BatchSampler: Sampling Mini-Batches for Contrastive Learning in Vision, Language, and Graphs

Zhen Yang*
Tsinghua University
yangz21@mails.tsinghua.edu.cn

Tinglin Huang*†
Tsinghua University
Yale University
tinglin.huang@yale.edu

Ming Ding*
Tsinghua University
dm18@mails.tsinghua.edu.cn

Yuxiao Dong‡
Tsinghua University
yuxiaod@tsinghua.edu.cn

Rex Ying
Yale University
rex.ying@yale.edu

Yukuo Cen
Tsinghua University
cyk20@mails.tsinghua.edu.cn

Yangliao Geng
Tsinghua University
gengyla@mail.tsinghua.edu.cn

Jie Tang‡
Tsinghua University
jietang@tsinghua.edu.cn

ABSTRACT

In-Batch contrastive learning is a state-of-the-art self-supervised method that brings semantically-similar instances close while pushing dissimilar instances apart within a mini-batch. Its key to success is the negative sharing strategy, in which every instance serves as a negative for the others within the mini-batch. Recent studies aim to improve performance by sampling hard negatives *within the current mini-batch*, whose quality is bounded by the mini-batch itself. In this work, we propose to improve contrastive learning by sampling mini-batches from the input data. We present BatchSampler[1] to sample mini-batches of hard-to-distinguish (i.e., hard and true negatives to each other) instances. To make each mini-batch have fewer false negatives, we design the proximity graph of randomly-selected instances. To form the mini-batch, we leverage random walk with restart on the proximity graph to help sample hard-to-distinguish instances. BatchSampler is a simple and general technique that can be directly plugged into existing contrastive learning models in vision, language, and graphs. Extensive experiments on datasets of three modalities show that BatchSampler can consistently improve the performance of powerful contrastive models, as shown by significant improvements of SimCLR on ImageNet-100, SimCSE on STS (language), and GraphCL and MVGRL on graph datasets.

CCS CONCEPTS

• **Computing methodologies** → **Learning paradigms**.

KEYWORDS

Mini-Batch Sampling; Global Hard Negatives; Contrastive Learning

ACM Reference Format:
Zhen Yang, Tinglin Huang, Ming Ding, Yuxiao Dong, Rex Ying, Yukuo Cen, Yangliao Geng, and Jie Tang. 2023. BatchSampler: Sampling Mini-Batches for Contrastive Learning in Vision, Language, and Graphs. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '23), August 6–10, 2023, Long Beach, CA, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3580305.3599263

1 INTRODUCTION

Contrastive learning [25, 34] is one of the dominant strategies for self-supervised representation learning across various data domains, such as MoCo [20] and SimCLR [10] in computer vision, SimCSE [16] in natural language processing, and GraphCL [57] in graph representation learning. The essence of self-supervised contrastive learning is to make similar instances close to each other and dissimilar ones farther away in the learned representation space.

The course of self-supervised contrastive models usually starts with loading each mini-batch of B instances sequentially from the input data (e.g., the images in Figure 2 (a)). In each batch, each instance x is associated with its augmentation x^+ as the positive samples and the other instances as negatives $\{x^-\}$. Commonly by using the InfoNCE loss [34], the goal of contrastive learning is to discriminate instances by mapping positive pairs (x, x^+) to similar embeddings and negative pairs (x, x^-) to dissimilar embeddings.

Given the self-supervised contrastive setting, the negative samples $\{x^-\}$—with unknown labels—play an essential role in the contrastive optimization process. To improve in-batch contrastive learning, there are various attempts to take on them from different perspectives. Globally, SimCLR [10] shows that simply increasing the size B of the mini-batch (e.g., 8192)—i.e., more negative samples—outperforms previously carefully-designed strategies, such as the memory bank [49] and the consistency improvement of the stored negatives in MoCo [20]. Locally, given each mini-batch, recent studies such as the DCL and HCL [12, 38] methods have focused on identifying true or hard negatives within this batch. In other words,

*These authors contributed equally to this research.
†This work was done when the author worked at Tsinghua University.
‡YD and JT are the corresponding authors.
[1]The code is available at https://github.com/THUDM/BatchSampler

KDD '23, August 6–10, 2023, Long Beach, CA, USA

ACM ISBN 979-8-4007-0103-0/23/08.
https://doi.org/10.1145/3580305.3599263

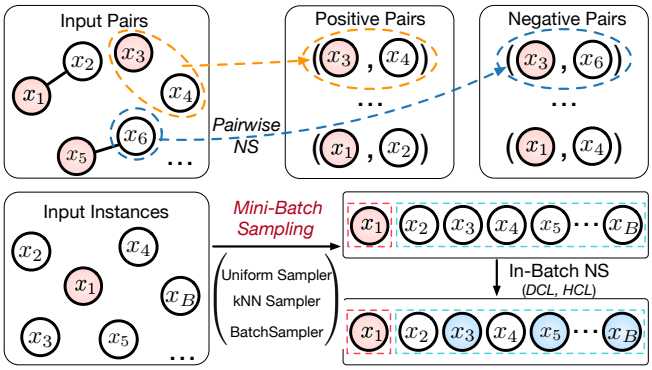

Figure 1: Technical comparison between Mini-Batch Sampling, Pairwise Negative Sampling (NS), and In-Batch Negative Sampling. The blue nodes are hard negatives.

existing efforts have largely focused on designing better negative sampling techniques after each mini-batch of instances is loaded.

Problem. In this work, we instead propose to globally sample instances from the input data to form mini-batches. The goal is to have mini-batches that naturally contain as many hard negatives $\{x^-\}$—that have different (underlying) labels from x (true negatives) but similar representations with x—for each instance x as possible. In this way, the discrimination between positive and negative instances in each mini-batch can better inform contrastive optimization.

Uniform & kNN Samplers. Traditionally, there are two ways to form mini-batches (Cf. Figure 2 (a)). The common option is the *Uniform Sampler* that sequentially loads or uniformly samples a batch of instances for each training step[10, 16, 57]. However, the Uniform Sampler neglects the effect of hard negatives [27, 38], and the batches formed contain easy negatives with low gradients that contribute little to optimization [50, 54]. In order to have hard negatives, it is natural to cluster instances that are nearest to each other in the representation space to form each batch, that is, the *kNN Sampler*. Unfortunately, the instances with the same underlying labels are also expected to cluster together [8, 10], resulting in high percentages of false negatives in the batch (Cf. Figure 2 (c)).

Contributions: BatchSampler. We present BatchSampler to sample mini-batches of hard-to-distinguish instances for in-batch contrastive learning. Fundamentally, each mini-batch is required to cover hard yet true negatives, thus addressing the issues faced by Uniform and kNN Samplers. To achieve this, we design the proximity graph of randomly-selected instances in which each edge is used to control the pairwise similarity of their representations—that is, the hardness of negative samples. The false negative issue in the kNN Sampler is mitigated when random instances are picked to construct the graph. To form one batch, we leverage random walk with restart on the proximity graph to draw instances. The premise is that the local neighbors sampled by the walkers are similar, that is, hard-to-distinguish.

BatchSampler is a simple and general technique that can be directly plugged into in-batch contrastive models in vision, language, and graphs. The experimental results show that the well-known contrastive learning models—SimCLR [10] and MoCo v3 [11] in vision, SimCSE [16] in language, and GraphCL [57] and MVGRL [19] in graphs—can benefit from the mini-batches formed by BatchSampler. We also theoretically and empirically show that how BatchSampler balances the challenges faced in the Uniform and kNN Samplers.

Differences from Pairwise Negative Sampling. The problem of mini-batch sampling is different from the pairwise negative sampling problem. As shown in Figure 1, the pairwise negative sampling method samples negative instances for each positive pair, whereas the mini-batch sampling methods focus on sampling mini-batches that reuses other instances as negatives in the batch. Massive previous works in pairwise negative sampling [22, 23, 28, 50, 54] have attempted to improve performance by globally selecting similar negatives for a given anchor from the entire dataset. However, our focus is on globally sampling mini-batches that contain many hard negatives rather than only selecting hard negatives for each pair.

2 RELATED WORK

Contrastive learning in different modalities. Contrastive learning follows a similar paradigm that contrasts similar and dissimilar observations based on noise contrastive estimation (NCE) [18, 34]. The primary distinction between contrastive methods of different modalities is how they augment the data. As for computer vision, MoCo [20] and SimCLR [10] augment data with geometric transformation and appearance transformation. Besides simply using data augmentation, WCL [62] additionally utilizes an affinity graph to construct positive pairs for each example within the mini-batch. Wu et al. [47] design a data replacement strategy to mine hard positive instances for contrastive learning. As for language, CLEAR [48] and COCO-LM [32] augment the text data through word deletion, reordering, and substitution, while SimCSE [16] obtains the augmented instances by applying the standard dropout twice. As for graphs, DGI [36] and InfoGraph [41] treat the node representations and corresponding graph representations as positive pairs. Besides, InfoGCL[51], JOAO [56], GCA [65], GCC [37] and GraphCL [57] augment the graph data by graph sampling or proximity-oriented methods. MVGRL [19] proposes to compare the node representation in one view with the graph representation in the other view. Zhu et al. [64] compares different kinds of graph augmentation strategies. Our proposed BatchSampler is a general mini-batch sampler that can directly be applied to any in-batch contrastive learning framework with different modalities.

