In *ICCV*. 6023–6032.
[60] Feng Zhang, Jidong Zhai, Bingsheng He, Shuhao Zhang, and Wenguang Chen. 2017. Understanding Co-Running Behaviors on Integrated CPU/GPU Architectures. *IEEE Transactions on Parallel and Distributed Systems* 28, 3 (2017), 905–918.
[61] Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz. 2018. mixup: Beyond Empirical Risk Minimization. In *ICLR*.
[62] Shijie Zhang, Hongzhi Yin, Tong Chen, Quoc Viet Nguyen Hung, Zi Huang, and Lizhen Cui. 2020. GCN-Based User Representation Learning for Unifying Robust

Recommendation and Fraudster Detection. In *SIGIR*. 689–698.
[63] Jiong Zhu, Yujun Yan, Lingxiao Zhao, Mark Heimann, Leman Akoglu, and Danai Koutra. 2020. Beyond Homophily in Graph Neural Networks: Current Limitations and Effective Designs. In *NeurIPS*. 7793–7804.

A DERIVATION OF ELBO

Here we show how to obtain the ELBO in Eq. 10. Given any node v, its surrogate label s and the index of its latent cluster c, $\log p\left(s|v\right)$ can be written as:

$$\begin{aligned}\log p\left(s|v\right) &= \log \frac{p\left(v,s\right)p\left(c|v,s\right)}{p\left(v\right)p\left(c|v,s\right)} \\ &= \log \frac{p\left(v,s,c\right)}{p\left(v\right)p\left(c|v,s\right)} \\ &= \log \frac{p\left(s|v,c\right)p\left(v,c\right)}{p\left(v\right)p\left(c|v,s\right)} \\ &= \log \frac{p\left(s|v,c\right)p\left(c|v\right)}{p\left(c|v,s\right)} \\ &= \log p\left(s|v,c\right) - \log p\left(c|v,s\right) + \log p\left(c|v\right) \\ &= \log p\left(s|v,c\right) - \log \frac{p\left(c|v,s\right)}{q\left(c|v,s\right)} + \log \frac{p\left(c|v\right)}{q\left(c|v,s\right)}\end{aligned} \tag{15}$$

We integrate both sides of the above equation.
For the left-hand side,

$$\begin{aligned}\int q\left(c|v,s\right)\log p\left(s|v\right)dc &= \log p\left(s|v\right)\int q\left(c|v,s\right)dc \\ &= \log p\left(s|v\right)\end{aligned} \tag{16}$$

For the right-hand side,

$$\begin{aligned}&\int q\left(c|v,s\right)\log p\left(s|v,c\right)dc - \int q\left(c|v,s\right)\log\frac{p\left(c|v,s\right)}{q\left(c|v,s\right)}dc \\ &+\int q\left(c|v,s\right)\log\frac{p\left(c|v\right)}{q\left(c|v,s\right)}dc \\ &= \mathbb{E}_{q(c|v,s)}\left[\log p\left(s|v,c\right)\right] + \mathrm{KL}\left(q\left(c|v,s\right)||p\left(c|v,s\right)\right) \\ &\quad - \mathrm{KL}\left(q\left(c|v,s\right)||p\left(c|v\right)\right) \\ &\geq \mathbb{E}_{q(c|v,s)}\left[\log p\left(s|v,c\right)\right] - \mathrm{KL}\left(q\left(c|v,s\right)||p\left(c|v\right)\right)\end{aligned} \tag{17}$$

where the inequality holds due to $\mathrm{KL}\left(q\left(c|v,s\right)||p\left(c|v,s\right)\right) \geq 0$. Thus, we can derive the ELBO:

$$\log p\left(s|v\right) \geq \mathbb{E}_{q(c|v,s)}\left[\log p\left(s|v,c\right)\right] - \mathrm{KL}\left(q\left(c|v,s\right)||p\left(c|v\right)\right) \tag{18}$$

B WHY USE THE BATCH-BASED DISCRIMINATOR

Strictly, ClusterSCL predicts the surrogate label s_i of the given node v_i via Eq. 6, which is prohibitive for large-scale datasets. Thus, we provide an estimator to efficiently calculate within a batch,

$$\tilde{p}\left(s_i|v_i,c_i\right) = \frac{\exp\left(\mathbf{h}_i^\intercal\tilde{\mathbf{h}}_{s_i}/\tau\right)}{\sum_{v_j\in B\setminus\{v_i\}}\exp\left(\mathbf{h}_i^\intercal\tilde{\mathbf{h}}_j/\tau\right)} \tag{19}$$

