Negative Sampling Strategies for Contrastive Self-Supervised Learning of Graph Representations

Hakim Hafidi[a,b], Mounir Ghogho[a], Philippe Ciblat[b], Ananthram Swami[c]

[a] *TICLab, College of Engineering and Architecture, Université Internationale de Rabat, Morocco*
[b] *LTCI, Telecom Paris, Institut Polytechnique de Paris, France*
[c] *United States Army Research Laboratory, Adelphi, Maryland, USA*

Abstract

Contrastive learning has become a successful approach for learning powerful text and image representations in a self-supervised manner. Contrastive frameworks learn to distinguish between representations coming from augmentations of the same data point (*positive* pairs) and those of other (*negative*) examples. Recent studies aim at extending methods from contrastive learning to graph data. In this work, we propose a general framework for learning node representations in a self supervised manner called Graph Constrastive Learning (GraphCL). It learns node embeddings by maximizing the similarity between the nodes representations of two randomly perturbed versions of the same graph. We use graph neural networks to produce two representations of the same node and leverage a contrastive learning loss to maximize agreement between them. We investigate different standard and new negative sampling strategies as well as a comparison without negative sampling approach. We demonstrate that our approach significantly outperforms the state-of-the-art in unsupervised learning on a number of node classification benchmarks in both transductive and inductive learning setups.

Key words: Graph Neural Network, Contrastive Learning, Self-Supervised Learning, Node Classification.

[1]The main ideas of this paper have been posted in July 2020 on Arxiv with reference *arXiv:2007.08025*

1. Introduction

In many fields, the rapid increase in data volume and the complexity of its structure/representation make it difficult to exploit it effectively. Graphs offer a unified framework for aligning well-structured and unstructured data. However, graphs have long been poorly leveraged because of their complexity, and limited approaches relying on content associated with nodes and links. Recently, graph representation learning has attracted the attention of the scientific community as a way of analysing graphs and helping to exploit the richness of information that resides in poor-structured data. Graphs are characterized by a set of nodes, which represent the entities, and a set of links connecting them, representing relationships between the nodes. Nodes may be of different types, and may further be associated with several features. And links may represent different relationships and may also be associated with different attributes or semantic content. One of the major challenges facing graph representation learning is learning node embeddings which capture both node features and graph structure. These representations can then be fed into downstream machine learning models.

Most successful approaches for graph representation have been great efforts to generalize neural networks to graph data and fall under the umbrella of Graph Neural Networks (GNNs) or Deep Geometric Learning [1–4]. These approaches have achieved remarkable results in a number of important tasks such as node classification [5–7] and link prediction [8, 9]. However, these methods are very reliant on human annotation and suffer from the necessity of some form of supervision. This requires high cost, expert knowledge in the domain and the use of annotated data, which is not often available. Hence, it is of importance to develop methods capable of learning representations in an unsupervised manner.

In order to compensate for the absence of labels or predefined tasks, some unsupervised methods have adopted the homophily hypothesis, which states that linked nodes should be nearby in the embedding space [10]. Inspired by the Skipgram algorithm for embedding words into a latent space, where adjacent vectors correspond to co-occurring words in a sentence [11], a majority of

these methods use random walks to generate sentence-like sequences where co-occurring nodes are close to one another in the embedding space [12, 13] and can also be adapted to heterogeneous graphs [14–16]. Other methods, such as autoencoders, also employ the homophily hypothesis by reconstructing either the adjacency matrix or the neighborhood of a node [8, 17]. Despite their success in learning relatively powerful representations, relying on the homophily hypothesis may bias these methods towards emphasizing the direct proximity of nodes over topological information [17]. More recently, [18] proposed Deep Graph Infomax (DGI) that learns representations by training a discriminator to distinguish between representations of nodes that belong to the graph from nodes that belong to a corrupted graph. Leveraging recent advances in unsupervised visual representations [19], the success of DGI has been attributed to the maximization of mutual information between global and local parts of the input. This requires learning global representations of the entire graph which can be very costly and even intractable when dealing with large graphs.

To overcome the above-mentioned challenges, we here propose a contrastive framework for self-supervised learning of nodes' representations, called GraphCL. We take inspiration from the success of contrastive losses in learning meaningful representations of images [20, 21] and develop a model that learns node embeddings by maximizing the similarity between the representations of two randomly perturbed versions (views) of the intrinsic features and link structure of the same node's local subgraph. The perturbation consists of randomly dropping from its L-hop subgraph, a subset of edges and nodes' intrinsic features. Other researchers have also used the contrastive loss to learn nodes or graph representations using different augmentation (perturbation) strategies. In [22], the authors used the diffusion matrix as a second view of the graph. In [23], the authors used four different strategies consisting of dropping nodes, perturbing edges, masking attributes or sampling subgraphs.

Contrastive learning is a special case of Siamese networks, which are weight-sharing neural networks applied to two or multiple inputs. Recent approaches use augmentations of the same data point as inputs and maximize the similarity

between the learned representations of the two inputs. Maximizing the similarity between each pair of augmented data points in the dataset can lead to a trivial solution. Since we want representations of all pairs of augmented views to be equal, a possible solution is to map all nodes to a single point (representation). This is what we call a collapsing of representations to a single data point (i.e. if all representations are the same, then so are those of each pair of augmented views). Contrastive learning is one way of preventing this undesirable solution. It does so by contrasting between *positive* (similar) examples and *negative* (dissimilar) examples. The objective of the training phase is to map positive examples to nearby locations in the destination (representation) space while pushing away negative examples often by using *noise-contrastive estimation* [24]. One key component of contrastive learning frameworks is the choice of negative examples. The most common strategy is to uniformly sample from the training dataset using examples either from the current batch or from a memory bank. It has been shown that these approaches require large batches or memory banks to perform well for visual representation [20, 21]. To improve the performance and efficiency of contrastive frameworks, recent studies have proposed novel sampling strategies. Most strategies are based on the assumption that hard negative examples (i.e. examples that are hard to distinguish from a positive pair) are beneficial in learning more powerful representations. In [25], authors use hard negative mixing to synthesize new examples from the available hard negatives. In [26], the authors sample negatives from a *ring* around each positive (i.e. they sample negatives that are neither too close nor too far from the positive example).

