Supplementary Materials of: CoCoS: Enhancing Semi-Supervised Learning on Graphs with Unlabeled Data via Contrastive Context Sharing

Siyue Xie, Da Sun Handason Tam, Wing Cheong Lau

The Chinese University of Hong Kong
{xs019, tds019, wclau}@ie.cuhk.edu.hk

Discussions on Other Related Works

In our paper, we introduce some related works of semi-supervised GNN models and some GCL methods that motivate our work. In this part, we discuss more about these related works that may share some similar concepts of our proposed framework.

Comparison with Self-training Methods

Self-training is a technique for model performance enhancement under the semi-supervised setting. In general, the role (or label) of unlabeled samples will be estimated by the model during the training process. An unlabeled sample will be paired with its estimation and be added to the training set if the model is confident on the corresponding estimation. By progressively extending the training set, models can therefore learn from unlabeled data so as to enhance the overall learning capacity.

In recent years, self-training is also adapted by some recent works for graph representation learning, e.g., Multi-Stage Self-Supervised (M3S) Training algorithm (Sun, Lin, and Zhu 2020) and the Graph Agreement Models (GAM) (Stretcu et al. 2019). M3S extends the work of (Li, Han, and Wu 2018) by introducing a independent deep clustering algorithm for estimated label alignment. Specifically, a K-means algorithm is applied to cluster unlabeled samples and assign a pseudo label for samples in each cluster. During the training stage, an unlabeled sample will be added to the training set only when the model is confident enough on the estimation and the pseudo label matches the model's estimation. GAM introduces a set of regularizations when training the node classification model. Instead of following the assumption that connected nodes are supposed to have the same label (Bui, Ravi, and Ramavajjala 2018), GAM proposes an agreement model to estimate the label agreement between the two end nodes of an edge. The learned label agreement scores of each edge are utilized to penalize regularization terms to prevent overfitting. Jointing with the self-training strategy, GAM significantly improves the performance of some baseline models.

Although CoCoS and self-training both aim to incorporate unlabeled nodes into training, they are orthogonal enhancement techniques. Even though, CoCoS observes some advantages over M3S and GAM. M3S and GAM both require to introduce additional modules to assist the backbone model, where the former uses K-means clustering while the latter adopts an agreement model. These additionally introduced modules are independent to the backbone classification model, which can only be trained separately based on prior knowledge or predefined rules. Instead, CoCoS can adopt end-to-end training in the entire learning process. The backbone classification model and the discriminator can therefore be jointly integrated, which makes it possible to share knowledge learned from different part of the framework. On the other hands, adding unlabeled samples to the training set for the classification model can be a double-edge sword, as it may introduce noises to the learning process. On the contrary, CoCoS does not include unlabeled samples in the training set but correlates them with labeled data to regularize the model. Such an operation alleviates the influence of incorrect estimations, which makes it more robust to noises.

Comparison with Methods Using Data Augmentations

Data augmentation has already been demonstrated an effective way to improve the performance of a model (DeVries and Taylor 2017; Singh et al. 2018; Zhong et al. 2020) when available data are limited. Generally, samples are perturbated based on prior knowledge so as to generate more instances that are different from the one before perturbation but keep the same semantics or concept. Recent works on semi-supervised graph learning also adopt such a technique to improve model learning. For example, NodeAug (Wang et al. 2020b) introduces three different data augmentation strategies, including node attributes replacement, edge removal and edge adding, to enrich the training data. GRAND (Feng et al. 2020) generates multiple augmented graphs by random propagation with node dropping. Augmented instances can be paired with the original one, which derives a consistency loss to regularize the model.