Negative sampling in contrastive learning. Previous studies about negative sampling in contrastive learning roughly fall into two categories: **(1) Memory-based negative sampling strategy**, such as MoCo [20], maintains a fixed-size memory bank to store negatives which are updated regularly during the training process. MoCHI [27] and m-mix [60] propose to mix the hard negative candidates at the feature level to generate more challenging negative pairs. MoCoRing [46] samples hard negatives from a defined conditional distribution which keeps a lower bound on the mutual information. **(2) In-batch negative sharing strategy**, such as SimCLR [10] and MoCo v3 [11], adopts different instances in the

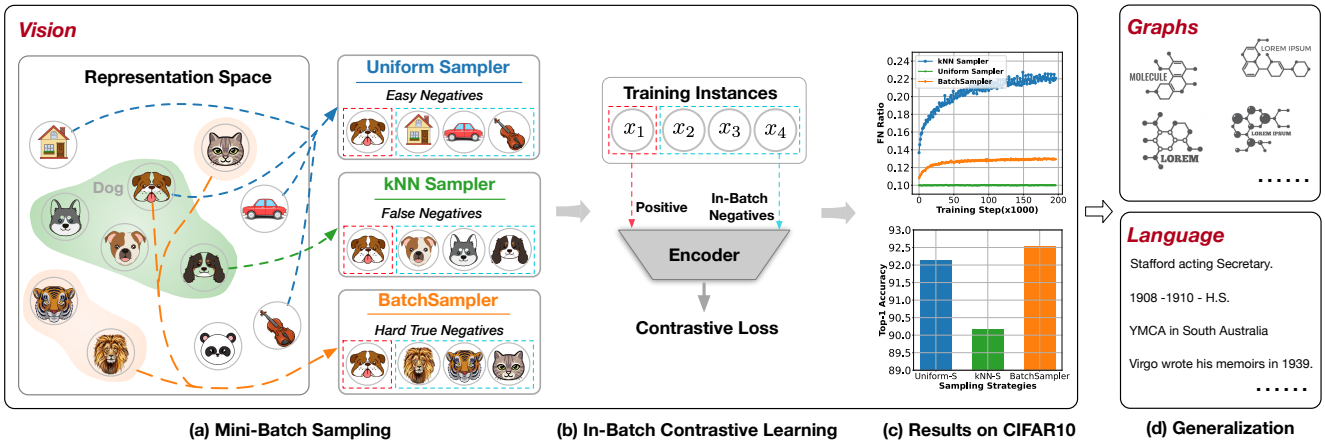

Figure 2: A motivating illustration of BatchSampler, using vision as an example. Uniform Sampler randomly samples a batch of instances, which contains easy negatives. kNN Sampler selects the nearest instances to form a batch, resulting in so many false negatives. BatchSampler samples a mini-batch with hard yet true negatives based on proximity graph.

current mini-batch as negatives. To mitigate the false negative issue, DCL [12] modifies the original InfoNCE objective to reweight the contrastive loss. Huynh et al. [24] identifies the false negatives within a mini-batch by comparing the similarity between negatives and the anchor image's multiple support views. Additionally, HCL [38] revises the original InfoNCE objective by assigning higher weights for hard negatives among the mini-batch. Recently, UnReMix [42] is proposed to sample hard negatives by effectively capturing aspects of anchor similarity, representativeness, and model uncertainty. However, such locally sampled hard negatives cannot exploit hard negatives sufficiently from the dataset.

Global hard negative sampling methods on triplet loss have been widely investigated, which aim to globally sample hard negatives for a given positive pair. For example, Wang et al. [45] proposes to take rank-k hard negatives from some randomly sampled negatives. Xiong et al. [50] globally samples hard negatives by an asynchronously-updated approximate nearest neighbor (ANN) index for dense text retrieval. Different from the above methods which are applied to a triplet loss for a given pair, BatchSampler samples mini-batches with hard negatives for InfoNCE loss.

3 PROBLEM: MINI-BATCH SAMPLING FOR CONTRASTIVE LEARNING

In-Batch Contrastive Learning. *In-batch contrastive learning* commonly follows or slightly updates the following objective [18, 34] across different domains, such as graphs [37, 57], vision [10, 11], and language [16]:

$$\min \ \mathbb{E}_{\{x_1 \dots \ x_B\} \subset \mathcal{D}} \left[-\sum_{i=1}^{B} \log \frac{e^{f(x_i)^T f(x_i^+)}}{e^{f(x_i)^T f(x_i^+)} + \sum_{j \neq i} e^{f(x_i)^T f(x_j)}} \right], \quad (1)$$

where $\{x_1 \dots \ x_B\}$ is a mini-batch of samples (usually) sequentially loaded from the dataset $\mathcal{D}$, and x_i^+ is an augmented version of x_i. The encoder $f(\cdot)$ learns to discriminate instances by mapping different data-augmentation versions of the same instance (positive pairs) to similar embeddings, and mapping different instances in the mini-batch (negative pairs) to dissimilar embeddings.

Usually, the in-batch negative sharing strategy—every instance serves as a negative to the other instances within the mini-batch—is used to boost the training efficiency [10]. It is then natural to have hard negatives in each mini-batch for improving contrastive learning. Straightforwardly, we could attempt to sample hard negatives within the mini-batch [12, 28, 38]. However, the batch size of a mini-batch is—by definition—far smaller than the size of the input dataset, and existing studies show that sampling such a local sampling method fails to effectively explore all the hard negatives [50, 61].

Problem: Mini-Batch Sampling. The goal of this work is to have a general mini-batch sampling strategy to support different modalities of data. Specifically, given a set of data instances $\mathcal{D} = \{x_1, \cdots, x_N\}$, the objective is to design a modality-independent sampler to sample a mini-batch of instances where each pair of instances are hard to distinguish across the dataset.

There are two existing strategies—Uniform Sampler and kNN Sampler—adopted in contrastive learning.

Uniform Sampler is the most common strategy used in contrastive learning [10, 16, 57]. The pipeline is to first randomly sample a batch of instances for each training step, then feed them into the model for optimization.

Though simple and model-independent, Uniform Sampler neglects the effect of hard negatives [27, 38], and the batches formed contain negatives with low gradients that contribute little to optimization. Empirically, we show that Uniform Sampler results in a low percentage of similar instance pairs in a mini-batch (Cf. Figure 4). Theoretically, Yang et al. [54] and Xiong et al. [50] also prove that the sampled negative should be similar to the query instance since it can provide a meaningful gradient to the model.

kNN Sampler globally samples a mini-batch with many hard negatives. As its name indicates, it tries to pick an instance at random and retrieve a set of nearest neighbors to construct a batch.

However, the instances of the same 'class' will be clustered together in the embedding space [8, 10]. Hence the negatives retrieved by it are hard at first but would be replaced by false negatives (FNs) as the training epochs increase, misguiding the model training.

In summary, Uniform Sampler leverages random negatives to guide the optimization of the model; whereas the kNN one explicitly samples hard negatives but suffers from false negatives. Thus, an ideal negative sampler for in-batch contrastive learning should balance between kNN and Uniform Samplers, ensuring both the exploitation of hard negatives and the mitigation of the FN issue.

4 THE BATCHSAMPLER METHOD

To improve contrastive learning, we propose the BatchSampler method with the goal of forming mini-batches with 1) hard-to-distinguish instances while 2) avoiding false negatives. BatchSampler is a simple and general strategy that can be directly plugged into existing in-batch contrastive learning models in vision, language, and graphs. Figure 3 shows the overview of BatchSampler.

The basic idea of BatchSampler is to form mini-batches by **globally** sampling instances based on the similarity between each pair of them, which is significantly different from previous local sampling methods in contrastive learning [12, 27, 38]. To achieve this, the first step of BatchSampler is to construct the proximity graph of instances with edges measuring the pairwise similarity. Second, we perform mini-batch sampling as a walk by leveraging a simple and commonly-used graph sampling technique, random walk with restart, to sample instances (nodes) from the proximity graph.

4.1 Proximity Graph Construction

The goal of the proximity graph is to collect candidate instances that alleviate the issues faced by Uniform and kNN Sampler by connecting similar instances while reducing the false negative pairs. If neighbors of an instance are randomly chosen from the dataset, sampling on the resulting graph resembles Uniform Sampler. Conversely, if the most similar instances are chosen as neighbors, it resembles the behavior of kNN Sampler. In BatchSampler, we propose a simple strategy to construct the proximity graph as follows.

Given a training dataset, we have N instances $\{v_i | i = 1, \cdots, N\}$ and their corresponding representations $\{\mathbf{e}_i | i = 1, \cdots, N\}$ generated by the encoder $f(\cdot)$. The proximity graph is formulated as:

$$G = (\mathcal{V}, \mathcal{E}), \tag{2}$$

where the node set $\mathcal{V} = \{v_1, \cdots, v_N\}$ denotes the instances and $\mathcal{E} \subseteq \{(v_i, v_j) | v_i, v_j \in \mathcal{V}\}$ is a set of node pairs. Let $\mathcal{N}_i$ be the neighbor set of v_i in the proximity graph. To construct $\mathcal{N}_i$, we first form a candidate set $C_i = \{v_m\}$ for each instance v_i by uniformly picking $M (M \ll N)$ neighbor candidates. Each node and its neighbor form an edge, resulting in the edge set $\mathcal{E}$. Then we select the K nearest ones from the candidate set:

$$\mathcal{N}_i = \underset{v_m \in C_i}{\text{TopK}} \left(\mathbf{e}_i \cdot \mathbf{e}_m \right), \tag{3}$$

where $\cdot$ is the inner product operation. M is used to control the similarity between the center node and its immediate neighbor nodes, which can be demonstrated by the following proposition:

Proposition 1. *Given an instance v_i with the corresponding representation $\mathbf{e}_i$, assume that there are at least S instances whose inner product similarity with v_i is larger than s, i.e.,*

$$\left| \{ v_j \in \mathcal{V} \mid \mathbf{e}_i \cdot \mathbf{e}_j > s \} \right| \geq S. \tag{4}$$

Then in the proximity graph G, the similarity between v_i and its neighbors is larger than s with proximate probability at least:

$$\mathbb{P}\left\{ \mathbf{e}_i \cdot \mathbf{e}_k > s, \forall v_k \in \mathcal{N}_i \right\} \gtrapprox \left(1 - p^M\right)^K, \tag{5}$$

where $p = \frac{N-S}{N}$, and K is the number of neighbors.