More recent Siamese network architectures preventing representation collapsing still rely on the use of pairs of positive examples only. In [27], the authors experimentally show that the learned representations of their proposed framework do not collapse when using a momentum network even when not using negative examples. In [28], the authors avoid the collapsing phenomenon by simply using a stop-gradient strategy when directly maximizing the similarity between two augmented views. The stop-gradient strategy consists of consid-

ering the representation of one of the augmented views as a constant when updating the network parameters.

While these methods have shown surprisingly good results when applied to image datasets, it is not clear whether they are easily generalizable to non-Euclidean data such as graphs. In this work, we introduce novel sampling strategies of negative examples based on the graph structure and show that our approach improves the performance of the learned representations on downstream classification tasks and outperforms existing methods. In addition, we conduct extensive experiments to study the different components of our Siamese network-based approach for learning nodes' representations which enable us to answer the following questions:

- Is a larger set of negative examples always useful in learning good representations?
- Does sampling hard negative examples improve the quality of the representation?
- Can we train a Siamese neural network to learn nodes' representations without using negative examples?

2. Background

2.1. Problem formulation.

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be an undirected graph where $\mathcal{V}$ is a set of nodes and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is a set of edges. Each node $u \in \mathcal{V}$ is represented by a feature vector $x_u \in \mathbb{R}^P$. An adjacency matrix $A \in \mathbb{R}^{N \times N}$ represents the topological structure of the graph where $N = |\mathcal{V}|$ is the number of nodes in the graph. Without loss of generality we assume the graphs to be unweighted i.e $A_{u,v} = 1$ if $(u, v) \in \mathcal{E}$ and $A_{u,v} = 0$ otherwise. We are also provided with a matrix $X \in \mathbb{R}^{N \times P}$ that summarizes the intrinsic feature vectors of all nodes.

Our objective is to learn an effective representation of nodes without human annotation. This will be done through the learning of a graph neural network

encoder f that maps both node original feature and the graph structure to a higher level representation i.e. $f(X, A) = H^{(L)} \in \mathbb{R}^{N \times P'}$, where P' is the embedding size. The $u-$th row of $H^{(L)}$ corresponds to the embedding $h_u^{(L)}$ of node u. In the remainder of the paper, h_u refers to the output of the GNN's last layer, i.e. $h_u = h_u^{(L)}$.

2.2. Graph Neural Networks (GNNs).

GNNs are a class of graph embedding architectures which use the graph structure in addition to node and edge features to generate a representation vector (i.e., embedding) for each node. Recent GNNs learn node representations by aggregating the features of neighboring nodes and edges. The output of the l-th layer of these GNNs is generally expressed as

$$h_u^{(l)} = COMBINE^{(l)}(h_u^{(l-1)}, AGGREGATE^{(l)}(\{(h_u^{(l-1)}, h_v^{(l-1)}) : v \in \mathcal{N}(u)\})), \tag{1}$$

where $h_u^{(l)}$ is the feature vector of node u at the l-th layer initialized by $h_u^{(0)} = x_u$ and $\mathcal{N}(u)$ is the set of first-order neighbors of node u. According to Eq. (1), $h_u^{(L)}$ corresponds to the output of the last layer of the GNN, which involves the nodes of node u's L-hop subgraph. Different GNNs use different formulations of the COMBINE and AGGREGATE functions; the ones used in this work are described in subsection 4.1.3.

2.3. Contrastive learning

In this work, we consider the dictionary look up formulation of contrastive learning, which means that considering a query h_q, a corresponding positive pair h_q^+ and a set of negative examples Q_q^-, a contrastive loss is a function which has a low value when h_q is similar to h_q^+ and dissimilar to all elements of Q_q^-. A successful and widely used form of contrastive loss is defined as:

$$\mathcal{L}_{h_q, h_q^+, Q_q^-} = -\log \frac{\exp(h_q^\top h_q^+ / \tau)}{\exp(h_q^\top h_q^+ / \tau) + \sum_{h_n \in Q_q^-} \exp(h_q^\top h_n / \tau)}, \tag{2}$$

where τ is a temperature hyperparameter and h_q, h_q^+ and all h_n in Q_q^- are L_2 normalized feature vectors. The final loss is summed accross all queries q

belonging to the dataset $\mathcal{D}$ and can be expressed as follows when scaled by the temperature τ [29]:

$$\mathcal{L} = -\frac{1}{\tau|\mathcal{D}|}\sum_{q\in\mathcal{D}} h_q^\top h_q^+ + \frac{1}{|\mathcal{D}|}\log\sum_{q\in\mathcal{D}}\left(\exp(h_q^\top h_q^+/\tau) + \sum_{h_n\in Q_q^-}\exp(h_q^\top h_n/\tau)\right), \tag{3}$$

where $|\mathcal{D}|$ is the number of elements in $\mathcal{D}$.

In Contrastive learning framework,, different negative sampling strategies (i.e., the way to build Q_q^-) may be employed to avoid collapsing of the contrastive loss optimization problem into a unique representation of all samples.

2.4. Simple Siamese neural networks for nodes representation

In [28], the authors argue that their approach can prevent collapsing when maximizing the similarities between the representations of two views of the same image without the use of negative examples. Their approach works by sampling two views of x_1 and x_2 of the same image x which they process using an encoder f and a multi-layer perceptron (MLP) prediction head g. Letting $p_1 = g(f(x_1))$, $p_2 = g(f(x_2))$, $h_1 = f(x_1)$ and $h_2 = f(x_2)$ denote respectively the outputs of the MLP prediction and the encoder, the objective is to minimize the symmetric negative cosine similarity loss which is defined as:

$$\mathcal{L} = \frac{1}{2}\mathrm{S}(p_1, \mathrm{stopgrad}(h_2)) + \frac{1}{2}\mathrm{S}(p_2, \mathrm{stopgrad}(h_1)), \tag{4}$$

where $\mathrm{S}(p_1, h_2) = -\frac{p_1}{\|p_1\|}.\frac{h_2}{\|h_2\|}$ and the stopgrad operation consists of treating h_2, respectively h_1, as constant when updating the models' parameters.

3. Methodology of the proposed approach

3.1. GraphCL

GraphCL's objective is to learn node representations by maximizing the similarity between two embeddings of the same node. The two embeddings are obtained from applying a GNN encoder to two perturbed versions of the graph. This framework has three main components: a stochastic perturbation, a GNN

based encoder and a contrastive loss function. We first introduce each of these components, and then give a high-level overview of the proposed method.

