CuCo: Graph Representation with Curriculum Contrastive Learning

Guanyi Chu, **Xiao Wang**, **Chuan Shi**[*] and **Xunqiang Jiang**
Beijing University of Posts and Telecommunications
{cgy463, xiaowang, shichuan, skd621}@bupt.edu.cn

Abstract

Graph-level representation learning is to learn low-dimensional representation for the entire graph, which has shown a large impact on real-world applications. Recently, limited by expensive labeled data, contrastive learning based graph-level representation learning attracts considerable attention. However, these methods mainly focus on graph augmentation for positive samples, while the effect of negative samples is less explored. In this paper, we study the impact of negative samples on learning graph-level representations, and a novel curriculum contrastive learning framework for self-supervised graph-level representation, called CuCo, is proposed. Specifically, we introduce four graph augmentation techniques to obtain the positive and negative samples, and utilize graph neural networks to learn their representations. Then a scoring function is proposed to sort the negative samples from easy to hard, and a pacing function is to automatically select the negative samples in each training procedure. Extensive experiments on fifteen graph classification real-world datasets, as well as the parameter analysis, well demonstrate that our proposed CuCo yields truly encouraging results in terms of performance on classification and convergence.

1 Introduction

Graph-structured data, such as social networks, protein-protein interactions and financial networks, are ubiquitous in real-world scenes. Recently, it is well recognized that an adequate representation of graphs is vital to the learning performance of a statistical or machine learning model, and therefore, much work has been done on learning the low-dimensional [Cai *et al.*, 2018]. To date, graph representation learning has achieved remarkable success in many domains such as social network analysis and recommendation system [Fouss *et al.*, 2007].

Most of the current graph representation learning methods aim at learning node-level representation in a single graph. However, in practice, a variety of applications need to extract the representation of entire graphs such as predicting molecule properties in drug and material discovery [Gilmer *et al.*, 2017], and forecasting protein function in biological networks [Jiang *et al.*, 2017]. Therefore, graph-level representation learning has also attracted a large amount of attention in recent years. These methods design a message passing algorithm and aggregation procedure to get the node embedding, and then use pooling mechanisms [Ying *et al.*, 2018] to obtain a representation of the entire graph in a supervised way.

Nevertheless, in many real applications, labeled data are very limited and expensive to obtain. For example, in the chemical domain, labels are typically produced with a costly density functional theory calculation. As a consequence, it is becoming increasingly important to learn the representations of entire graphs in an unsupervised or self-supervised fashion. Recently, motivated by profound success in natural language processing [Devlin *et al.*, 2018] and computer vision [He *et al.*, 2020], self-supervised contrastive leaning based graph-level representation learning attracts considerable attention. To be specific, the basic idea of these methods is to enforce the embeddings of views augmented from the same instance (forming the positive pair) close to each other, while those from different instances (negatives) apart using a contrastive loss, leading to powerful and transferable representations.

When applying contrastive learning to graphs, two fundamental problems need to be carefully considered: one is how to augment positive pairs on graphs and the other is how to select and train negative samples effectively. These two factors both essentially determine the success of contrastive learning on graphs. Most of current works focus on the former issue and design different graph augmentation strategies, such as substructure augmentation [Veličković *et al.*, 2018; Sun *et al.*, 2019; Weihua *et al.*, 2020a; You *et al.*, 2020a]. However, little efforts have been made towards an effective negative sample selection and training strategies, while in fact treating negative samples effectively is rather crucial on the generalization performance of the contrastive learning [Robinson *et al.*, 2020; Kalantidis *et al.*, 2020]. This requirement is quite challenging because in contrastive leaning setting the label information is unknown, making it infeasible to adopt existing negative sampling strategies that use label information. At present, most of contrastive methods select

[*]Corresponding author

negative samples randomly in the training process. Nevertheless, different negative samples play different roles and have different influence. Specifically, for a specific sample, the difficulties of its negative samples is different, and these negative samples play distinct roles in different training periods. For instance, harder negative samples play a major role in the later stage of training [Kalantidis *et al.*, 2020]. Therefore, we need to consider the order of sampling negatives based on their difficulty in the training process, especially in the case of no label information, the problem becomes more challenging.

In this paper, we attempt to tackle the above challenging problem by proposing a **Cu**rriculum **Co**ntrastive learning framework for graph-level representation, called **CuCo**. In particular, we introduce four types of graph augmentations for constructing positive sample pairs firstly and use a scoring function to measure the difficulty of negative samples in the training dataset. According to the learning process of human beings, we should start with easy samples when learning a new model and then learn difficult samples gradually [Bengio *et al.*, 2009], so we sort the negative samples from easy to hard. In addition, we design a pacing function to schedule how the negative samples are introduced to the training procedure. At last, we utilize a popular contrastive object function to optimize our model. In this way, compared with state-of-the-art self-supervised graph-level representation learning methods, our method not only achieves overall better performance, but also reduces training time.