The main objective of ClusterSCL is to maximize the ELBO $\mathcal{L}_{\mathrm{ELBO}}\left(\boldsymbol{\theta},\mathbf{w};v_i,s_i\right)$. For any given variational distribution $q\left(c_i|v_i,s_i\right)$, the ELBO in Eq. 10 can be written as:

$$\begin{aligned}&\mathcal{L}_{\mathrm{ELBO}}\left(\boldsymbol{\theta},\mathbf{w};v_i,s_i\right) \\ &= \mathbb{E}_{q(c_i|v_i,s_i)}\left[\log p\left(s_i|v_i,c_i\right)\right] - \mathrm{KL}\left(q\left(c_i|v_i,s_i\right)||p\left(c_i|v_i\right)\right) \\ &= \mathbb{E}_{q(c_i|v_i,s_i)}\log\frac{\exp\left(\mathbf{h}_i^\intercal\tilde{\mathbf{h}}_{s_i}/\tau\right)}{\sum_{v_j\in V\setminus\{v_i\}}\exp\left(\mathbf{h}_i^\intercal\tilde{\mathbf{h}}_j/\tau\right)} - \mathrm{KL}\left(q\left(c_i|v_i,s_i\right)||p\left(c_i|v_i\right)\right) \\ &= \mathbb{E}_{q(c_i|v_i,s_i)}\log\frac{\exp\left(\mathbf{h}_i^\intercal\tilde{\mathbf{h}}_{s_i}/\tau\right)}{\sum_{v_j\in B\setminus\{v_i\}}\exp\left(\mathbf{h}_i^\intercal\tilde{\mathbf{h}}_j/\tau\right)} - \mathrm{KL}\left(q\left(c_i|v_i,s_i\right)||p\left(c_i|v_i\right)\right) \\ &+ \mathbb{E}_{q(c_i|v_i,s_i)}\log\frac{\sum_{v_j\in B\setminus\{v_i\}}\exp\left(\mathbf{h}_i^\intercal\tilde{\mathbf{h}}_j/\tau\right)}{\sum_{v_j\in V\setminus\{v_i\}}\exp\left(\mathbf{h}_i^\intercal\tilde{\mathbf{h}}_j/\tau\right)} \\ &= \tilde{\mathcal{L}}_{\mathrm{ELBO}}\left(\boldsymbol{\theta},\mathbf{w};v_i,s_i\right) + \mathbb{E}_{q(c_i|v_i,s_i)}\log\frac{\sum_{v_j\in B\setminus\{v_i\}}\exp\left(\mathbf{h}_i^\intercal\tilde{\mathbf{h}}_j/\tau\right)}{\sum_{v_j\in V\setminus\{v_i\}}\exp\left(\mathbf{h}_i^\intercal\tilde{\mathbf{h}}_j/\tau\right)}\end{aligned} \tag{20}$$

where $\tilde{\mathcal{L}}_{\mathrm{ELBO}}\left(\boldsymbol{\theta},\mathbf{w};v_i,s_i\right)$ adopts the batch-based estimator of $p\left(s_i|v_i,c_i\right)$.

Since $\mathbf{h}$ are ℓ_2-normalized node representations, if we update the cluster prototypes via Eq. 13 before each batch training, we can derive that:

$$\sum_{v_j\in B\setminus\{v_i\}}\exp\left(\mathbf{h}_i^\intercal\tilde{\mathbf{h}}_j/\tau\right) \geq \left(|B|-1\right)\exp\left(\frac{-1}{\tau}\right) \tag{21}$$

$$\sum_{v_j\in V\setminus\{v_i\}}\exp\left(\mathbf{h}_i^\intercal\tilde{\mathbf{h}}_j/\tau\right) \leq \left(|V|-1\right)\exp\left(\frac{1}{\tau}\right) \tag{22}$$

Since they are both positive, we can derive that:

$$\log\frac{\sum_{v_j\in B\setminus\{v_i\}}\exp\left(\mathbf{h}_i^\intercal\tilde{\mathbf{h}}_j/\tau\right)}{\sum_{v_j\in V\setminus\{v_i\}}\exp\left(\mathbf{h}_i^\intercal\tilde{\mathbf{h}}_j/\tau\right)} \geq \log\frac{|B|-1}{|V|-1} - \frac{2}{\tau} \tag{23}$$

We can further derive a bound of the ELBO:

$$\mathcal{L}_{\mathrm{ELBO}}\left(\boldsymbol{\theta},\mathbf{w};v_i,s_i\right) \geq \tilde{\mathcal{L}}_{\mathrm{ELBO}}\left(\boldsymbol{\theta},\mathbf{w};v_i,s_i\right) + \log\frac{|B|-1}{|V|-1} - \frac{2}{\tau} \tag{24}$$

where $\log\frac{|B|-1}{|V|-1}-\frac{2}{\tau}$ is a constant. So maximizing $\tilde{\mathcal{L}}_{\mathrm{ELBO}}\left(\boldsymbol{\theta},\mathbf{w};v_i,s_i\right)$ indirectly maximizes $\mathcal{L}_{\mathrm{ELBO}}\left(\boldsymbol{\theta},\mathbf{w};v_i,s_i\right)$ to some extent.

In practice, we update the cluster prototypes via Eq. 13 after each training epoch for training efficiency, and can obtain good experimental results.

C COMPLEXITY ANALYSIS

Here we provide the complexity analysis of ClusterSCL. Given the dimension of node embeddings d, the batch size $2N$, the number of clusters M, ClusterSCL's time complexity is $O\left(N^2(d+M)+NMd\right)$. For reference, the time complexity of SupCon is $O\left(N^2d\right)$. Notice that d and M are relatively small, the time complexity of ClusterSCL can be approximate to that of SupCon. Additionally, since the GNN encoders have much fewer parameters than encoders used in CV and NLP, the batch size is not required to be too large.