- **Stochastic perturbation.** We apply two stochastic perturbations to the graph which allow us to obtain two representations of the same node which we consider as positive examples. In this work, we consider simultaneous transformations of both node features and the connectivity of the graph. The graph structure is transformed by randomly dropping edges using samples from a Bernoulli distribution. For the node's original features, we apply a similar strategy by simply applying dropout to the input features;

- **Graph neural network encoder.** We apply a GNN based encoder that learns representations of all nodes in the graph. Our framework supports several choices of GNN architectures. Details about the choices of architectures are given in section 4.1.3.

- **Contrastive loss function.** We define a *pretext* prediction task that aims at identifying the corresponding positive example h_q^+ of a representation h_q given a set of generated examples, with h_q and h_q^+ being a positive pair of examples (i.e. obtained from the GNN representations of two transformations of the graph). As for the negative examples generation, details are provided in subsection 3.2.

3.2. Negative sampling strategies

Negative sampling has been shown to be a key ingredient for the success of contrastive learning frameworks. Different strategies have been proposed to build negatives examples for visual presentations [20, 21, 25, 26]. First, we investigate whether the conclusions that have been drawn from the most successful approaches of visual representations are still valid when applied to graphs. We hereafter introduce three negative sampling strategies: the two first are standard while the third one is new and well adapted to graph.

- **Random sampling.** This approach consists of considering the samples of the current (randomly generated) mini-batch as negatives. The problem

with this approach is the number of negative examples is limited by the size of the mini-batch which is limited by the memory of the GPU. An alternative would be to randomly sample negatives from a memory bank that contains either representations of the whole training set or a queue with representations of the last few batches.

- **Feature-based sampling.** In [26], the authors propose to pick two percentiles ω_k and $\omega_l \in [0, 100]$ and considering h_{n_c} as a negative example for a representation of a query h_q if and only if $h_q^\top h_{n_c}$ is within the ω_k-th to the ω_l-th percentile of all $h_n \in Q_q^-$.This enables to build easily hard negative examples (i.e., negatives that are hard to distinguish from the current sample) which are beneficial in learning powerful representations as mentioned in [30, 31]. To adapt this method to the graph setting, instead of considering the similarities in the representation space, which requires using the encoder to learn the representations of all nodes in the graph, we simply consider the similarities of the nodes' original features. For each node u, we consider as negatives all nodes v whose original features are neither too close nor too far from those of node u (i.e. $x_u^\top x_v / \|x_u\| \|x_v\|$ is within the ω_k-th to the ω_l-th percentile of all nodes of the graph).

- **Graph-based sampling.** Using original feature similarities as a negative sampling strategy requires computing similarities between each pair of nodes in the graph then sorting them and fine tuning the model to select the best values for the percentiles ω_k and ω_l. To avoid this, we propose to make use of the graph structure information to select negatives. Instead of considering distances between the nodes' original features, we use the distance between the nodes on the graph. For each node u, we simply sample negatives from its l-th order neighbors.

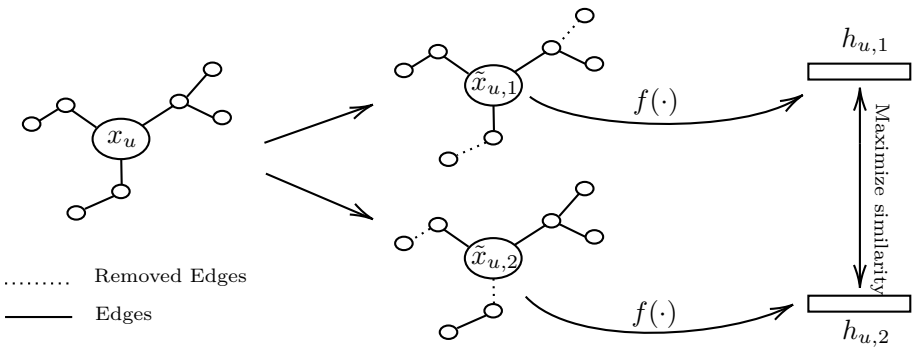


Figure 1: A high-level overview of our method for a subgraph around node u. $h_{u,1}$ and $h_{u,2}$ form a positive pair with a query $h_q = h_{u,1}$ and its corresponding key $h_q^+ = h_{u,2}$.

3.3. Overview of GraphCL

The training algorithm of GraphCL is summarized in the following steps:

1. Draw two stochastic perturbations t_1 and t_2 as defined in section 3.1 and illustrated in Figure 1. Apply them to nodes' original features and the graph structure:
   - $(\widetilde{X}_1, \widetilde{A}_1) \sim t_1(X, A)$
   - $(\widetilde{X}_2, \widetilde{A}_2) \sim t_2(X, A)$
2. Apply the encoder to both views of the graph:
   - $H_1^L = f(\widetilde{X}_1, \widetilde{A}_1)$
   - $H_2^L = f(\widetilde{X}_2, \widetilde{A}_2)$
3. Select negative examples as suggested in subsection 3.2.
4. Update parameters of the encoder f using the loss function defined in Eq. (3).

3.4. Extension to inductive setup

Unlike the transductive setup where we have access to the whole graph and features of all nodes of the graph during training time. In the inductive setup, the objective is to generate representations of nodes that were not used when training the model. These previously unseen nodes can be new nodes in an

evolving graph such as a social network, we refer to this by the single graph inductive setup. These nodes can also come from previously unseen graphs which helps generalization across graphs with the same form of features, we refer to this by the multiple graph inductive setup.

GraphCL can be easily extended to both setups. The extension to the multiple graph setup is straightforward, as it consists of training the encoder on each of the available graphs for the training and using the learnt encoder to produce representations of nodes of the new graphs. For inductive learning on large graphs, we train the encoder by sampling minibatches of nodes. The training algorithm of GraphCL for the inductive setup is summarized in the following steps for each sampled minibatch $\mathcal{B}$:

1. For each node u in the minibatch we define (X_u, A_u) as the subgraph containing all nodes and edges that are at most L-hops from u in the graph and their corresponding features;
2. Draw two stochastic perturbations t_1 and t_2 as defined in section 3.1 and apply them to u's L-hop neighborhood subgraph:
   - $(\widetilde{X}_{u,1}, \widetilde{A}_{u,1}) \sim t_1(X_u, A_u)$
   - $(\widetilde{X}_{u,2}, \widetilde{A}_{u,2}) \sim t_2(X_u, A_u)$
3. Apply the encoder to the two representations of node u:
   - $h_{u,1} = f(\widetilde{X}_{u,1}, \widetilde{A}_{u,1})$
   - $h_{u,2} = f(\widetilde{X}_{u,2}, \widetilde{A}_{u,2})$
4. Update parameters of the encoder.