We summarize the contributions as follows:

- We make the first attempt towards studying the impact of negative samples on learning graph-level representation, which is largely ignored by previous works but rather practical and important for a good self-supervised graph-level representation learning.
- We propose a novel curriculum contrastive learning based graph representation learning model. Our model effectively combines curriculum learning and contrastive learning, which is able to automatically select and train negative samples in a human-learning manner.
- We conduct comprehensive experiments, which show that the proposed method improves downstream convergence speed and performance on fifteen datasets.

2 Related Work

Graph Neural Networks. In recent years, Graph Neural Networks (GNNs) [Kipf and Welling, 2016; Xu *et al.*, 2018] have emerged as a promising approach for analyzing graph-structured data. They follow an iterative neighborhood aggregation (or message passing) scheme, where each node aggregates feature vectors of its neighbors to compute its new feature vector [Gilmer *et al.*, 2017]. After k iterations of aggregation, a node is represented by its transformed feature vector, which captures the structural information within the node's k-hop neighborhood. The representation of an entire graph can then be obtained through pooling, for example, by meaning the representation vectors of all nodes in the graph. Unlike these methods which are mainly trained in a supervised fashion, our approach aims for unsupervised/self-supervised learning for GNNs.

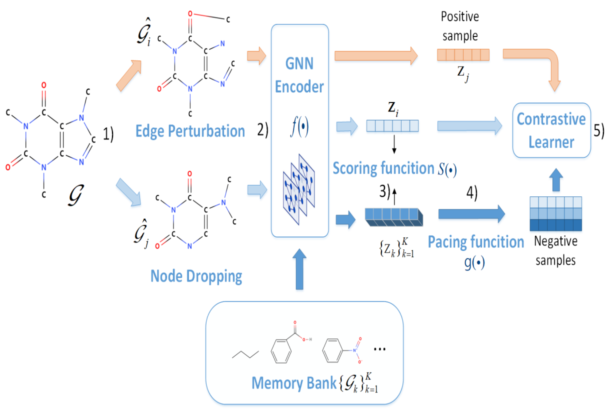

Figure 1: Overall framework of CuCo. 1) Get positive pairs by graph augmentations. 2) Get entire graph embeddings for each graph. 3) Sort the negative samples by scoring function. 4) Use pacing function to specify the size of the memory bank used at each step. 5) Optimize objective function with gradient descent.

Contrastive Learning. Contrastive learning is one of the state-of-the-art self-supervised representation learning algorithms that achieves great success for natural language processing [Devlin *et al.*, 2018] and visual representation learning [He *et al.*, 2020]. Contrastive learning force the embeddings of views generated from the same data instance to be closer to each other, while those from different instances apart. In this work, we adopt the noise-contrastive estimation from [Oord *et al.*, 2018], as discussed in Section 3.

Curriculum Learning. Curriculum learning [Bengio *et al.*, 2009] is a training strategy that trains a machine learning model from easier samples to harder samples, which imitates the meaningful learning order in human curricula. Previous empirical studies have shown that curriculum learning strategy can improve the generalization capacity and convergence speed of various models in a wide range of scenarios such as computer vision [Pentina *et al.*, 2015] and natural language processing [Guo *et al.*, 2018], etc. These methods are mainly developed for images or words but not for graph-structured data.

3 Our Proposed Model: CuCo

The main goal of this paper is to train a GNN encoder with a curriculum learning based negative sampling strategy. As shown in Figure 1, given a graph $\mathcal{G}$, we employ the graph augmentation techniques to obtain the positive pairs $\hat{\mathcal{G}}_i, \hat{\mathcal{G}}_j$ at first. Here we use every other graph except $\mathcal{G}$ in the training set as negative samples $\{\mathcal{G}_k\}_{k=1}^K$, called memory bank. Then we utilize graph neural networks to learn their representations $z_i, z_j, \{z_k\}_{k=1}^K$ respectively. Thereafter we use a scoring function $S(\cdot)$ to measure the difficulty of negative samples and sort the negative samples from easy to hard. Next a pacing function $g(\cdot)$ is to schedule how the negative samples are introduced to the training procedure. Finally, the parameters of GNN are optimized with gradient descent using the contrastive learner.

Problem Definition. Given an undirected graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ where $\mathcal{V} = \{v_i\}_{i=1}^{|\mathcal{V}|}$ denotes a set of $|\mathcal{V}|$ nodes and $\mathcal{E} = [e_{ij}] \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}|}$ denotes the adjacency matrix, with the node feature matrix $\boldsymbol{X} \in \mathbb{R}^{|\mathcal{V}| \times N}$ where $\mathbf{x}_n = \boldsymbol{X}[n,:]^T$ is the N-dimensional attribute vector of the node $v_n \in \mathcal{V}$, for self-supervised graph representation learning, a set of unlabeled graphs $\mathbb{G} = \{\mathcal{G}_1, \mathcal{G}_2, \cdots, \mathcal{G}_M\}$ are given, and we aim to learn a d-dimensional vector $z_{\mathcal{G}_i} \in \mathbb{R}^d$ for each graph $\mathcal{G}_i \in \mathbb{G}$ under the guidance of the data itself.