Proof. Since $M \ll N$, we can approximately assume that the sampling is with replacement. In this case, we have

$$\mathbb{P}\left\{ \mathbf{e}_i \cdot \mathbf{e}_k > s, \forall v_k \in \mathcal{N}_i \right\} = 1 - \sum_{k=0}^{K-1} \binom{M}{k} p^{M-k} \left(1-p\right)^k. \tag{6}$$

Then let us prove (5) by induction. When $K = 1$, the conclusion clearly holds.

Assuming that the conclusion holds when $K = L - 1$, let us consider the case when $K = L$. We have

$$1 - \sum_{k=0}^{L-1} \binom{M}{k} p^{M-k} (1-p)^k \gtrapprox \left(1 - p^M\right)^{L-1} - \binom{M}{L-1} p^{M-L+1} \left(1-p\right)^{L-1}. \tag{7}$$

To prove the conclusion, we only need to show

$$\left(1 - p^M\right)^{L-1} p^M \gtrapprox \binom{M}{L-1} p^{M-L+1} \left(1-p\right)^{L-1}, \tag{8}$$

or equivalently

$$\begin{aligned} \left(1 - p^M\right)^{L-1} p^{L-1} &= \left(1 - p^M\right)^{L-1} \left(\frac{N-S}{N}\right)^{L-1} \\ &\gtrapprox \binom{M}{L-1} \left(\frac{S}{N}\right)^{L-1} = \binom{M}{L-1} \left(1-p\right)^{L-1}. \end{aligned} \tag{9}$$

On the other hand, according to [29], we have

$$\binom{M}{L-1} \leq \left(\frac{eM}{L-1}\right)^{L-1}, \tag{10}$$

where e denotes the Euler's number. Substituting (10) into (9), we only need to show

$$(N-S)(L-1)\left(1 - p^M\right) \gtrapprox eMS. \tag{11}$$

The above relation holds depending on the choices of M, S and L, which can be satisfied in our scenario in most cases. □

Proposition 1 suggests that the candidate set size M can control the similarity between each center node and its immediate neighbor nodes. A larger M indicates a greater probability that two adjacent nodes are similar, and the proximity graph constructed would be more like the graph of nodes clustered by kNN. If M is small and close to K, the instances in the proximity graph can be considered as randomly-selected ones, that is, by Uniform Sampler.

4.2 Proximity Graph Sampling

We perform the mini-batch sampling as a walk in the proximity graph, which collects the visited instances as sampling results from a sourced node. Here we propose to apply Random Walk with Restart (RWR), which offers a theoretically supported ability to control the walker's behavior.

As shown in Algorithm 3, starting from a node, the sampler moves from one node to another by either teleporting back to the start node with probability α or moving to a neighboring node

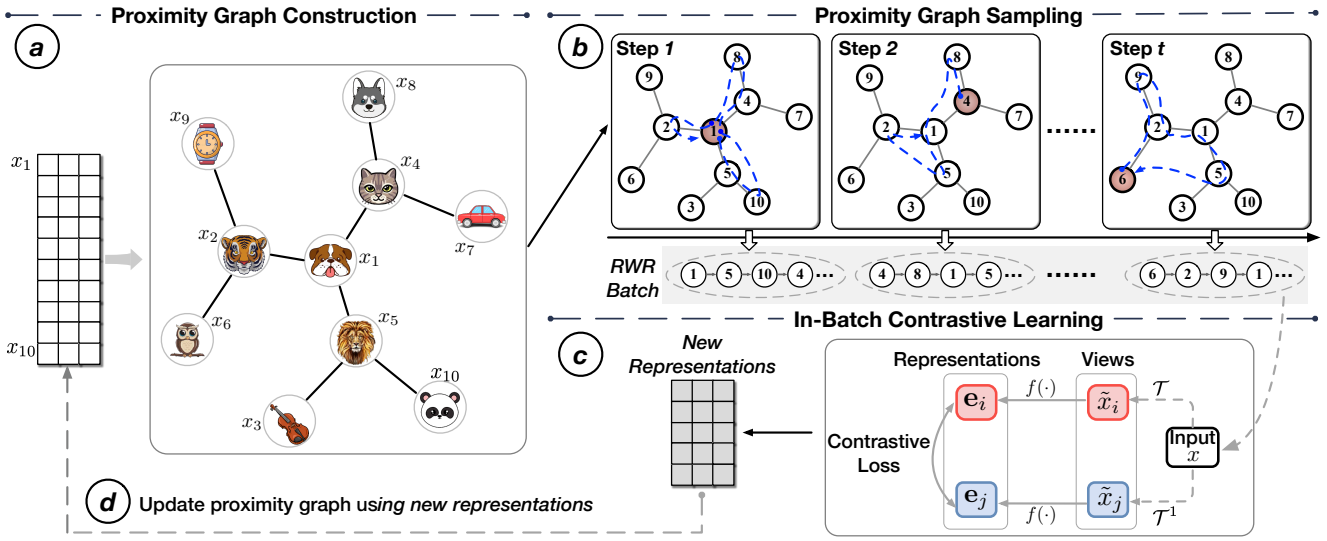

Figure 3: The framework of BatchSampler, using the vision modality as an example. The proximity graph is first constructed based on generated image representations and will be updated every t training step. Next, a proximity graph-based negative sampler is applied to generate a batch with hard negatives for in-batch contrastive learning.

proportional to the edge weight. The process continues until a fixed number of nodes are collected and taken as the mini-batch. The effectiveness of RWR lies in its ability to modulate the probability of sampling within a neighborhood by adjusting α, as demonstrated by Proposition 2:

Proposition 2. *For all $0 < \alpha \leq 1$ and $\mathcal{S} \subset \mathcal{V}$, the probability that a Lazy Random Walk with Restart starting from a node $u \in \mathcal{S}$ escapes $\mathcal{S}$ satisfies $\sum_{v \in (\mathcal{V}-\mathcal{S})} \mathbf{p}_u(v) \leq \frac{1-\alpha}{2\alpha}\Phi(\mathcal{S})$, where $\mathbf{p}_u$ is the stationary distribution, and $\Phi(\mathcal{S})$ is the graph conductance of $\mathcal{S}$.*

Proof. We first introduce the definition of graph conductance [66] and Lazy Random Walk [40]:

Graph Conductance. For an undirected graph $G = (\mathcal{V}, \mathcal{E})$, the graph volume of a node set $\mathcal{S} \subset \mathcal{V}$ is defined as $\text{vol}(\mathcal{S}) = \sum_{v \in \mathcal{S}} d(v)$, where $d(v)$ is the degree of node v. The edge boundary of a node set is defined to be $\partial(\mathcal{S}) = \{(x, y) \in \mathcal{E} | x \in \mathcal{S}, y \notin \mathcal{S}\}$. The conductance of $\mathcal{S}$ is calculated as followed:

$$\Phi(\mathcal{S}) = \frac{|\partial(\mathcal{S})|}{\min(\text{vol}(\mathcal{S}), \text{vol}(\mathcal{V} - \mathcal{S}))} \tag{12}$$

Lazy Random Walk. Lazy Random Walk (LRW) is a variant of Random Walk, which first starts at a node, then stays at the current position with a probability of 1/2 or travels to a neighbor. The transition matrix of a lazy random walk is $\mathbf{M} \triangleq (\mathbf{I} + \mathbf{A}\mathbf{D}^{-1})/2$, where the $\mathbf{I}$ denotes the identity matrix, $\mathbf{A}$ is the adjacent matrix, and $\mathbf{D}$ is the degree matrix. The K-th step Lazy Random Walk distribution starting from a node u is defined as $\mathbf{q}^{(K)} \leftarrow \mathbf{M}^K \mathbf{1}_u$.

We then present a theorem that relates the Lazy Random Walk to graph conductance, which has been proved in Spielman and Teng [40]:

Theorem 1. *For all $K \geq 0$ and $\mathcal{S} \subset \mathcal{V}$, the probability that a K-step Lazy Random Walk starting at $u \in \mathcal{S}$ escapes $\mathcal{S}$ satisfies $\mathbf{q}^{(K)}(\mathcal{V} - \mathcal{S}) \leq K\Phi(\mathcal{S})/2$.*

Theorem 1 guarantees that given a non-empty node set $\mathcal{S} \subset \mathcal{V}$ and a start node $u \in \mathcal{S}$, the Lazy Random Walker will be more likely stuck at $\mathcal{S}$. Here we extend the LRW to Lazy Random Walk with Restart (LRWR) which will return to the start node with probability α or perform Lazy Random Walk. According to the previous studies [7, 14, 35, 44], we can obtain a stationary distribution $\mathbf{p}_u$ by recursively performing Lazy Random Walk with Restart, which can be formulated as a linear system:

$$\mathbf{p}_u = \alpha \mathbf{1}_u + (1 - \alpha)\mathbf{M}\mathbf{p}_u \tag{13}$$

where α denotes the restart probability. $\mathbf{p}_u$ can be expressed as a geometric sum of Lazy Random Walk [13]:

$$\mathbf{p}_u = \alpha \sum_{l=0}^{\infty} (1 - \alpha)^l \mathbf{M}^l \mathbf{1}_u = \alpha \sum_{l=0}^{\infty} (1 - \alpha)^l \mathbf{q}_u^{(l)} \tag{14}$$

Applying Theorem 1, we have:

$$\begin{aligned} \sum_{v \in (\mathcal{V}-\mathcal{S})} \mathbf{p}_u(v) &= \alpha \sum_{l=0}^{\infty} \sum_{v \in (\mathcal{V}-\mathcal{S})} (1 - \alpha)^l \mathbf{q}_u^{(l)}(v) \\ &\leq \alpha \sum_{l=0}^{\infty} l(1 - \alpha)^l \Phi(\mathcal{S})/2 \\ &= \frac{1 - \alpha}{2\alpha} \Phi(\mathcal{S}) \end{aligned} \tag{15}$$

where the element $\mathbf{p}_u(v)$ represents the probability of the walker starting at u and ending at v. The desired result is obtained by comparing two sides of (15). □

The only difference between Lazy Random with Restart and Random Walk with Restart is that the former has a probability of remaining in the current position without taking action. They are equivalent when sampling a predetermined number of nodes.