4. Experiments

We evaluate the effectiveness of GraphCL representations on both tranductive and inductive learning setups. The transductive learning setup consists of embedding nodes from a fixed graph (i.e. all node features and the entire graph structure are known during training time). On the other hand, the inductive learning setup consists of generating representation of unseen nodes or new

graphs. Following common practice, we opt for a linear evaluation of the learned node representations. Specifically, we use these representations to train a logistic regression model to solve multiclass node classification tasks on five well-know benchmark datasets, three for the transductive learning setup and two for the inductive setup. We summarize the datasets and the baselines respectively in sections 4.1.1 and 4.1.2, provide model configuration and implementation details in section 4.1.3, and discuss the results in section 4.2.

4.1. Experimental setup

4.1.1. Datasets

For the transductive setting, we utilize Cora, Citeseer and Pubmed [32], three citation networks where nodes are bag of words representations of documents and edges correspond to (undirected) citations. Each node belongs to one class. We also use ogbn-arxiv, which is another citation network, where nodes are computer science papers represented by a 128-dimensional feature vector obtained by averaging the embeddings of words in its title and abstract. Each node belongs to one of forty subject areas of arXiv CS papers [33].

On the other hand, a protein-protein interaction dataset (PPI) is used for the inductive setting on multiple graphs [34]. It consists of multiple graphs corresponding to different human tissues where node features are the positional gene sets, motif gene sets and immunological signatures. Each node has several labels among 121 labels from the gene ontology. For the inductive setting on large graphs, we use a Reddit dataset [5]. It represents a large social network where nodes correspond to Reddit posts (i.e. represented by their GloVe embedding [35]) and edges connecting two posts mean that the same user commented on them. Labels are the posts' *subreddit* and the objective is to predict the community structure of the social network.

Statistics of the datasets including data splits are given in table 1. For ogbn-arxiv dataset, we follow recommendations from the Open Graph Benchmark initiative and adopt a data split that is based on the publication dates of the papers [36]. More precisely, we train on papers published until 2017, validate

Dataset	Task	Nodes	Edges	Features	Classes	Train/Val/Test Nodes
Cora	Transductive	2,708	5,429	1,433	7	140/500/1,000
Citeseer	Transductive	3,327	4,732	3,707	6	120/500/1,000
Pubmed	Transductive	19,717	44,338	500	3	60/500/1,000
ogbn-arxiv	Transductive	169,343	1,166,243	128	40	Time
Reddit	Inductive	231,443	11,606,919	602	41	151,708/23,699/55,334
PPI	Inductive	56,944 (24 graphs)	818,716	50	121 (multilabel)	44,906/6,154/5,524 (20/2/2 graphs)

Table 1: Description of datasets

on those published in 2018, and test on those published since 2019.

4.1.2. Baselines

For the transductive learning tasks, we use four unsupervised methods for comparison: Label Propagation (LP) [37], DeepWalk [12], Embedding Propagation (EP-B) [38], and Deep Graph Infomax (DGI) [18]. We also report the results of training logistic regression on the intrinsic input features only, and also on the concatenation of DeepWalk embeddings and the nodes' intrinsic features. Aside from unsupervised methods, we also compare our approach to strong supervised baselines, Graph Convolution Networks (GCN) [2].

For the inductive learning tasks, in addition to DeepWalk and DGI, we compare GraphCL with the unsupervised GraphSAGE methods [5]. We also provide results of two supervised approaches, FastGCN [39] and Gated Attention Networks (GaAN) [40].

4.1.3. Model configurations

Eq. (1) provides a general formulation of graph neural networks. Several architectures have been proposed for the choice of $AGGREGATE$ and $COMBINE$. In all our experiments the basic update rule is the mean pooling variant from [5].

$$h_u^{(l)} \leftarrow \left(W^{(l-1)}\right)^\top \cdot MEAN(\{h_u^{(l-1)}\} \cup \{h_v^{(l-1)}, \forall v \in \mathcal{N}(u)\}), \tag{5}$$

where the $MEAN$ operator is the element-wise mean of all vectors in $(\{h_u^{(l-1)}\} \cup \{h_v^{(l-1)}, \forall v \in \mathcal{N}(u)\})$, and $W^{(0)} \in \mathbb{R}^{P \times P'}$ and $W^{(l-1)} \in \mathbb{R}^{P' \times P'}$, for $l > 1$, are

learnable linear transformations.

All GNN aggregation operations are computed in parallel resulting in a matrix representation as follows:

$$H^{(l)} = \hat{A}H^{(l-1)}W^{(l-1)} \tag{6}$$

where $H^{(l)} = [h_1^{(l)}, h_2^{(l)}, \ldots, h_N^{(l)}]^\top$ is the matrix of nodes' hidden feature vectors at the $l-$th layer and $\hat{A} = \check{D}^{-1}\check{A}$ is the normalized version of the adjacency matrix with added self-loop $\check{A} = A + I_N$ with $\check{D}$ being its diagonal degree matrix, i.e. $\check{D}_{ii} = \sum_j \check{A}_{ij}$. We also consider the symmetrically normalized version of the adjacency matrix where $\hat{A} = \check{D}^{-\frac{1}{2}}\check{A}\check{D}^{\frac{1}{2}}$. We refer to encoders using this variant by GCN when needed.

Transductive learning. For Citeseer and Pubmed, we use a one layer GNN as defined in Eq. (6), the encoder is then simply expressed as:

$$f(X, A) = \hat{A}XW^{(0)} \tag{7}$$

For Cora, our encoder is a two-layer GNN:

$$f(X, A) = \hat{A}\sigma(\hat{A}XW^{(0)})W^{(1)} \tag{8}$$

where σ is an exponential linear unit [41], and $f(X, A)$ is the concatenation of all nodes' embeddings. In each layer, we compute $P' = 512$ features resulting in a node embedding size of 512. For the larger ogbn-arxiv dataset, we use a three-layer GNN, and train the model by randomly sampling 1024 negative examples for each node.