3.1 Graph Augmentations

We know that, without any data augmentation, contrastive learning is not helpful and often worse compared with training from scratch. Data augmentation aims at creating novel and realistically rational data through applying certain transformation without affecting the semantics label. We focus on graph-level augmentations. Specifically, given a graph $\mathcal{G} \in \{\mathcal{G}_m : m \in M\}$ in the dataset of M graphs, we formulate the augmented graph $\hat{\mathcal{G}}$ satisfying: $\hat{\mathcal{G}} \sim \mathcal{T}(\hat{\mathcal{G}}|\mathcal{G})$, where $\mathcal{T}(\cdot|\mathcal{G})$ is the augmentation distribution conditioned on the original graph, which is pre-defined, representing the human prior for data distribution. Usually, there are four basic data augmentation strategies to construct positive pairs of graphs [You *et al.*, 2020a] : (1) node dropping that randomly discards certain portion of nodes along with their connections; (2) edge perturbation that perturbs the connectivities in $\mathcal{G}$ through randomly adding or dropping certain ratio of edges; (3) attribute masking that prompts models to recover masked node attributes using their context information, i.e., the remaining attributes; and (4) subgraph that samples a subgraph from $\mathcal{G}$ using random walk.

The given graph $\mathcal{G}$ undergoes graph data augmentations to obtain two correlated views $\hat{\mathcal{G}}_i$, $\hat{\mathcal{G}}_j$, as a positive pair, where $\hat{\mathcal{G}}_i$, $\hat{\mathcal{G}}_j \sim \mathcal{T}(\cdot|\mathcal{G})$ independently. Furthermore, although any other graph in the training set can be regarded as a negative pair with $\mathcal{G}$, most of contrastive methods simply select negative samples randomly. However, how to strategically select negative samples matters, we will discuss this in Sec. 3.4. For different domains of graph datasets, we adopt [You *et al.*, 2020a] to strategically select data augmentations. Specifically, for chemical and biochemical molecules, we employ two augmentations strategies as: node dropping and subgraph; for social networks, we use all four data augmentation strategies.

3.2 Graph Encoder

After we obtain the positive augmented graph pairs, i.e., $\hat{\mathcal{G}}_i$ and $\hat{\mathcal{G}}_j$, we need to learn their representations. Although our method allows various choices of the network architecture without any constraint, here we employ graph neural networks (GNNs) as the graph encoder. Next we take $\hat{\mathcal{G}}_i$ as an example to introduce the GNN based learning process, and this procedure is the same as $\hat{\mathcal{G}}_j$. Given an undirected graph $\hat{\mathcal{G}}_i = \{\mathcal{V}_i, \mathcal{E}_i\}$ with the node feature matrix $\boldsymbol{X} \in \mathbb{R}^{|\mathcal{V}_i| \times N}$ where $\mathbf{x}_n = \boldsymbol{X}[n,:]^T$ is the N-dimensional attribute vector of the node $v_n \in \mathcal{V}_i$. Considering a L-layer GNN $f(\cdot)$, the propagation of the lth layer is represented as:

$$\mathbf{a}_n^{(l)} = \text{AGGREGATION}^{(l)}(\{\mathbf{h}_{n'}^{(l-1)} : n' \in \mathcal{N}(n)\}),$$
$$\mathbf{h}_n^{(l)} = \text{COMBINE}^{l}(\mathbf{h}_n^{(l-1)}, \mathbf{a}_n^{(l)}), \tag{1}$$

where $\mathbf{h}_n^{(l)}$ is the embedding of the node v_n at the lth layer with $\mathbf{h}_n^{(0)} = \mathbf{x}_n$, $\mathcal{N}(n)$ is a set of nodes adjacent to v_n, and $\text{AGGREGATION}^{(l)}(\cdot)$ and $\text{COMBINE}^{(l)}(\cdot)$ are component functions of the GNN layer. After the L-layer propagation, the output embedding $f(\hat{\mathcal{G}}_i)$ for $\hat{\mathcal{G}}_i$ is summarized on layer embeddings through the READOUT function. Then a multilayer perceptron (MLP) is adopted for the graph-level downstream task (classification or regression):

$$f(\hat{\mathcal{G}}_i) = \text{READOUT}(\{\mathbf{h}_{n'}^{(l-1)} : v_n \in \mathcal{V}_i, l \in L\}),$$
$$z_i = \text{MLP}(f(\hat{\mathcal{G}}_i)). \tag{2}$$