Algorithm 1: Constrative Learning with BatchSampler

Input: Dataset $\mathcal{D} = \{x_i | i = 1, \cdots, N\}$, Encoder $f(\cdot)$, Batchsize B, Graph update interval t, Modality-specific augmentation functions $\mathcal{T}$.

for *iter* $\leftarrow 0, 1, \cdots$ **do**

// BatchSampler

if *iter*%t == 0 **then**

// Proximity Graph Construction

Build the proximity graph G by Algorithm 2.

end

// Proximity Graph Sampling

Perform sampling in G by Algorithm 3.

// Standard Contrastive Pipeline in Different Modalities

Load the mini-batch $\{x_i\}_B$.

Obtain positive pairs $\{(x_i, x_i^+)\}_B$ by augmentations $\mathcal{T}$.

Generate representations $\{(\mathbf{e}_i, \mathbf{e}_i^+)\}_B$ by Encoder $f(\cdot)$.

Compute the loss by treating $\{(\mathbf{e}_i, \mathbf{e}_j)\}^{i \neq j}$ as negatives.

Update the parameters of $f(\cdot)$.

end

Overall, Proposition 2 indicates that the probability of RWR escaping from a local cluster [6, 40] can be bounded by the graph conductance [66] and the restart probability α. Besides, RWR can exhibit a mixture of two straightforward sampling methods Breadth-first Sampling (BFS) and Depth-first Sampling (DFS) [17]:

- BFS collects all of the current node's immediate neighbors, then moves to its neighbors and repeats the procedure until the number of collected instances reaches batch size.
- DFS randomly explores the node branch as far as possible before the number of visited nodes reaches batch size.

Specifically, higher α indicates that the walker will approximate BFS behavior and sample within a small locality, while a lower α encourages the walker to explore nodes further away, like in DFS.

4.3 Discussion on BatchSampler

As shown in Algorithm 1, BatchSampler serves as a mini-batch sampler and can be easily plugged into any in-batch contrastive learning methods. Specifically, during the training process, BatchSampler first constructs the proximity graph, which will be updated after t training steps, then selects a start node at random and samples a mini-batch on proximity graph by RWR.

Connects to the Uniform and kNN Samplers. As shown in Figure 4, the number of candidates M and the restart probability α are the key to flexibly controlling the hardness of a sampled batch. When we set M as the size of dataset and α as 1, proximity graph is equivalent to kNN graph and graph sampler will only collect the immediate neighbors around a central node, which behaves similarly to a kNN Sampler. On the other hand, if M is set to 1 and α is set to 0, the RWR degenerates into the DFS and chooses the neighbors that are linked at random, which indicates that BatchSampler performs as a Uniform Sampler. We provide an empirical criterion of choosing M and α in Section 5.3.

Complexity. The time complexity of building a proximity graph is $O(NMd)$ where N is the dataset size, M is the candidate set size and d denotes the embedding size. It is practically efficient since usually M is much smaller than N, and the process can be accelerated by embedding retrieval libraries such as Faiss [26]. The space cost of BatchSampler mainly comes from graph construction and graph storage. The total space complexity of BatchSampler is $O(Nd+NK)$ where K is the number of neighbors in the proximity graph.

5 EXPERIMENTS

We plug BatchSampler into popular contrastive learning algorithms on three modalities, including vision, language, and graphs. Extensive experiments are conducted with 5 algorithms and 19 datasets, a total of 31 experimental settings. Additional experiments are reported in Appendix A.2, including batchsize B, neighbor number K, proximity graph update interval t.

5.1 Results

Results in Computer Vision. We first adopt SimCLR [10] and MoCo v3 [11] as the backbone based on ResNet-50 [21]. We start with training the model for 800 epochs with a batch size of 2048 for SimCLR and 4096 for MoCo v3, respectively. We then use linear probing to evaluate the representations on ImageNet. As shown in Table 1, our proposed model can consistently boost the performance of original SimCLR and MoCo v3, demonstrating the superiority of BatchSampler. Besides, we evaluate BatchSampler on the other benchmark datasets: two small-scale (CIFAR10, CIFAR100) and two medium-scale (STL10, ImageNet-100), which can be found in Appendix A.2.1.

Results in Language Model. We evaluate BatchSampler on learning the sentence representations by SimCSE [16] framework with pretrained BERT [15] as the backbone. The results of Table 2 suggest that BatchSampler consistently improves the baseline models with an absolute gain of 1.09%∼2.91% on 7 semantic textual similarities (STS) tasks [1–5, 31, 43]. Specifically, we observe that when applying DCL and HCL, the performance of the self-supervised language model averagely drops by 2.45% and 3.08% respectively. As shown in Zhou et al. [63], the pretrained language model offers a prior distribution over the sentences, leading to a high cosine similarity of both positive pairs and negative pairs. So DCL and HCL, which leverage the similarity of positive and negative scores to tune the weight of negatives, are inapplicable because the high similarity scores of positives and negatives will result in homogeneous weighting. However, the hard negatives explicitly sampled by our proposed BatchSampler can alleviate it, with an absolute improvement of 1.64% on DCL and 2.64% on HCL. The results of RoBERTa [30] are reported in Appendix A.2.2.

Results in Graph Learning. We test BatchSampler on graph-level classification task using GraphCL [57] and MVGRL [19] frameworks. Similar to language modality, we replace the original InfoNCE loss function in GraphCL with the DCL and HCL. Table 3 reports the detailed performance comparison on 7 benchmark datasets: IMDB-B, IMDB-M, COLLAB, REDDIT-B, PROTEINS, MUTAG, and NCI1 [58, 59]. We can observe that BatchSampler consistently

Table 1: Top-1 accuracy under the linear evaluation with the ResNet-50 backbone on ImageNet.

Method	100 ep	400 ep	800 ep
SimCLR	64.0	68.1	68.7
w/ BatchSampler	**64.7**	**68.6**	**69.2**
MoCo v3	68.9	73.3	73.8
w/ BatchSampler	**69.5**	**73.7**	**74.2**

* Only conduct experiments on BatchSampler.

Table 2: Overall performance comparison with the BERT backbone on STS tasks.

Method	STS12	STS13	STS14	STS15	STS16	STS-B	SICK-R	Avg.
SimCSE-BERT$_{base}$	68.62	80.89	73.74	80.88	77.66	**77.79**	**69.64**	75.60
w/ kNN Sampler	63.62	74.86	69.79	79.17	76.24	74.73	67.74	72.31
w/ BatchSampler	**72.37**	**82.08**	**75.24**	**83.10**	**78.43**	77.54	68.05	**76.69**
DCL-BERT$_{base}$	65.22	77.89	68.94	79.88	**76.72**	73.89	**69.54**	73.15
w/ kNN Sampler	66.34	76.66	72.60	78.30	74.86	73.65	67.92	72.90
w/ BatchSampler	**69.55**	**82.66**	**73.37**	**80.40**	75.37	**75.43**	66.76	**74.79**
HCL-BERT$_{base}$	62.57	79.12	69.70	78.00	75.11	73.38	69.74	72.52
w/ kNN Sampler	61.12	75.73	68.43	76.64	74.78	71.22	68.04	70.85
w/ BatchSampler	**66.87**	**81.38**	**72.96**	**80.11**	**77.99**	**75.95**	**70.89**	**75.16**

Table 3: Accuracy on graph classification task under LIBSVM [9] classifier.