Inductive learning. For both inductive learning setups on large graphs and on multiple graphs , we use a three-layer mean-pooling encoder with residual units as follows:

$$H^{(1)} = \sigma(\hat{A}XW_1^{(0)} + XW_2^{(0)}) \tag{9}$$

$$H^{(2)} = \sigma(\hat{A}H^{(1)}W_1^{(1)} + H^{(1)}W_2^{(1)}) \tag{10}$$

$$f(X, A) = \hat{A}H^{(2)}W_1^{(2)} + H^{(2)}W_2^{(2)} \tag{11}$$

We set the hidden layers and the embedding size to $P' = 512$ and apply RELU as an activation function.

For the multiple-graph setting, we sample one graph at a time from the training set to train the contrastive loss function. For the single graph inductive setup, the scale of the dataset makes it impossible to fit into GPU memory. We therefore adopt the sub-sampling strategy of [5]. We first select a minibatch of nodes and construct for each of them their L-hop neighborhood subgraph by sampling a fixed size neighborhood. We sample 10 nodes in each of the three levels resulting in $1 + 10 + 100 + 1000 = 1111$ neighboring nodes .

We use Pytorch [42] and the Pytorch Geometric [43] libraries to implement all our experiments. We initialize all models using Glorot initialization [44] and trained them to minimize the contrastive loss provided in Eq. (3) using the Adam optimizer [45] with an initial learning rate of 0.001. We tune the weight decay in $\{0.001, 0.01, 0.05, 0.1, 0.15\}$. We further tune the temperature τ in the loss function in $\{0.1, 0.5, 0.8, 1.0\}$ and the number of epochs in $\{20, 50, 100, 150, 200\}$.

To define the stochastic perturbation, we tune the probability of dropping an edge in $[0.05, 0.75]$ and the probability of dropping node features in $[0.2, 0.8]$. GraphCL is found to be robust to different choices of the perturbation parameters. However, we found that applying high perturbations to node features (i.e. randomly dropping 50% to 70% of input features) and small perturbations of the graph structure (i.e. randomly dropping 10% to 20% of edges) results in stronger representations.

4.2. Results

We present the results of evaluating node representations using downstream multiclass node classification tasks in Table 2. We report average results over 50 runs of training followed by a logistic regression. Specifically, we use the mean classification accuracy on the test nodes for transductive tasks and the micro-averaged F1 score on the (unseen) test nodes for the inductive setting.

Transductive

Method	Cora	Citeseer	Pubmed	ogbn-arxiv
Raw features	$47.9 \pm 0.4\%$	$49.3 \pm 0.2\%$	$69.1 \pm 0.3\%$	$55.50 \pm 0.23\%$
DeepWalk [12]	67.2%	43.2%	65.3%	$70.07 \pm 0.13\%$
DeepWalk + features	$70.7 \pm 0.6\%$	$51.4 \pm 0.5\%$	$74.3 \pm 0.9\%$	_
EP-B [38]	$78.1 \pm 1.5\%$	$71.0 \pm 1.4\%$	$79.6 \pm 2.1\%$	$68 \pm 0.00\%$
DGI [18]	$82.3 \pm 0.6\%$	$71.8 \pm 0.7\%$	$76.8 \pm 0.6\%$	$70.18 \pm 0.12\%$
GraphCL	$\mathbf{83.6 \pm 0.5}\%$	$\mathbf{72.5 \pm 0.7}\%$	$\mathbf{79.8 \pm 0.5}\%$	$\mathbf{70.18 \pm 0.17}\%$
GraphCL*	$\mathbf{84.6 \pm 0.4}\%$	$\mathbf{73.1 \pm 0.6}\%$	$\mathbf{80.1 \pm 0.5}\%$	$\mathbf{71.38 \pm 0.13}\%$
GCN(supervised)[2]	81.5%	70.3%	79.0%	$71.74 \pm 0.002\%$

Inductive

	Method	Reddit	PPI
Unsupervised	Raw features	0.585	0.422
	GraphSage-GCN [5]	0.908	0.465
	GraphSage-mean [5]	0.897	0.486
	GraphSage-LSTM [5]	0.907	0.482
	GraphSage-pool [5]	0.892	0.502
	DGI [18]	0.940 ± 0.001	0.638 ± 0.002
	GraphCL	$\mathbf{0.951 \pm 0.01}$	$\mathbf{0.659 \pm 0.006}$
	GraphCL*	$\mathbf{0.960 \pm 0.01}$	$\mathbf{0.841 \pm 0.004}$
Supervised	FastGCN [39]	0.937	–
	GaAN [40]	0.958 ± 0.001	0.969 ± 0.002

Table 2: Classification accuracy on transductive tasks and micro-averaged F1 score on inductive tasks

We report the results of EP-B provided in [38] and [46], and also the results provided in [18]. To insure a fair comparison with the other methods, we report

the results of the standard implementation of GraphCL which was described in the previous section. In particular we use a standard embedding size $P' = 512$. We refer to the results by **GraphCL** in table 2. We also report **GraphCL*** which refers to the results that were achieved using the best parameters including the best negative sampling strategy, choice of encoders and embedding size. For example, we notice a 1% gain on the classification accuracy of Cora when using a GCN encoder and sampling negative from the second order neighbors of the current example. Moreover, we notice a surprising 0.2 gain on the F1 score on PPI when increasing the embedding size to $P' = 2048$.

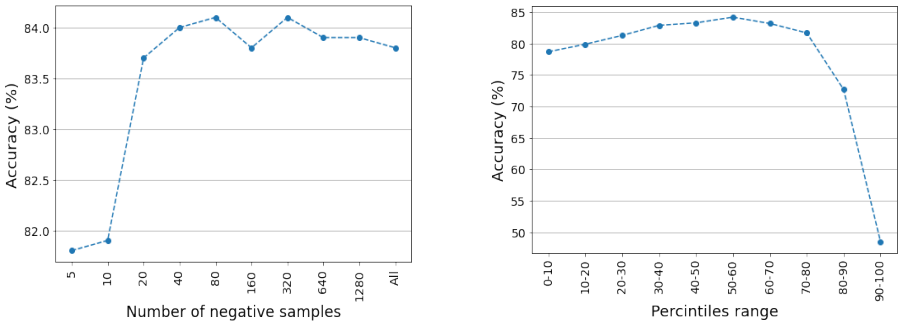
We see that the proposed GraphCL outperforms the previous state-of-the-art by achieving the best classification accuracy over the three transductive tasks and the best F1 score on inductive tasks. We note that, except for PPI dataset, GraphCL achieves competitive performance with strong supervised baselines without using label information. We assume that by maximizing agreement between representations that share the same information but have independent noise, GraphCL is able to learn representations that benefit from the richness of information in the graph which compensate for the information provided by the labels.