3.3 Contrastive Learning with Memory

To enforce maximizing the consistency between positive pairs $\{z_i,\ z_j\}$ compared with negative pairs, we adopt the noise-contrastive estimation loss [Oord *et al.*, 2018]. Specifically, we define a "memory bank" Q which contains K negative sample embeddings $\{z_k\}_{k=1}^{K}$ for each positive pair $\{z_i,\ z_j\}$, then we employ a similarity metric function $sim(\cdot,\cdot)$ to calculate the similarity of positive pair $\{z_i, z_j\}$ and the negative pair $\{z_i, z_k\}$. Based on this, the loss function is as follows:

$$\mathcal{L}_{\text{NCE}} = -\log \frac{\exp(sim(z_i, z_j)/\tau)}{\exp(sim(z_i, z_j)/\tau) + \sum_{k=1}^{K} \exp(sim(z_i, z_k)/\tau)}, \tag{3}$$

where τ denotes the temperature parameter. Minimizing Eq.(3) implies that we enforce the positive pair score higher than the negative pairs in the memory bank.

To simplify the calculation, we use dot product as the similarity metric function. The log-likelihood function of Eq.(3) is defined over the probability distribution created by applying a softmax function for each z_i. Let $p_{\hat{z}}$ be the matching probability for the z_i and the other samples $\hat{z} \in Z = Q \cup \{z_j\}$, then the gradient of the loss with respect to the z_i is given by

$$\frac{\partial \mathcal{L}_{\text{NCE}}}{\partial z_i} = -\frac{1}{\tau}\left((1 - p_{z_j}) \cdot z_j - \sum_{z_k \in Q} p_{z_k} \cdot z_k\right),$$

where

$$p_{\hat{z}} = \frac{\exp({z_i}^T \hat{z}/\tau)}{\sum_{m \in Z} \exp({z_i}^T z_m/\tau)}, \tag{4}$$

and p_{z_j}, p_{z_k} are the matching probability of the positive pair $\hat{z} = z_j$ and negative pair $\hat{z} = z_k$ respectively.

3.4 Curriculum Setting for Negative Sampling

Obviously, we need negative sampling from our memory bank. In the following, we will introduce a novel negative sampling method by adopting curriculum learning. The main idea is ordering negative samples during training based on their difficulty. Generally speaking, a curriculum is defined by specifying three ingredients [Hacohen and Weinshall, 2019]: the scoring function, the pacing function and the order.

The Scoring Function

Here, in our memory bank, we have K negative embeddings $\{z_k\}$ with different diffuculties. We then define a scoring function, which maps a negative embedding z_k to a numerical score $S(z_k)$, to measure such difficulty. Usually, higher score corresponds to a more difficult negative embedding. We can set up the scoring function $S(z_k)$ as $sim(z_i, z_k) \in \mathbb{R}$ to return the measure of a negative embedding's difficulty, since the higher the scoring function value is, the loss for z_i also becomes higher according to Eq. (3), meaning that z_k is a more difficult negative embedding. Specially, we use two common score functions based on embedding's similarity:

- **Cosine Similarity** : It uses the cosine value of the angle between two vectors to measure the similarity.

$$S(z_k) = \frac{|z_k \cdot z_i|}{|z_k||z_i|}. \tag{5}$$

- **Dot Product Similarity** : In contrast to the cosine similarity, the dot product is proportional to the vector length.

$$S(z_k) = z_k \cdot z_i. \tag{6}$$

The Pacing Function

After we get the score $S(z_k)$ of every single negative embedding z_k in the memory bank, we use pacing function to schedule how the negative samples are introduced to the training procedure. The pacing function $g(t)$ specifies the size of the memory bank used at each step t. The memory bank at step t consists of the $g(t)$ lowest scored samples. Negative sample batches are sampled uniformly from this set. We denote the full memory bank size by K and the total number of training steps by T. Here we consider four common function families: logarithmic, linear , quadratic, and root.

- **Logarithmic** :

$$g(t) = [1 + .1\log(\frac{t}{T} + e^{-10})] \cdot K. \tag{7}$$

- **Polynomial** :

$$g(t) = (t/T)^{\lambda} \cdot K. \tag{8}$$

where λ is a smoothing parameter which controls the pace in the training procedure. $\lambda = 1/2, 1, 2$ denote root, linear and quadratic pacing functions, respectively.

The Order

In order to narrow down the specific effects of using scoring function based on ascending difficulty level, we specify an order of either curriculum (ordering examples from the lowest score to the highest score), anti-curriculum (ordering examples from the highest score to the lowest), or random.

3.5 Early Stop Mechanism

In the later stage of training, with the increasing difficulty of negative samples, the proportion of false negative samples which have the same label with the z_i will increase, and false negative samples will hurt the generalization performance of the model [Kalantidis *et al.*, 2020]. To relieve this problem, we design an early stop mechanism. Specially, we define a hyperparameter patience of early stop p. When loss no longer decreases, p value begins to decreased. Once p becomes 0, the training will stop.