Method	IMDB-B	IMDB-M	COLLAB	REDDIT-B	PROTEINS	MUTAG	NCI1
GraphCL	70.90±0.53	48.48±0.38	70.62±0.23	90.54±0.25	74.39±0.45	86.80±1.34	77.87±0.41
w/ kNN Sampler	70.72±0.35	47.97±0.97	70.59±0.14	90.21±.74	74.17±0.41	86.46±0.82	77.27±0.37
w/ BatchSampler	**71.90±0.46**	**48.93±0.28**	**71.48±0.28**	**90.88±0.16**	**75.04±0.67**	**87.78±0.93**	**78.93±0.38**
DCL	71.07±0.36	48.93±0.32	**71.06±0.51**	90.66±0.29	74.64±0.48	88.09±0.93	78.49±0.48
w/ kNN Sampler	70.94±0.19	48.47±0.35	70.49±0.37	90.26±1.03	74.28±0.17	87.13±1.40	78.13±0.52
w/ BatchSampler	**71.32±0.17**	**48.96±0.25**	70.44±0.35	**90.73±0.34**	**75.02±0.61**	**89.47±1.43**	**79.03±0.32**
HCL	**71.24±0.36**	48.54±0.51	71.03±0.45	90.40±0.42	74.69±0.42	87.79±1.10	78.83±0.67
w/ kNN Sampler	71.14±0.44	48.36±0.93	70.86±0.74	90.64±0.51	74.06±0.44	87.53±1.37	78.66±0.48
w/ BatchSampler	71.20±0.38	**48.76±0.39**	**71.70±0.35**	**91.25±0.25**	**75.11±0.63**	**88.31±1.29**	**79.17±0.27**
MVGRL	74.20±0.70	51.20±0.50	-	84.50±0.60	-	89.70±1.10	-
w/ kNN Sampler	73.30±0.34	50.70±0.36	-	82.70±0.67	-	85.08±0.66	-
w/ BatchSampler	**76.70±0.35**	**52.40±0.39**	-	**87.47±0.79**	-	**91.13±0.81**	-

* The results not reported are due to the unavailable code or out-of-memory caused by the backbone model itself.

boosts the performance of GraphCL and MVGRL, with an absolute improvement of 0.4% $\sim$ 2.9% across all the datasets. Besides, equipped with BatchSampler, DCL and HCL can achieve better performance in 12 out of 14 cases. It can also be found that BatchSampler can reduce variance in most cases, showing that the exploited hard negatives can enforce the model to learn more robust representations. We also compare BatchSampler with 11 graph classification models including the unsupervised graph learning methods [52, 55], graph kernel methods [39, 53], and self-supervised graph learning methods [19, 33, 37, 41, 51, 56, 57] (See Appendix A.2.3).

5.2 Why does BatchSampler Perform Better?

In this section, we conduct experiments to investigate why BatchSampler performs better, using the example of vision modality. We evaluate SimCLR on CIFAR10 and CIFAR100, comparing the performance and false negatives of Uniform Sampler, kNN Sampler, and BatchSampler to gain a deeper understanding of BatchSampler. Figure 4 presents the histogram of cosine similarity for all pairs within a sampled batch, and the corresponding percentage of false negatives during training.

The results indicate that although kNN Sampler can explicitly draw a data batch with similar pairs, it also leads to a substantially higher number of false negatives, resulting in a notable degradation of performance. Uniform Sampler is independent of the model so the percentage of false negatives within the sampled batch remains consistent during training, but it fails to effectively sample the hard negatives. BatchSampler can modulate M and α to control the hardness of the sampled batch, which brings about the best balance between these two sampling methods, resulting in a mini-batch of hard-to-distinguish instances with fewer false negatives compared to the kNN Sampler. Specifically, BatchSampler can sample hard mini-batch but only exhibits a slightly higher percentage of false negatives than Uniform Sampler with optimal parameter setting, which enables BatchSampler to achieve the best performance. A similar phenomenon can also be found in CIFAR100.

5.3 Empirical Criterion for BatchSampler

BatchSampler modulates the hardness of the sampled batch by two important parameters M and α to achieve better performance on three modalities. To analyze the impact of these, we vary the M and α in the range of $\{500, 1000, 2000, 4000, 6000\}$ and $\{0.1, 0.3, 0.5, 0.7\}$ respectively, and apply SimCLR, SimCSE and GraphCL as backones. We summarize the performance of BatchSampler with different M and α in Table 4. It shows that in most cases, the performance of the model peaks when $M = 1000$ but plumbs quickly with the increase of M. Such phenomena are consistent with the intuition that higher M raises the probability of selecting similar instances as neighbors,

Table 4: Empirical criterion for BatchSampler on neighbor candidate size M and restart probability α.

Modality	Dataset	Neighbor Candidate Size M					Restart Probability α				
		500	1000	2000	4000	6000	0.1	0.3	0.5	0.7	0.2~0.05
Image	CIFAR10	**92.54**	92.49	91.83	91.72	91.43	92.41	92.26	92.12	92.06	**92.54**
	CIFAR100	67.92	**68.68**	67.05	66.19	65.55	68.31	67.98	68.20	68.00	**68.68**
	STL10	84.16	**84.38**	82.80	81.91	80.92	83.01	80.69	83.93	82.56	**84.38**
	ImageNet-100	59.6	**60.8**	60.1	59.1	58.4	60.8	59.6	58.1	57.7	**60.8**
Text	Wikipedia	71.36	**76.69**	76.09	75.76	75.11	71.74	72.13	72.41	**76.69**	–
Graph	IMDB-B	**71.90±.46**	71.28±.51	71.13±.48	70.86±.56	70.68±.59	71.26±.29	71.00±.46	71.06±.21	70.78±.58	**71.90±.46**
	IMDB-M	**48.93±.28**	48.68±.35	48.88±.94	48.71±.93	48.12±.75	48.48±1.07	48.27±.67	48.72±.41	48.78±.60	**48.93±.28**
	COLLAB	70.47±.33	**71.48±.28**	70.93±.50	70.46±.28	70.24±.56	70.36±.28	70.63±.53	70.63±.54	70.31±.37	**71.48±.28**
	REDDIT-B	**90.88±.16**	89.45±.99	90.64±.38	89.92±.75	90.37±.89	90.22±.38	89.51±.61	90.44±.48	90.28±.89	**90.88±.16**

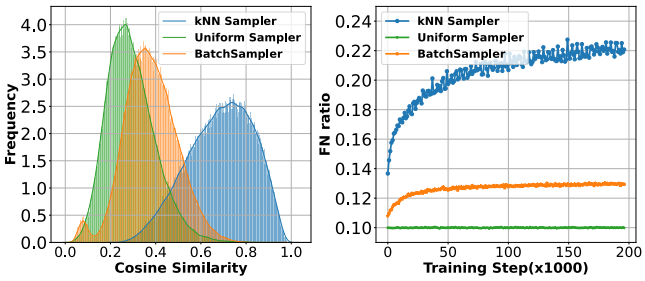

Figure 4: Cosine similarity and percentage of false negatives among various mini-batch sampling methods on CIFAR10.

but the sampler will be more likely to draw the mini-batch with false negatives, degrading the performance.

Besides, to better understand the effect of α, we visualize the histograms of cosine similarity for all pairs from a sampled batch after training and plot the corresponding percentage of false negatives during training on CIFAR10 (See Figure 5). We can observe that α moving from 0.1 to 0.7 causes cosine similarities to gradually skew left, but introduces more false negatives in the batch, creating a trade-off. This phenomenon indicates that the sampler with a higher α sample more frequently within a local neighborhood, which is more likely to yield similar pairs. A similar phenomenon can also be found on CIFAR100. However, as training progresses, the instances of the same class tend to group together, increasing the probability of collecting false negatives. To find the best balance, we linearly decay α from 0.2 to 0.05 as the training epoch increases, which is presented as $0.2 \sim 0.05$ in Table 4. It can be found that this dynamic strategy achieves the best performance in all cases except SimCSE which only trains for one epoch. Interestingly, SimCSE achieves the best performance by a large margin when $\alpha = 0.7$ since hard negatives can alleviate the distribution issue brought by the pre-trained language model. More analysis can be found in Section 5.1.

Criterion. From the above analysis, the suggested M would be 500 for the small-scale dataset, and 1000 for the larger dataset. The suggested α should be relatively high, e.g., 0.7, for the pre-trained language model-based method. Besides, dynamic decay α, e.g., 0.2 to 0.05, is the best strategy for the other methods. Such an empirical criterion provides critical suggestions for selecting the appropriate M and α to modulate the hardness of the sampled batch.

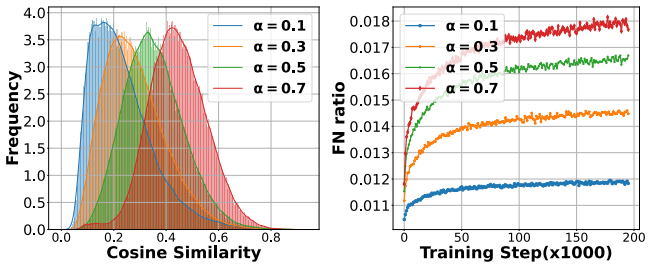

Figure 5: Cosine similarity and percentage of false negatives among various restart probabilities on CIFAR10.

5.4 Efficiency Analysis

To further investigate the efficiency of BatchSampler, we analyze the wall-clock time performance. Here, we introduce three metrics to analyze the time cost of mini-batch sampling by BatchSampler: (1) Batch Sampling Cost ($Cost_S$) is the average time of RWR taken to sample a mini-batch from a proximity graph; (2) Proximity Graph Construction Cost ($Cost_G$) refers to the time consumption of BatchSampler for constructing a proximity graph; (3) Batch Training Cost ($Cost_T$) is the average time taken by the encoder to forward and backward; (4) Proximity Graph Construction Amortized Cost ($Cost_{G/t}$) is the ratio of $Cost_G$ to the graph update interval t. The time cost of BatchSampler is shown in Table 5, from which we can make the following observations: (1) Sampling a mini-batch $Cost_S$ takes an order of magnitude less time than training with a batch $Cost_T$ at most cases. (2) Although it takes 100s for BatchSampler to construct a proximity graph in ImageNet, the cost shares across t training steps, which take only $Cost_{G/t} = 0.2$ per batch. A similar phenomenon can be found in other datasets as well. In particular, SimCSE trains for one epoch, and proximity graph is built once.

Table 5: The time cost of mini-batch sampling by BatchSampler on an NVIDIA V100 GPU.