4.3. Ablation study

We report on a study to understand the effects of different parameters. All experiments have been conducted using Cora dataset.

4.3.1. Effect of the number of negatives

Figure 2a shows the effect of the number of negatives on the accuracy of the downstream classification task. We find that training a contrastive loss with a small number of negatives leads to poor representations. However, our experiments show that at a certain threshold increasing the number of negatives does not improve the quality of the representations. Beyond that threshold the variations of the classification accuracy seem to be due to the randomness of the training procedure only. Since using a large number of negatives slows down



(a) (b)

Figure 2: Classification accuracy on Cora dataset. **(a)** Effect of the number of negative examples. **(b)** Effect of the similarity between the current example and its corresponding negative samples.

the training and requires more computing power, our findings suggest that one has to properly choose the number of negatives to optimize for both the quality of the representations and the training efficiency.

4.3.2. Effect of feature similarity based negative sampling strategies

We next analyse the effect of hard negative samples on the quality of the learned representations. We first implement the feature similarity based negative sampling strategy described in section 3.2. We select negatives from a *ring* around the current example. This is done by varying the values of the percentiles ω_l and ω_k. Figure 2b show the accuracy of a linear classifier trained on the learned representation while varying the distance of the *ring* from the current example. We select negatives from a ring of diameter 10% (i.e. $\omega_l - \omega_k = 10\%$). The results confirm our intuition that hard negatives improve the quality of the representations. We notice that selecting only negatives that are too far from the current example leads to poor representations. In fact, if all negatives are easy to distinguish from the current example, there is no reason for the encoder to learn higher level features that can help to distinguish between the corresponding positive example and all the negatives. On the other hand, selecting negatives from nodes that are very similar to the current example worsens the

l-th order neighbors	Accuracy
1	31.4 ± 1.2 %
2	**84.6 ± 0.4%**
3	80.8 ± 0.6 %

Table 3: Classification accuracy on Cora dataset. Effect of the number of hops between the current example and its corresponding negative samples.

Encoder	Accuracy
MLP	66.1 %
Mean Pooling	83.6 %
GCN	**84.2 %**

Table 4: Classification accuracy on Cora dataset. Effect of the choice of the encoder

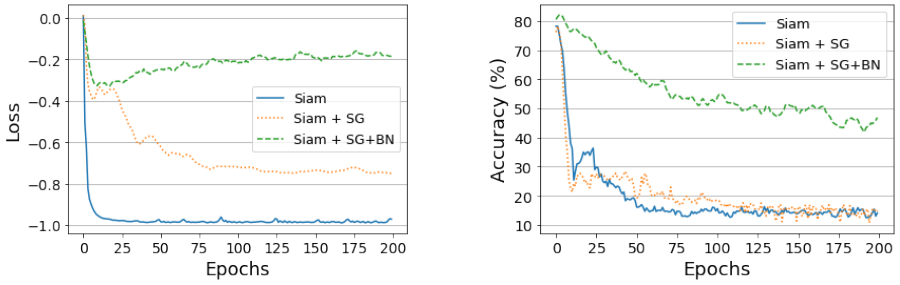
quality of the representations. This can be explained by the fact that negatives that are close to the current example are likely to belong to the same class and should rather be considered as positive examples. Training an encoder to push these examples away from the current example unsurprisingly leads to lower quality representations.

4.3.3. Effect of graph based negative sampling strategies

Graphs provide additional information about the examples. We aim at taking advantage of the graph structure to sample negative examples. Table 3 shows the average accuracy of 50 runs of training to learn nodes' embeddings on top of which we apply a linear classifier. We sample negatives from the l-th order neighbors of the current example. Similarly to the results of the feature similarity based negative sampling strategy, we find that negatives that are at the right distance from the current example improve the quality of the learned representations. More specifically, we achieve the best performance when sampling negatives from the second order neighbors.

4.3.4. Training without negative samples

In the previous section, we have discussed the effect of negative sampling strategies on the quality of the learned representations by using them to linearly classify nodes on a multi-class classification downstream task. Here, we would like to see whether it is possible to learn meaningful representations by maximizing the similarities between the representations obtained from two views



(a) (b)

Figure 3: A comparison of Siamese NN trained *with* vs *without* stop-gradient and batch normalization. **(a)** Training loss across epochs. **(b)** Accuracy of a linear classifier trained on top of the representation on Cora dataset

of the same graph without the use of any negative examples. To do so, we train a Siamese neural network to minimize the negative cosine similarity loss in Eq. (4). We implement the stop-gradient strategy described in section 2.4 and apply batch normalization on the hidden layer of the prediction MLP head (see section 2.4). Both stop-gradient and batch normalization have been reported to prevent the collapsing to a single representation when applied to Siamese neural networks for visual representations [27, 28]. Figures 3a and 3b respectively show the loss and accuracy of training a Siamese neural network without neither batch normalization nor stop-gradient referred to as *Siam*, with stop-gradient but without batch normalization referred to as *Siam+SG*, and with both stop-gradient and batch normalization referred to as *Siam+SG+BN*.

We observe that when training without stop-gradient and batch normalization, the loss function quickly converges to the minimum possible value -1. To verify that the cause is the collapsing to the single representation solution, we compute the standard deviation of all the representations which we found to be equal to zero for all features. We also notice that although adding stop-gradient and batch normalization prevent the collapsing to a single representation, the learned representations are still of low quality and perform much worse than the representations learned using negative samples.

5. Discussion

5.1. Connection to mutual information

The contrastive loss in Eq. (3) has been proposed as a lower bound estimator of the mutual information. A formal proof given by [47] shows that:

$$I(h_q, h_q^+) \geq \log(N) - \mathcal{L}, \tag{12}$$

where N is the number of negative samples Q_q^- and $I(h_q, h_q^+)$ is the mutual information between h_q and h_q^+:

$$I(h_q, h_q^+) = \mathop{\mathbb{E}}_{(h_q, h_q^+) \sim p_{h_q, h_q^+}(\cdot)} \log \left[\frac{p(h_q, h_q^+)}{p(h_q) p(h_q^+)} \right] \tag{13}$$

where $p(h_q, h_q^+)$ is the joint distribution of h_q and h_q^+, and $p(h_q)$ and $p(h_q^+)$ are the corresponding marginals.