Algorithm 1 Training procedure of CuCo

```
Input: Training set D = $\{\mathcal{G}_j\}_{j=1}^{N_D}$ , the number of training iterations $N_T$, difficulty scoring function $S$, pacing function $g$, a GNN model $f$, patience of early stop $p$, the augmentation distribution $\mathcal{T}$
Output: The pre-trained GNN $f_\theta$
  Initialize model parameters $\theta$ with an Xavier initialization.
  Let $t = 1$.
  while $p \geq 0$ and $t \leq N_T$ do
    $B_{\mathcal{G}} \leftarrow \text{RandomSample}(D)$
    $B_{\hat{\mathcal{G}}} \sim \mathcal{T}(B_{\hat{\mathcal{G}}}|B_{\mathcal{G}})$
    $B'_{\hat{\mathcal{G}}} \sim \mathcal{T}(B'_{\hat{\mathcal{G}}}|B_{\mathcal{G}})$
    $z_{\hat{\mathcal{G}}_j}, z_{\hat{\mathcal{G}}'_j} \leftarrow$ Eqs. (1,2)   $(j = 1, 2, \cdots, N)$
    $z_{\mathcal{G}_i} \leftarrow$ Eqs. (1,2)   $(i = 1, 2, \cdots, N_D - 1)$
    for $j = 1$ to $N$ do
      sort $z_{\mathcal{G}_i}$ according to $S(z_{\hat{\mathcal{G}}_j}, z_{\mathcal{G}_i})$ in ascending order
      $z_{\mathcal{G}'_i} \leftarrow z_{\mathcal{G}}[1, \ldots, g(t)]$
      $z_{\mathcal{G}_k} \leftarrow \text{UniformlySample}(z_{\mathcal{G}'_i})$
    end for
    $\mathcal{L}_{\text{NCE}} \leftarrow$ Eq (3)
    $f_\theta \leftarrow f_\theta - \nabla_{f_\theta}(\mathcal{L}_{\text{NCE}})$
    if $\mathcal{L}_{\text{NCE}}$ not descend then
      $p = p - 1$
    end if
  end while
```

Datasets	Category	Graph Num.	Avg.Node	Avg.Edges
MUTAG	Molecules	188	17.93	19.79
NCI1	Molecules	4110	29.87	32.30
PROTEINS	Molecules	1113	39.06	72.82
DD	Molecules	1178	284.32	715.66
COLLAB	Social Networks	5000	74.49	2457.78
RDT-B	Social Networks	4999	508.52	594.87
RDT-M5K	Social Networks	2000	429.63	497.75

Table 1: Statistics of datasets

3.6 Model Optimization

The training procedure of CuCo is summarized in Algorithm 1. During the training, using the online-updated graph embeddings, we sort the negative embeddings by their difficulty according to score function $S(\cdot)$, and then sample the negative pairs according to pacing function $g(\cdot)$. In each iteration, the parameters of GNN are optimized with gradient descent using the objective $\mathcal{L}_{NCE}$. If the objective no longer decreases, the patience value p begins to reduce 1 every iteration. Once the patience value p becomes 0 or epoch t achieves the last iteration N_T, the training will stop.

4 Experiments

In this section, in order to indicate that our self-supervised model, CuCo, can learn graph representation fast and well, we compare our model with state-of-the-art methods (SOTAs) in the settings of unsupervised and transfer learning on graph classification.

Dataset	NCI1	PROTEINS	DD	MUTAG	RDT-B	RDT-M5K	COLLAB
DGK	**80.31** ± 0.46	73.30 ± 0.82	74.85 ± 0.74	87.44 ± 2.72	78.04 ± 0.39	41.27 ± 0.18	64.66 ± 0.50
WL	80.01 ± 0.50	72.92 ± 0.56	74.02 ± 2.28	80.72 ± 3.00	68.82 ± 0.41	46.06 ± 0.21	69.30 ± 3.44
sub2vec	52.84 ± 1.47	53.03 ± 5.55	54.33 ± 2.44	61.05 ± 15.80	71.48 ± 0.41	36.68 ± 0.42	55.26 ± 1.54
graph2vec	73.22 ± 1.81	73.30 ± 2.05	70.32 ± 2.32	83.15 ± 9.25	75.78 ± 1.03	46.86 ± 0.26	71.10 ± 0.54
InfoGraph	76.20 ± 1.06	74.44 ± 0.31	72.85 ± 1.78	89.01 ± 1.13	82.50 ± 1.42	53.46 ± 1.03	70.65 ± 1.13
GraphCL	77.87 ± 0.41	74.39 ± 0.45	78.62 ± 0.40	86.80 ± 1.34	87.53 ± 0.84	55.99 ± 0.28	71.36 ± 0.44
CuCo	79.24 ± 0.56	**75.91** ± 0.55	**79.20** ± 1.12	**90.55** ± 0.98	**88.6** ± 0.55	**56.49** ± 0.19	**72.30** ± 0.34

Table 2: The performance of different methods on graph classification.