Metric	STL10	ImageNet-100	Wikipedia	ImageNet
Cost_S	0.013s	0.015s	0.005	0.15s
Cost_G	2s	3s	79s	100s
Cost_T	0.55s	1.1s	0.08s	1.1s
$\text{Cost}_{G/t}$	0.02(t = 100)	0.03(t = 100)	0.005(t = 15625)	0.2(t = 500)

Table 6: Overall performance comparison with different graph sampling methods.

Method	Vision			Language	Graphs	
	CIFAR10	CIFAR100	STL10	Wikipedia	IMDB-B	COLLAB
BFS	91.03	65.15	77.08	74.39	70.48	69.98
DFS	92.14	68.29	83.05	73.40	71.12	70.60
RW	92.28	68.33	83.54	75.56	71.26	70.72
RWR	**92.54**	**68.68**	**84.38**	**76.69**	**71.90**	**71.48**

5.5 Further Analysis

Strategies of Proximity Graph Sampling. We conduct experiments to explore different choices of graph sampling methods, including (1) Depth First Search (DFS); (2) Breadth First Search (BFS); (3) Random Walk (RW); (4) Random Walk with Restart (RWR). Table 6 presents an overall performance comparison with different graph sampling methods. As expected, RWR consistently achieves better performance since it samples hard yet true negatives within a local cluster on proximity graph. Besides, we illustrate the histograms of cosine similarity for all pairs from a sampled batch and plot the corresponding percentage of false negatives during training in Figure 6. It can be observed that although BFS brings the most similar pairs in the mini-batch, it performs worse than the original SimCLR since it introduces substantial false negatives. While having a slightly lower percentage of false negatives than RWR, DFS, and RW do not exhibit higher performance since they are unable to collect the hard negatives in the mini-batch. The restart property allows RWR to exhibit a mixture of DFS and BFS, which can flexibly modulate the hardness of the sampled batch and find the best balance between hard negatives and false negatives.

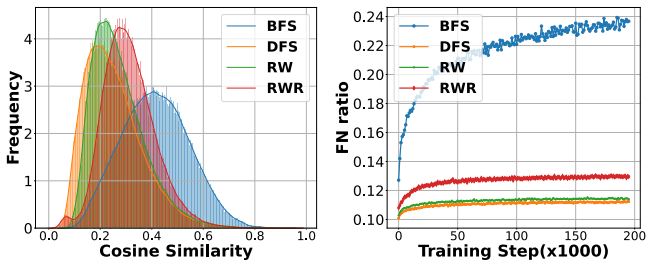

Figure 6: Histograms of cosine similarity and percentage of false negatives in a batch using different sampling methods.

Proximity Graph vs. kNN Graph. To demonstrate the effectiveness of proximity graph, we do an ablation study by replacing proximity graph with kNN graph which directly selects k neighbors with the highest scores for each instance from the whole dataset. The neighbor number k is 100 by default. The comparison results are shown in Table 7, from which we can observe that proximity graph outperforms the kNN graph by a margin. BatchSampler with kNN graph even performs worse than the original contrastive learning method because of the false negatives.

Table 7: Performance comparison of different graph construction methods.

Method	Vision		Language	Graphs	
	CIFAR10	CIFAR100	Wikipedia	IMDB-B	COLLAB
Default	92.13	68.14	75.60	70.90	70.62
kNN graph	90.47	62.67	75.13	70.10	69.96
proximity graph	**92.54**	**68.68**	**76.69**	**71.90**	**71.48**

To develop an intuitive understanding of how proximity graph alleviates the false negative issue, Figure 7 plots the changing curve of false negative ratio in a batch. The results show that the proximity graph could discard the false negative significantly: by the end of the training, kNN will introduce more than 22% false negatives in a batch, while the proximity graph brings about 13% on CIFAR10. A similar phenomenon can also be found on CIFAR100.

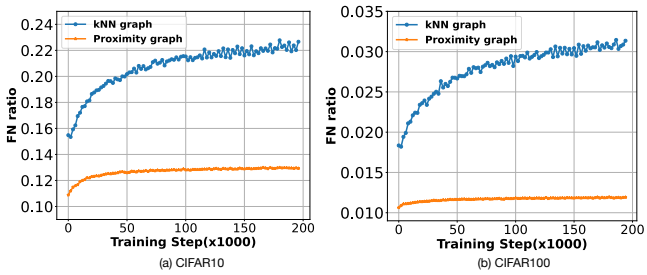

Figure 7: Percentage of false negatives using different graph building methods over the training step.

6 CONCLUSION

In this paper, we study the problem of mini-batch sampling for in-batch contrastive learning, which aims to globally sample mini-batches of hard-to-distinguish instances. To achieve this, we propose BatchSampler to perform mini-batch sampling as a walk on the constructed proximity graph. Specifically, we design the proximity graph to control the pairwise similarity among instances and leverage random walk with restart (RWR) on the proximity graph to form a batch. We theoretically and experimentally demonstrate that BatchSampler can balance kNN Sampler and Uniform Sampler. Besides, we conduct experiments with 5 representative contrastive learning algorithms on 3 modalities (e.g. vision, language, and graphs) to evaluate our proposed BatchSampler, demonstrating that BatchSampler can consistently achieve performance improvements.

Acknowledgments. This work was supported by Technology and Innovation Major Project of the Ministry of Science and Technology of China under Grant 2020AAA0108400 and 2020AAA0108402, NSF of China for Distinguished Young Scholars (61825602), NSF of China (62276148), and a research fund from Zhipu.AI.

REFERENCES

[1] Eneko Agirre, Carmen Banea, Claire Cardie, Daniel Cer, Mona Diab, Aitor Gonzalez-Agirre, Weiwei Guo, Inigo Lopez-Gazpio, Montse Maritxalar, Rada Mihalcea, et al. 2015. Semeval-2015 task 2: Semantic textual similarity, english, spanish and pilot on interpretability. In *SemEval'15*. 252–263.

[2] Eneko Agirre, Carmen Banea, Claire Cardie, Daniel M Cer, Mona T Diab, Aitor Gonzalez-Agirre, Weiwei Guo, Rada Mihalcea, German Rigau, and Janyce Wiebe. 2014. SemEval-2014 Task 10: Multilingual Semantic Textual Similarity. In *SemEval'14*. 81–91.

[3] Eneko Agirre, Carmen Banea, Daniel Cer, Mona Diab, Aitor Gonzalez Agirre, Rada Mihalcea, German Rigau Claramunt, and Janyce Wiebe. 2016. Semeval-2016 task 1: Semantic textual similarity, monolingual and cross-lingual evaluation. In *SemEval'16*.

[4] Eneko Agirre, Daniel Cer, Mona Diab, and Aitor Gonzalez-Agirre. 2012. Semeval-2012 task 6: A pilot on semantic textual similarity. In *SemEval'12*. 385–393.

[5] Eneko Agirre, Daniel Cer, Mona Diab, Aitor Gonzalez-Agirre, and Weiwei Guo. 2013. * SEM 2013 shared task: Semantic textual similarity. In *SemEval'13*. 32–43.

[6] Reid Andersen, Fan Chung, and Kevin Lang. 2006. Local graph partitioning using pagerank vectors. *FOCS'06*.

[7] Konstantin Avrachenkov, Remco van der Hofstad, and Marina Sokol. 2014. Personalized pagerank with node-dependent restart. In *WAW'14*. 23–33.

[8] Mathilde Caron, Ishan Misra, Julien Mairal, Priya Goyal, Piotr Bojanowski, and Armand Joulin. 2020. Unsupervised learning of visual features by contrasting cluster assignments. In *NIPS'20*. 9912–9924.

[9] Chih-Chung Chang and Chih-Jen Lin. 2011. LIBSVM: a library for support vector. *TIST'11* (2011), 1–27.

[10] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. 2020. A simple framework for contrastive learning of visual representations. In *ICML'20*. PMLR, 1597–1607.

[11] Xinlei Chen, Saining Xie, and Kaiming He. 2021. An empirical study of training self-supervised vision transformers. In *ICCV'21*. 9640–9649.

[12] Ching-Yao Chuang, Joshua Robinson, Lin Yen-Chen Antonio Torralba, and Stefanie Jegelka. 2020. Debiased Contrastive Learning. In *NIPS'20*.

[13] Fan Chung and Alexander Tsiatas. 2010. Finding and visualizing graph clusters using pagerank optimization. In *WAW'10*. 86–97.

[14] Fan Chung and Wenbo Zhao. 2010. PageRank and random walks on graphs. In *Fete of combinatorics and computer science*. 43–62.

[15] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL'19*. 4171–4186.

[16] Tianyu Gao, Xingcheng Yao, and Danqi Chen. 2021. Simcse: Simple contrastive learning of sentence embeddings. *EMNLP'21* (2021).

[17] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *KDD'16*. 855–864.

[18] Michael Gutmann and Aapo Hyvärinen. 2010. Noise-Contrastive Estimation: A new estimation principle for unnormalized statistical models. In *AISTATS'10*. 297–304.

[19] Kaveh Hassani and Amir Hosein Khasahmadi. 2020. Contrastive multi-view representation learning on graphs. In *International conference on machine learning*. PMLR, 4116–4126.

[20] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. 2020. Momentum contrast for unsupervised visual representation learning. In *CVPR'20*.

[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *CVPR'16*. 770–778.

[22] Jui-Ting Huang, Ashish Sharma, Shuying Sun, Li Xia, David Zhang, Philip Pronin, Janani Padmanabhan, Giuseppe Ottaviano, and Linjun Yang. 2020. Embedding-based retrieval in facebook search. In *KDD'20*. 2553–2561.