Therefore, given any N, minimizing the loss function $\mathcal{L}$ also maximizes the lower bound on the mutual information $I(h_q, h_q^+)$. We note however that it has been shown that the bound in Eq. (12) can be not tight . Our experiments suggest that contrastive methods' success highly depends on other parameter designs, and so cannot be solely attributed to the properties of the mutual information. This confirms the remark done in[48] where the bound in Eq. (12) was seen not to be tight. More precisely, results in Table 4 emphasize the impact of the choice of the encoder on the performance of the contrastive loss. The ablation study that we conducted also highlights the effect of the negative sampling strategy and the importance of hard negative examples for learning powerful representations.

5.2. Understanding contrastive learning through alignment and uniformity on the hypersphere

To better understand the behavior of GraphCL, we analyze it through the perspective of uniformity and alignment that has been introduced in [49]. The main idea behind the contrastive loss is to attracting positive pairs together in

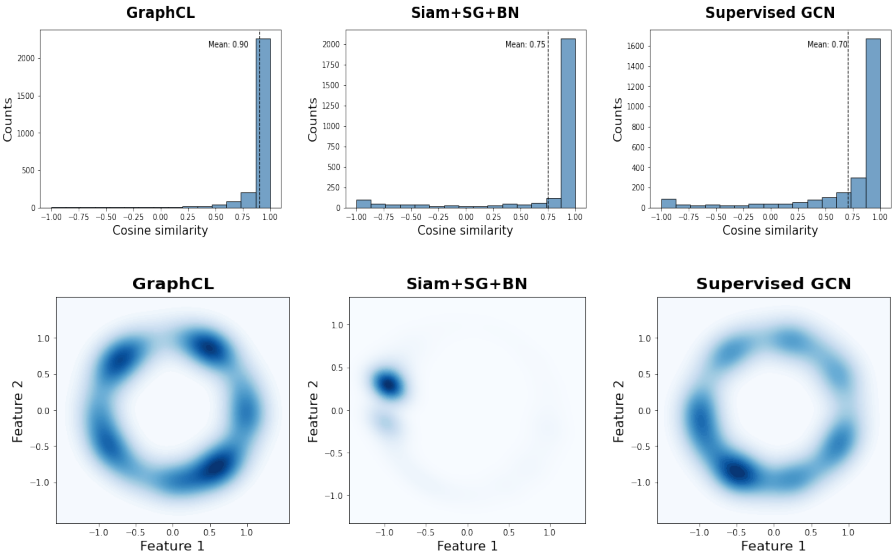


Figure 4: Representations of Cora dataset nodes on $\mathbb{R}^2$ using encoders trained with a contrastive loss (**Left plots**), a negative cosine similarity loss (**middle plots)** and a supervised cross entropy loss (**right plots**). Histograms of the cosine similarity between positive pairs (**Top)**. Feature distributions in $\mathbb{R}^2$ using Gaussian kernel density estimation (**Bottom**).

the representation space while pushing away the corresponding negative examples from the current sample. Eq. (3) actually encourages the learned representations to obey the following properties:

- **Alignment:** Representations of augmented views should be consistent and invariant to noise.
- **Uniformity:** The learned representations should match a prior distribution of high entropy (the uniform distribution over the hypersphere) to preserve as much information of the data as possible.

We visualize the learned representations of Cora dataset nodes in $\mathbb{R}^2$ (i.e $P' = 2$) to compare the behavior of the following methods:

- **GraphCL:** An encoder trained with the standard implementation of GraphCL as described above.

- **Siam+SG+BN:** A siamese neural network encoder trained with the negative cosine similarity loss using stop-gradient and batch normalization techniques.

- **Supervised GCN:** An encoder and a linear classifier trained jointly with a supervised cross entropy loss.

All encoders are 2-layers GCNs that map nodes to normalized feature vectors of dimension two. Figure 4 summarizes the resulting distributions. GraphCL embeddings clearly display both properties. Positive pairs are more aligned than those learned using the negative cosine similarity and supervised loss with an average cosine similarity of 0.9 for GraphCL and 0.75 and 0.7 respectively for the other methods. Representations of GraphCL are also evenly distributed on the hypersphere and exhibit the most uniform distribution.

It has been shown in [49] that both the alignment and uniformity properties are important in learning highly transferable features to downstream tasks. This contributes to the the success of GraphCL and may explain its ability to outperform strong supervised baselines on nodes classification, especially on the transductive learning setup.

It is also worth noticing that although adding stop-gradient and batch normalization techniques to the training procedure of the negative cosine similarity loss (i.e. Siam+SG+BN in Figure 4) prevent the collapsing to the single representation solution, the encoder fails to uniformly map the nodes' representations across the hypersphere. This explains its results on the classification downstream task (see Figure 3b).

5.3. Computational and model complexity

Last we discuss the computational and model complexity of GraphCL. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a graph and $N = |\mathcal{V}|$ the total number of nodes in the graph. Moreover, let L be the number of layers, M the minibatch size and R the number of neighbors being sampled for each node in the inductive setting. We

assume for simplicity that the dimension of the nodes' hidden features is constant and denote it as P'. The computational complexity and space complexity of GraphCL depend on the choice of the encoder. We use the same encoder for the two branches (i.e. each of the subgraphs). For the transductive learning setup, the computational and space complexity are linear with respect to the number of nodes and are respectively $\mathcal{O}(LNP'^2)$ and $\mathcal{O}(LNP' + KP'^2)$. For the inductive learning, we use a sub-sampling strategy to load the graphs into memory; the computational complexity is then $\mathcal{O}(R^L NP'^2)$ and the space complexity is $\mathcal{O}(MR^L P' + LP'^2)$. The computational complexity is linear with respect to the number of nodes. Both the number of layers L and the number of sampled neighbors R are fixed and user-specified. The space complexity is linear with respect to the minibatch size M. The sampling strategy sacrifices time efficiency to save memory which is necessary for very large graphs.