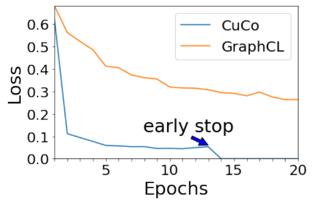

Figure 2: Training loss versus epochs.

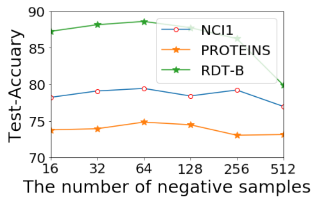

Figure 3: Accuracy versus negative sample size.

	Logarithmic	Root	Linear	Quadratic
MUTAG	90.33	**90.76**	90.55	88.28
NCI1	77.45	78.15	**78.95**	78.19
PROTEINS	75.48	74.48	**75.91**	74.57
DD	77.96	78.12	**79.22**	78.33
RDT-B	87.78	87.55	**88.61**	86.16

Table 3: Employing alternative pacing functions on five datasets

4.1 Unsupervised Representation Learning

We evaluate our method in the unsupervised representation learning following [Sun *et al.*, 2019], where CuCo is used to learn graph representations in a fully unsupervised manner, followed by evaluating the graph-level classification utility of these representations. This is performed by directly using these representations to train and test a SVM classifier.

Experiments Setting

We evaluate model performance on seven classical graph classification benchmarks shown in Table 1. Aside from SOTA graph kernel methods that Weisfeiler-Lehman subtree kernel (WL) [Shervashidze *et al.*, 2011] and deep graph kernel (DGK) [Yanardag and Vishwanathan, 2015], we also compare with four state-of-the-art unsupervised graph-level representation learning methods as sub2vec [Adhikari *et al.*, 2018], graph2vec [Narayanan *et al.*, 2017], InfoGraph [Sun *et al.*, 2019] and GraphCL[You *et al.*, 2020a]. For our proposed model, we adopt a three-layer Graph Isomorphism Network (GIN) with 32-dimensional hidden units and a sum pooling readout function for performance comparisons. We use 10-fold cross validation accuracy to report the classification performance. Experiments are repeated 5 times. We report results from previous papers with the same experimental setup if available. If results are not previously reported, we implement them and conduct a hyperparameter search according to the original paper.

Performance Analysis

The results of evaluating unsupervised graph level representations using downstream graph classification tasks are presented in Table 2. While two kernel methods perform well on individual datasets, none of them are competitive across all of the datasets. We find that CuCo outperforms all of these baselines on 6 out of 7 of the datasets. In the other dataset NCI1, CuCo still has very competitive performance.

We also investigate the benefits of convergence speed. We compare CuCo with GraphCL for fairness, since our method and GraphCL employ the same graph augmentations and GNN architecture. Due to our early stop mechanism and ordered learning, as shown in Figure 2, our method leads to faster training: our negative sampling only needs 13 to 14 epochs to reach the better performance than GraphCL in 20 epochs, which reduces time cost by about 40%.

Alternative Pacing Functions

In our empirical setup, we investigate all four pacing functions on five datasets, and the results are shown in Table 3. We find that the linear pacing function has a slight advantage on most datasets and the other pacing functions have comparable performance. For the logarithmic and root which are concave functions, the memory bank we used will contain more diffculty negative samples in a fewer iteration, resulting in the effect of curriculum learning is similar to random sampling. For the quadratic which is a convex function, the number of hard negative samples will rise rapidly in the later stage of training. It leads to a rapid increase in the number of false negative samples, which will be harmful for the model. Therefore, we think that increasing the difficulty of introducing negative samples at a uniform rate have a positive effect on improving the generalization performance.

The Number of Negative Samples

We employ three datasets to study the effect of negative sample size on generalization performance, and the results are shown in Figure 3. We can find that too small or too large number of negative samples are not good in our negative sampling scheme. It is retional that few negative samples cannot provide enough information for contrastive learning, while too many negative samples will inevitably contain more false