[23] Tinglin Huang, Yuxiao Dong, Ming Ding, Zhen Yang, Wenzheng Feng, Xinyu Wang, and Jie Tang. 2021. MixGCF: An Improved Training Method for Graph Neural Network-based Recommender Systems. In *KDD'21*. 665–674.

[24] Tri Huynh, Simon Kornblith, Matthew R Walter, Michael Maire, and Maryam Khademi. 2022. Boosting contrastive self-supervised learning with false negative cancellation. In *WACV'22*. 2785–2795.

[25] Ashish Jaiswal, Ashwin Ramesh Babu, Mohammad Zaki Zadeh, Debapriya Banerjee, and Fillia Makedon. 2020. A survey on contrastive self-supervised learning. *Technologies* 9, 1 (2020), 2.

[26] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data* 7, 3 (2019), 535–547.

[27] Yannis Kalantidis, Mert Bulent Sariyildiz, Noe Pion, Philippe Weinzaepfel, and Diane Larlus. 2020. Hard Negative Mixing for Contrastive Learning. In *NIPS'20*.

[28] Vladimir Karpukhin, Barlas Oğuz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. In *EMNLP'20*.

[29] Donald Ervin Knuth. 1997. *The art of computer programming: Fundamental Algorithms*. Vol. 1. Pearson Education.

[30] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692* (2019).

[31] Marco Marelli, Stefano Menini, Marco Baroni, Luisa Bentivogli, Raffaella Bernardi, and Roberto Zamparelli. 2014. A SICK cure for the evaluation of compositional distributional semantic models. In *LREC'14*. 216–223.

[32] Yu Meng, Chenyan Xiong, Payal Bajaj, Paul Bennett, Jiawei Han, and Xia Song. 2021. Coco-lm: Correcting and contrasting text sequences for language model pretraining. In *NIPS'21*.

[33] Annamalai Narayanan, Mahinthan Chandramohan, Rajasekar Venkatesan, Lihui Chen, Yang Liu, and Shantanu Jaiswal. 2017. graph2vec: Learning distributed representations of graphs. In *MLG'17*.

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

[35] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. The PageRank citation ranking: Bringing order to the web. *Stanford University Technical Report,* (1999).

[36] Veličković Petar, William Fedus, William L. Hamilton, Pietro Liò, Yoshua Bengio, and R. Devon Hjelm. 2018. Deep graph infomax. In *ICLR'19*.

[37] 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'20*. 1150–1160.

[38] Joshua Robinson, Ching-Yao Chuang, Suvrit Sra, and Stefanie Jegelka. 2021. Contrastive Learning with Hard Negative Samples. In *ICLR'21*.

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

[40] Daniel A Spielman and Shang-Hua Teng. 2013. A local clustering algorithm for massive graphs and its application to nearly linear time graph partitioning. *SIAM Journal on computing* (2013).

[41] Fan-Yun Sun, Jordan Hoffmann, Vikas Verma, and Jian Tang. 2019. Infograph: Unsupervised and semi-supervised graph-level representation learning via mutual information maximization. In *ICLR'20*.

[42] Afrina Tabassum, Muntasir Wahed, Hoda Eldardiry, and Ismini Lourentzou. 2022. Hard negative sampling strategies for contrastive representation learning. *arXiv preprint arXiv:2206.01197* (2022).

[43] Junfeng Tian, Zhiheng Zhou, Man Lan, and Yuanbin Wu. 2017. Ecnu at semeval-2017 task 1: Leverage kernel-based traditional nlp features and neural networks to build a universal model for multilingual and cross-lingual semantic textual similarity. In *SemEval'17*. 191–197.

[44] Hanghang Tong, Christos Faloutsos, and Jia-Yu Pan. 2006. Fast random walk with restart and its applications. In *ICDM'06*.

[45] Guangrun Wang, Keze Wang, Guangcong Wang, Philip HS Torr, and Liang Lin. 2021. Solving inefficiency of self-supervised representation learning. In *ICCV'21*. 9505–9515.

[46] Mike Wu, Milan Mosse, Chengxu Zhuang, Daniel Yamins, and Noah Goodman. 2020. Conditional negative sampling for contrastive learning of visual representations. In *ICLR'21*.

[47] Yawen Wu, Zhepeng Wang, Dewen Zeng, Yiyu Shi, and Jingtong Hu. 2021. Enabling on-device self-supervised contrastive learning with selective data contrast. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 655–660.

[48] Zhuofeng Wu, Sinong Wang, Jiatao Gu, Madian Khabsa, Fei Sun, and Hao Ma. 2020. Clear: Contrastive learning for sentence representation. *arXiv preprint arXiv:2012.15466* (2020).

[49] Zhirong Wu, Yuanjun Xiong, Stella X. Yu, and Dahua Lin. 2018. Unsupervised feature learning via non-parametric instance discrimination. In *CVPR'18*. 3733–3742.

[50] Lee Xiong, Chenyan Xiong, Ye Li, Kwok-Fung Tang, Jialin Liu, Paul Bennett, Junaid Ahmed, and Arnold Overwijk. 2020. Approximate nearest neighbor negative contrastive learning for dense text retrieval. In *ICLR'21*.

[51] Dongkuan Xu, Wei Cheng, Dongsheng Luo, Haifeng Chen, and Xiang Zhang. 2021. Infogcl: Information-aware graph contrastive learning. In *NIPS'21*, Vol. 34. 30414–30425.

[52] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2018. How powerful are graph neural networks?. In *ICLR'19*.

[53] Pinar Yanardag and SVN Vishwanathan. 2015. Deep graph kernels. In *KDD'15*. 1365–1374.

[54] Zhen Yang, Ming Ding, Chang Zhou, Hongxia Yang, Jingren Zhou, and Jie Tang. 2020. Understanding negative sampling in graph representation learning. In *KDD'20*. 1666–1676.

[55] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. 2018. Hierarchical graph representation learning with differentiable pooling. In *NIPS'18*, Vol. 31.
[56] Yuning You, Tianlong Chen, Yang Shen, and Zhangyang Wang. 2021. Graph contrastive learning automated. In *International Conference on Machine Learning*. PMLR, 12121–12132.
[57] Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. 2020. Graph Contrastive Learning with Augmentations. In *NIPS'20*. 5812–5823.
[58] Junchi Yu, Tingyang Xu, Yu Rong, Yatao Bian, Junzhou Huang, and Ran He. 2020. Graph information bottleneck for subgraph recognition. In *ICLR'21*.
[59] Junchi Yu, Tingyang Xu, Yu Rong, Yatao Bian, Junzhou Huang, and Ran He. 2021. Recognizing predictive substructures with subgraph information bottleneck. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021).
[60] Shaofeng Zhang, Meng Liu, Junchi Yan, Hengrui Zhang, Lingxiao Huang, Xiaokang Yang, and Pinyan Lu. 2022. M-Mix: Generating Hard Negatives via Multi-sample Mixing for Contrastive Learning. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 2461–2470.
[61] Weinan Zhang, Tianqi Chen, Jun Wang, and Yong Yu. 2013. Optimizing top-n collaborative filtering via dynamic negative item sampling. In *SIGIR'13*. 785–788.
[62] Mingkai Zheng, Fei Wang, Shan You, Chen Qian, Changshui Zhang, Xiaogang Wang, and Chang Xu. 2021. Weakly supervised contrastive learning. In *ICCV'21*. 10042–10051.
[63] Kun Zhou, Beichen Zhang, Wayne Xin Zhao, and Ji-Rong Wen. 2022. Debiased Contrastive Learning of Unsupervised Sentence Representations. In *ACL'22*.
[64] Yanqiao Zhu, Yichen Xu, Qiang Liu, and Shu Wu. 2021. An empirical study of graph contrastive learning. *arXiv preprint arXiv:2109.01116* (2021).
[65] Yanqiao Zhu, Yichen Xu, Feng Yu, Qiang Liu, Shu Wu, and Liang Wang. 2021. Graph contrastive learning with adaptive augmentation. In *Proceedings of the Web Conference 2021*. 2069–2080.
[66] Jiří Šíma and Satu Elisa Schaeffer. 2006. On the np-completeness of some graph cluster measures. In *International Conference on Current Trends in Theory and Practice of Computer Science*.

A APPENDICES

A.1 Algorithm Details

Algorithm 2: Proximity Graph Construction

Input: Dataset $\mathcal{D} = \{x_i\}$, Candidate set size M, Neighbor number K;
Output: A proximity graph G;
for v *in* $\mathcal{D}$ **do**
  Randomly select M neighbor candidates from $\mathcal{D}$;
  Select the K closest candidates $\mathcal{N}_v$ by Eq. 3;
  $G[v] \leftarrow \mathcal{N}_v$;
end
return G

Algorithm 3: Random Walk with Restart(RWR)

Input: Proximity graph $G = \{\mathcal{V}, \mathcal{E}\}$, seed node u, restart probability α, number of sampled node B;
Output: A sampled node set $\mathcal{S}$;
$\mathcal{S} \leftarrow \{\}, v \leftarrow u$;
while $len(\mathcal{S}) < B$ **do**
  if v *not in* $\mathcal{S}$ **then**
    $\mathcal{S}$.insert(v)
  end
  Sample r from Uniform distribution $U(0, 1)$;
  if $r < \alpha$ **then**
    $v \leftarrow u$;
  end
  else
    Randomly sample $\hat{v}$ from v's neighbors;
    $v \leftarrow \hat{v}$;
  end
end
return $\mathcal{S}$

A.2 Additional Experiments

A.2.1 ***Extensive studies on Vision.*** Here we evaluate the BatchSampler on two small-scale (CIFAR10, CIFAR100) and two medium-scale (STL10, ImageNet-100) benchmark datasets, and equip DCL [12] and HCL [38] with BatchSampler to investigate its generality. Experimental results in Table 8 show that BatchSampler can consistently improve SimCLR and its variants on all the datasets, with an absolute gain of 0.3%∼2.5%. We also can observe that the improvement is greater on medium-scale datasets than on small-scale datasets.