6. Conclusion

We introduced GraphCL, a general framework for self-supervised learning of nodes' representations. The key idea of our approach is to maximize agreement between two representations of the same node. The representations are generated by injecting random perturbations to the graph structure and nodes' intrinsic features. We have conducted a number of experiments on both transductive and inductive learning tasks. Experimental results show that GraphCL outperforms state-of-the-art unsupervised baselines on nodes' classification tasks and is competitive with supervised baselines. We further investigated different negative sampling strategies including training with a similarity based loss without contrasting with negative samples and propose a graph based negative sampling strategy. In the future, we will investigate the potential of our approach in learning graphs' representations that are robust to adversarial attacks on the graph data and explore the reasons of the low quality of nodes' representations when training a siamese neural network without negative samples.

References

[1] J. Atwood, D. Towsley, Diffusion-convolutional neural networks, in: Advances in Neural Information Processing Systems, 2016, pp. 1993–2001.

[2] T. N. Kipf, M. Welling, Semi-Supervised Classification with Graph Convolutional Networks, arXiv e-prints (2016) arXiv:1609.02907arXiv:1609.02907.

[3] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, P. Vandergheynst, Geometric deep learning: going beyond euclidean data, IEEE Signal Processing Magazine 34 (4) (2017) 18–42.

[4] K. Xu, W. Hu, J. Leskovec, S. Jegelka, How Powerful are Graph Neural Networks?, arXiv e-prints (2018) arXiv:1810.00826arXiv:1810.00826.

[5] W. Hamilton, Z. Ying, J. Leskovec, Inductive representation learning on large graphs, in: Advances in Neural Information Processing Systems, 2017, pp. 1024–1034.

[6] I. Chami, Z. Ying, C. Ré, J. Leskovec, Hyperbolic graph convolutional neural networks, in: Advances in Neural Information Processing Systems, 2019, pp. 4869–4880.

[7] S. Luan, M. Zhao, X.-W. Chang, D. Precup, Break the ceiling: Stronger multi-scale deep graph convolutional networks, in: Advances in Neural Information Processing Systems, 2019, pp. 10943–10953.

[8] T. N. Kipf, M. Welling, Variational Graph Auto-Encoders, arXiv e-prints (2016) arXiv:1611.07308arXiv:1611.07308.

[9] M. Zhang, Y. Chen, Link prediction based on graph neural networks, in: Advances in Neural Information Processing Systems, 2018, pp. 5165–5175.

[10] P. D. Hoff, A. E. Raftery, M. S. Handcock, Latent space approaches to social network analysis, Journal of the American Statistical Association 97 (460) (2002) 1090–1098.

[11] T. Mikolov, K. Chen, G. Corrado, J. Dean, Efficient Estimation of Word Representations in Vector Space, arXiv e-prints (2013) arXiv:1301.3781arXiv:1301.3781.

[12] B. Perozzi, R. Al-Rfou, S. Skiena, Deepwalk: Online learning of social representations, in: International Conference on Knowledge Discovery and Data Mining, 2014, pp. 701–710.

[13] A. Grover, J. Leskovec, node2vec: Scalable feature learning for networks, in: International Conference on Knowledge Discovery and Data Mining, 2016, pp. 855–864.

[14] Y. Dong, N. V. Chawla, A. Swami, metapath2vec: Scalable representation learning for heterogeneous networks, in: international conference on knowledge discovery and data mining, 2017, pp. 135–144.

[15] C. Zhang, D. Song, C. Huang, A. Swami, N. V. Chawla, Heterogeneous graph neural network, in: International Conference on Knowledge Discovery & Data Mining, 2019, pp. 793–803.

[16] C. Zhang, A. Swami, N. V. Chawla, Shne: Representation learning for semantic-associated heterogeneous networks, in: International Conference on Web Search and Data Mining, 2019, pp. 690–698.

[17] D. Wang, P. Cui, W. Zhu, Structural deep network embedding, in: International Conference on Knowledge Discovery and Data mining, 2016, pp. 1225–1234.

[18] P. Veličković, W. Fedus, W. L. Hamilton, P. Liò, Y. Bengio, R. Devon Hjelm, Deep Graph Infomax, arXiv e-prints (2018) arXiv:1809.10341arXiv:1809.10341.

[19] R. Devon Hjelm, A. Fedorov, S. Lavoie-Marchildon, K. Grewal, P. Bachman, A. Trischler, Y. Bengio, Learning deep representations by mutual information estimation and maximization, arXiv e-prints (2018) arXiv:1808.06670arXiv:1808.06670.

[20] T. Chen, S. Kornblith, M. Norouzi, G. Hinton, A Simple Framework for Contrastive Learning of Visual Representations, arXiv e-prints (2020) arXiv:2002.05709arXiv:2002.05709.

[21] K. He, H. Fan, Y. Wu, S. Xie, R. Girshick, Momentum contrast for unsupervised visual representation learning, in: IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 9729–9738.

[22] K. Hassani, A. Hosein Khasahmadi, Contrastive Multi-View Representation Learning on Graphs, arXiv e-prints (2020) arXiv:2006.05582arXiv:2006.05582.

[23] Y. You, T. Chen, Y. Sui, T. Chen, Z. Wang, Y. Shen, Graph contrastive learning with augmentations, in: Advances in Neural Information Processing Systems, Vol. 33, 2020.

[24] M. Gutmann, A. Hyvärinen, Noise-contrastive estimation: A new estimation principle for unnormalized statistical models, in: Conference on Artificial Intelligence and Statistics, 2010, pp. 297–304.

[25] Y. Kalantidis, M. B. Sariyildiz, N. Pion, P. Weinzaepfel, D. Larlus, Hard negative mixing for contrastive learning, in: Advances in Neural Information Processing Systems, Vol. 33, 2020.

[26] M. Wu, M. Mosse, C. Zhuang, D. Yamins, N. Goodman, Conditional Negative Sampling for Contrastive Learning of Visual Representations, arXiv e-prints (2020) arXiv:2010.02037arXiv:2010.02037.

[27] J.-B. Grill, F. Strub, F. Altché, C. Tallec, P. Richemond, E. Buchatskaya, C. Doersch, B. Avila Pires, Z. Guo, M. Gheshlaghi Azar, et al., Bootstrap your own latent-a new approach to self-supervised learning, in: Advances in Neural Information Processing Systems, Vol. 33, 2020.

[28] X. Chen, K. He, Exploring Simple Siamese Representation Learning, arXiv e-prints (2020) arXiv:2011.10566arXiv:2011.10566.