Dataset	BBBP	Tox21	ToxCast	SIDER	ClinTox	MUV	HIV	BACE	Avg
No-Pre-Train	65.8 ± 4.5	74.0 ± 0.8	63.4 ± 0.6	57.3 ± 1.6	58.0 ± 4.4	71.8 ± 2.5	75.3 ± 1.9	70.1 ± 5.4	67.0
EdgePred	67.3 ± 2.4	76.0 ± 0.6	64.1 ± 0.6	60.4 ± 0.7	64.1 ± 3.7	74.1 ± 2.1	76.3 ± 1.0	79.9 ± 0.9	70.3
InfoGraph	68.2 ± 0.7	75.5 ± 0.6	63.1 ± 0.3	59.4 ± 1.0	70.5 ± 1.8	75.6 ± 1.2	77.6 ± 0.4	78.9 ± 1.1	70.3
AttrMasking	64.3 ± 2.8	**76.7** ± 0.4	64.2 ± 0.5	61.0 ± 0.7	71.8 ± 4.1	74.7 ± 1.4	77.2 ± 1.1	79.3 ± 1.6	71.1
ContextPred	68.0 ± 2.0	75.7 ± 0.4	63.9 ± 0.6	60.9 ± 0.6	65.9 ± 3.8	**75.8** ± 1.7	77.3 ± 1.0	79.6 ± 1.2	70.9
GraphPartition	70.3 ± 0.7	75.2 ± 0.4	63.2 ± 0.3	61.0 ± 0.8	64.2 ± 0.5	75.4 ± 1.7	77.1 ± 0.7	79.6 ± 1.8	70.8
CuCo	**71.4** ± 1.2	75.8 ± 0.4	**65.2** ± 0.5	**62.1** ± 0.5	**76.8** ± 2.6	72.2 ± 2.5	**79.8** ± 0.7	**80.6** ± 1.3	**72.9**

Table 4: Transfer learning comparison with different manually designed pre-training schemes.

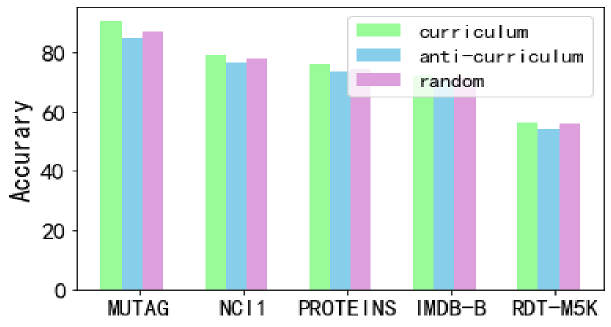

Figure 4: Bar plots showing the best mean accuracy, for curriculum (green), anti-curriculum (blue), random-curriculum (purple) sampling for five classical graph classification benchmarks.

negative samples in the later stage of the curriculum, which will hurt the generalization.

The Value of Ordered Learning

Equipped with the ingredients described in the previous sections, we investigate the relative benefits of curriculum negative sampling. To understand whether anti-curriculum, or random learning provides any benefit over ordinary training, we replicate this procedure with three random seeds and five standard benchmark datasets. For each experiment in this sweep, we select the best configuration and stopping time and record the corresponding test accuracy. The results of these runs are shown in Figure 4. Compared with random negative sampling, the curriculum negative sampling has an average improvement of the accuracy by about 2% for five benchmarks. In addition, the anti-curriculum negative sampling is not helpful for the model.

4.2 Transfer Learning

We perform transfer learning on molecular property prediction in chemistry and protein function prediction in biology following [Weihua *et al.*, 2020a], which pre-trains and fine-tunes the model in different datasets to evaluate the transferability of the pre-training scheme. We fine-tune the pre-trained GNN model with a small portion of labels on downstream tasks.

Experiments Setting

We adopt the same train-test and model selection procedure as in [Xu *et al.*, 2018], where we perform 10-fold cross- validation and select the epoch with the best cross-validation performance averaged over the 10 folds. The evaluation metric is ROC-AUC score. For fine-tuning on a downstream task, a linear classifier is appended on the top of pre-trained GNN. All reported results are averaged over five independent runs under the same configuration. In our transfer learning experiments, we evaluate model performance on eight Open Graph Benchmark (OGB) [Weihua *et al.*, 2020b] molecule property prediction datasets. Traditional unsupervised methods have no ability to transfer knowledge to other domain datasets. So we consider six baselines, including non-pretrain (direct supervised learning) and five state-of-the-art GNN self-supervised learning (SSL) methods including EdgePred [Hamilton *et al.*, 2017], InfoGraph [Sun *et al.*, 2019], AttrMasking [Weihua *et al.*, 2020a], ContextPred [Weihua *et al.*, 2020a] and GraphPartition [You *et al.*, 2020b].

Transferability of CuCo

Table 4 reports the performance of proposed CuCo method compared with other works in transfer learning setting. Among all self-supervised learning strategies, our method outperforms all baselines on average performance and achieves the highest results on on six of eight datasets. We gain a 5% performance enhancement against non-pretrain baseline, which well indicates the effectiveness of our proposed CuCo on transfer learning.

5 Conclusion

In this paper, we study the impact of negative samples on learning graph-level representations, and propose a novel curriculum contrastive learning framework for self-supervised graph-level representation called CuCo. We look into different ways of associating difficulty to negative samples using scoring functions and a variety of schedules known as pacing functions for introducing negative samples to the self-supervised training procedure. We conduct comprehensive experiments, which show that the proposed method improves downstream convergence speed and performance on fifteen datasets.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (No.U20B2045, 61772082, 61702296, 62002029), ”The Fundamental Research Funds for the Central Universities 2021RC28”.