A.2.2 ***Experiments with Roberta on Language.*** We also apply BatchSampler to the SimCSE with the pretrained RoBERTa [30], and present the results in Table 9. Similar to the results of BERT, BatchSampler can consistently improve the performance of the baseline model. Besides, as discussed in Section 5.1, the hard negative sampled by BatchSampler explicitly can alleviate the low distribution gap between positive score and negative score distribution caused by the pretrained language model, alleviating the performance degradation of DCL and HCL.

Table 8: Overall performance comparison on image classification task in terms of Top-1 Accuracy.

Method	CIFAR10	CIFAR100	STL10	ImageNet-100
SimCLR	92.13	68.14	83.26	59.30
w/ kNN Sampler	90.16	62.30	79.25	57.70
w/ BatchSampler	**92.54**	**68.68**	**84.38**	**60.80**
DCL	92.28	68.52	84.92	59.90
w/ BatchSampler	**92.74**	**68.91**	**86.39**	**60.14**
HCL	92.39	68.92	88.20	60.60
w/ BatchSampler	**92.41**	**69.13**	**89.49**	**61.50**

Table 9: Performance comparison for sentence embedding learning based on RoBERTa.

Method	STS12	STS13	STS14	STS15	STS16	STS-B	SICK-R	Avg.
SimCSE-RoBERTa$_{base}$	67.90	80.91	73.14	80.58	80.74	80.26	69.87	76.20
w/ kNN Sampler	68.78	79.49	73.34	81.05	80.15	77.09	67.18	75.30
w/ BatchSampler	**68.29**	**81.96**	**73.86**	**82.16**	**80.94**	**80.77**	**69.30**	**76.75**
DCL-RoBERTa$_{base}$	**66.60**	79.16	**71.05**	80.40	77.76	**77.94**	**67.57**	74.35
w/ kNN Sampler	65.39	79.04	69.71	78.37	75.98	74.72	64.39	72.51
w/ BatchSampler	65.53	**80.09**	71.00	**80.64**	**78.35**	77.75	67.52	**74.41**
HCL-RoBETa$_{base}$	**67.20**	80.47	72.44	80.88	80.57	78.79	67.98	75.49
w/ kNN Sampler	65.99	77.32	73.71	80.59	79.78	77.70	65.40	74.36
w/ BatchSampler	66.01	**80.79**	**73.58**	**81.25**	**80.66**	**79.22**	**68.52**	**75.72**

A.2.3 ***Comparison with baselines on Graphs.*** In Table 10, we compare different kinds of baselines on graph classification tasks, including the unsupervised graph learning methods [52, 55], graph kernel methods [39, 53], and self-supervised graph learning methods [19, 33, 37, 41, 51, 56, 57]. BatchSampler can consistently improve the performance of both GraphCL and MVGRL on all the datasets, demonstrating the effectiveness of global hard negatives. Benefiting from the performance gain brought by BatchSampler, MVGRL achieves the best performance on 4 datasets, outperforming InfoGCL and supervised methods.

A.2.4 ***Impact of Sampling Method on CIFAR100.*** We conduct different strategies of proximity graph sampling on CIFAR100 to investigate the impact of sampling methods. As shown in Figure 8, BFS achieves more similar pairs in the sampled batch but it introduces a higher percentage of false negatives, which obviously degrades the downstream performance. Conversely, DFS explores paths far away from the selected central node, which can not guarantee that the sampled path (i.e. batch) is within a local cluster. Thus, we theoretically leverage RWR to flexibly modulate the hardness of the sampled batch and achieve a balance between hard negatives and false negatives.

A.3 Parameter Analysis

A.3.1 ***Batchsize*** *B.* To analyze the impact of the batchsize B, we vary B in the range of {16, 32, 64, 128, 256} and summarize the results in Table 12. In vision, a larger batchsize leads to better

Table 10: Experiment results for graph classification task under LIBSVM [9] classifier.

Method	Dataset	IMDB-B	IMDB-M	COLLAB	REDDIT-B	PROTEINS	MUTAG	NCI1
Supervised	GIN	75.1±5.1	52.3±2.8	80.2±1.9	92.4±2.5	76.2±2.8	89.4±5.6	82.7±1.7
	DiffPool	72.6±3.9	-	78.9±2.3	92.1±2.6	75.1±3.5	85.0±10.3	-
Graph Kernels	WL	72.30±3.44	46.95±0.46	-	68.82±0.41	72.92±0.56	80.72±3.00	80.31±0.46
	DGK	66.96±0.56	44.55±0.52	-	78.04±0.39	73.30±0.82	87.44±2.72	80.31±0.46
Self-supervised	graph2vec	71.10±0.54	50.44±0.87	-	75.78±1.03	73.30±2.05	83.15±9.25	73.22±1.81
	infograph	73.03±0.87	49.69±0.53	70.65±1.13	82.50±1.42	74.44±0.31	89.01±1.13	76.20±1.06
	JOAO	70.21±3.08	49.20±0.77	69.50±0.36	85.29±1.35	74.55±0.41	87.35±1.02	78.07±0.47
	GCC	72.0	49.4	**78.9**	89.9	-	-	-
	InfoGCL	75.10±0.90	51.40±0.80	-	-	-	**91.20±1.30**	-
	GraphCL	70.90±0.53	48.48±0.38	70.62±0.23	90.54±0.25	74.39±0.45	86.80±1.34	77.87±0.41
	GraphCL + BatchSampler	71.90±0.46	48.93±0.28	71.48±0.28	**90.88±0.16**	**75.04±0.67**	87.78±0.93	78.93±0.38
	MVGRL	74.20±0.70	51.20±0.50	-	84.50±0.60	-	89.70±1.10	-
	MVGRL + BatchSampler	**76.70±0.35**	**52.40±0.39**	-	87.47±0.79	-	91.13±0.81	-

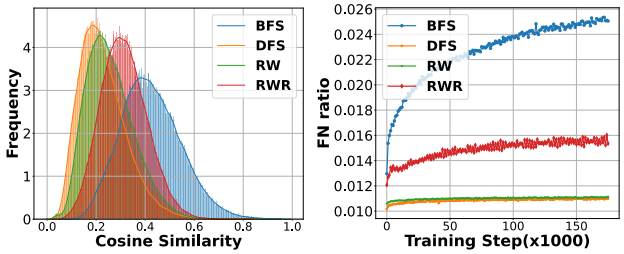

Figure 8: Histograms of cosine similarity and Percentage of false negative of all pairs in a batch.

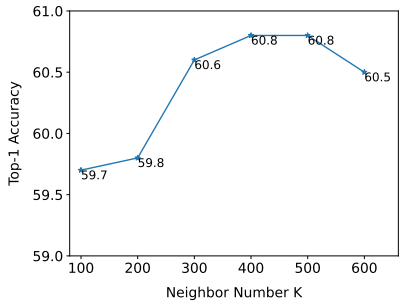

Figure 9: Impact of neighbor number K.

Table 11: Performance comparison with different t.

	Update Interval t	50	100	200	400
Vision	CIFAR10	92.29	**92.54**	92.34	92.26
	CIFAR100	68.37	**68.68**	67.83	68.59
	Update Interval t	10	25	50	100
Graphs	IMDB-B	71.30	**71.90**	71.40	71.10
	COLLAB	71.06	70.36	**71.48**	70.62

Table 12: Performance comparison of different batchsize B.

B	Vision			Language	Graphs	
	CIFAR10	CIFAR100	STL10	Wikipedia	IMDB-B	COLLAB
16	79.93	46.69	56.31	76.26	71.50	71.32
32	84.64	56.24	68.61	74.16	71.60	71.40
64	89.09	61.30	74.24	76.69	71.65	71.42
128	91.03	65.96	82.56	76.64	71.83	**71.48**
256	**92.54**	**68.68**	**84.38**	76.11	**71.90**	71.35

results, which is consistent with the previous studies [10, 20, 27]. In language domain, BatchSampler reaches its optimum at $B = 64$, which aligns with the results in SimCSE [16]. For graphs, the performance improves slightly with increasing batch size.

A.3.2 ***Impact of Neighbor Number*** K. In Figure 9, we investigate the impact of the neighbor number K on ImageNet-100 dataset with the default BatchSampler setting. We observe an absolute improvement of 1.1% with the increasing size of neighbors. Specifically, model achieves an absolute performance gain of 0.9% from $K = 100$ to $K = 300$, while only obtaining 0.2% from $K = 300$ to $K = 500$, demonstrating that sampling more neighbors increases the scale of proximity graph and urges BatchSampler to explore smaller-scope local clusters, leading to a significant improvement in performance. However, performance degrades after reaching the optimum, because larger K introduces more easy negatives.

A.3.3 ***Proximity Graph Update Interval*** t. To analyze the impact of update interval t, we vary t in the range of {50,100,200,400} for vision and {10,25,50,100} for graphs respectively. Table 11 summarizes the experimental results on different t. Update intervals that are too short or too long will degrade the performance. One reason is that sampling on a proximity graph that is frequently updated results in unstable learning of the model. Besides, the distribution of instances in the embedding space will change during the training process, resulting in a shift in hard negatives. As a result, after a few iterations, the lazy-updated graph cannot adequately capture the similarity relationship.