References

[Adhikari *et al.*, 2018] Bijaya Adhikari, Yao Zhang, Naren Ramakrishnan, and B Aditya Prakash. Sub2vec: Feature learning for subgraphs. In *PAKDD*, pages 170–182. Springer, 2018.

[Bengio *et al.*, 2009] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *ICML*, pages 41–48, 2009.

[Cai *et al.*, 2018] Hongyun Cai, Vincent W. Zheng, and Chen Chuan Chang. A comprehensive survey of graph embedding: Problems, techniques, and applications. *IEEE Transactions on Knowledge and Data Engineering*, 30(9):1616–1637, 2018.

[Devlin *et al.*, 2018] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[Fouss *et al.*, 2007] Francois Fouss, Alain Pirotte, Jean Michel Renders, and Marco Saerens. Random-walk computation of similarities between nodes of a graph with application to collaborative recommendation. *IEEE Transactions on Knowledge and Data Engineering*, 19:355–369, 2007.

[Gilmer *et al.*, 2017] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *ICML*, pages 1263–1272, 2017.

[Guo *et al.*, 2018] Sheng Guo, Weilin Huang, Haozhi Zhang, Chenfan Zhuang, Dengke Dong, Matthew R Scott, and Dinglong Huang. Curriculumnet: Weakly supervised learning from large-scale web images. In *ECCV*, pages 135–150, 2018.

[Hacohen and Weinshall, 2019] Guy Hacohen and D. Weinshall. On the power of curriculum learning in training deep networks. In *ICML*, 2019.

[Hamilton *et al.*, 2017] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NIPS*, pages 1024–1034, 2017.

[He *et al.*, 2020] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *CVPR*, pages 9729–9738, 2020.

[Jiang *et al.*, 2017] Biaobin Jiang, Kyle Kloster, David F Gleich, and Michael Gribskov. Aptrank: an adaptive pagerank model for protein function prediction on bi-relational graphs. *Bioinformatics*, 33(12):1829–1836, 2017.

[Kalantidis *et al.*, 2020] Yannis Kalantidis, Mert Bulent Sariyildiz, Noe Pion, Philippe Weinzaepfel, and Diane Larlus. Hard negative mixing for contrastive learning. *arXiv preprint arXiv:2010.01028*, 2020.

[Kipf and Welling, 2016] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2016.

[Narayanan *et al.*, 2017] Annamalai Narayanan, Mahinthan Chandramohan, Rajasekar Venkatesan, Lihui Chen, Yang Liu, and Shantanu Jaiswal. graph2vec: Learning distributed representations of graphs. *arXiv preprint arXiv:1707.05005*, 2017.

[Oord *et al.*, 2018] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

[Pentina *et al.*, 2015] Anastasia Pentina, Viktoriia Sharmanska, and Christoph H Lampert. Curriculum learning of multiple tasks. In *CVPR*, pages 5492–5500, 2015.

[Robinson *et al.*, 2020] Joshua Robinson, Ching-Yao Chuang, Suvrit Sra, and Stefanie Jegelka. Contrastive learning with hard negative samples. *arXiv preprint arXiv:2010.04592*, 2020.

[Shervashidze *et al.*, 2011] Nino Shervashidze, Pascal Schweitzer, Erik Jan Van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(9), 2011.

[Sun *et al.*, 2019] Fan-Yun Sun, Jordan Hoffmann, Vikas Verma, and Jian Tang. Infograph: Unsupervised and semi-supervised graph-level representation learning via mutual information maximization. *arXiv preprint arXiv:1908.01000*, 2019.

[Veličković *et al.*, 2018] Petar Veličković, William Fedus, William L Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. Deep graph infomax. *arXiv preprint arXiv:1809.10341*, 2018.

[Weihua *et al.*, 2020a] Hu Weihua, Liu Bowen, Gomes Joseph, Zitnik Marinka, Liang Percy, Pande Vijay, and Leskovec Jure. Strategies for pre-training graph neural networks. In *ICML*, 2020.

[Weihua *et al.*, 2020b] Hu Weihua, Fey Matthias, Zitnik Marinka, Dong Yuxiao, Ren Hongyu, Liu Bowen, Catasta Michele, and Leskovec Jure. Open graph benchmark: Datasets for machine learning on graphs. *arXiv preprint arXiv:2005.00687*, 2020.

[Xu *et al.*, 2018] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

[Yanardag and Vishwanathan, 2015] Pinar Yanardag and SVN Vishwanathan. Deep graph kernels. In *KDD*, pages 1365–1374, 2015.

[Ying *et al.*, 2018] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In *NIPS*, pages 4800–4810, 2018.

[You *et al.*, 2020a] Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. Graph contrastive learning with augmentations. *arXiv preprint arXiv:2010.13902*, 2020.

[You *et al.*, 2020b] Yuning You, Tianlong Chen, Zhangyang Wang, and Yang Shen. When does self-supervision help graph convolutional networks? In *ICML*, pages 10871–10880, 2020.