In our proposed enhancement framework, the operation of context sharing can be regarded as a strategy of data augmentation. However, different from previous methods (Veličković et al. 2018; Zhu et al. 2020), context sharing shuffles node features based on ground-truth labels or es-

timated labels. In other words, it is totally data-driven instead of relying on human priors or handcrafted rules. Such an operation enforces the model to mine correlations of nodes from the same class, which encourages the GNN backbone to learn more robust and task-oriented representation for each node. In addition, compared with the aforementioned works that introduce the constraint to induce intra-class consistency, CoCoS goes further by applying contrastive learning to distinguish instances among across different class (inter-class distinction). The model enhanced by CoCoS can therefore generate more discriminative representation for each node, which improves the performance on down-stream tasks.

Comparison with C&S

Correct and Smooth (C&S) (Huang et al. 2020) is a recently proposed method that is used to enhance the performance on semi-supervised node classification tasks. Instead of using a GNN model to train in an end-to-end manner, C&S goes by two stages. In the first stage, a simple MLP model is trained with limited labeled data, which finally yields estimated labels for unlabeled data. In the second stage, an elaborate propagation framework is proposed to propagate the 'error correlation' and 'prediction correlation' within the graph, which aims to correct the estimated labels of the first stage.

In fact, we can find some similarities between C&S and CoCoS. Firstly, C&S and CoCoS are both enhancement techniques that can be regarded as the post-processing steps for existing models. The learning processes are both separated into two stages, and the latter stage relies on the estimation results of the former stage. Secondly, in the second stage, unlabeled nodes are involved in the 'correction' part, which makes use of all available data. In addition, both frameworks are computation-efficient, where CoCoS only add little computational overheads and the second stage of C&S is even parameter-free.

The main differences between C&S and CoCoS are the label correction operation and the node correlation strategies. In C&S, labels are updated by a propagation process, while CoCoS will progressively override the estimations using the predictions of the current training model. The elaborate propagation strategy used in C&S is manually designed and relies heavily on the prior knowledge. Instead, CoCoS can be trained in an end-to-end manner that is totally data-driven, which yields task-related representations for different applications. On the other hand, in C&S, nodes are correlated based on the propagation process, which is based on the proximity assumption. In contrast, nodes in CoCoS are correlated based on the context sharing strategy. Therefore, nodes can be correlated even if they are remote from each other in the graph. This helps CoCoS capture some general patterns within the same class of nodes, which yields more discriminative node representations for down-stream tasks.

Comparison with Self-Knowledge Distillation

Self-knowledge Distillation (self-KD) is a common trick that is used to train models for the Ogbn-arxiv dataset [1]. The main idea of self-KD is to train two models with the same architecture: one is trained as the teacher network that follows the semi-supervised settings, while the other acts as the student network that is trained by both cross-entropy loss of the node classification task and knowledge-distillation loss from the teacher network. By sharing the knowledge learned by the teacher network with the student network, the latter model can learn more powerful representations for classification tasks.

Similar to self-KD, CoCoS also benefits from knowledge sharing. Concretely, in the first step of CoCoS, a GNN model is pretrained using labeled data. This helps the model to obtain some initial knowledge about the graph. In the following steps of CoCoS, we continue to fine-tune the pretrained model by integrating context sharing and contrastive learning strategies, which reuses the knowledge of the pretrained model (e.g., estimated labels). With the initial knowledge, the model can therefore pay less attention to those easy samples and focus more on samples that are unseen or hard to classify, which yields discriminative representations for down-stream tasks.

However, differences are also obvious. In self-KD, both teacher and student networks are trained only with labeled data, while CoCoS incorporates all available data for training. In other words, CoCoS can better utilize the data and be able to capture more patterns from unlabeled data. On the other hands, CoCoS are trained using only one model, while self-KD should independently train two models with the same architecture. Therefore, the size of learnable parameters of CoCoS can be much smaller than that of self-KD, which is more computation-efficient.

Details of Experimental Settings

Dataset Descriptions

In our experiments, we use four different datasets on citation networks to evaluate our proposed framework. Statistics of each dataset can refer to Table 1.

Cora This dataset (Sen et al. 2008) contains 2708 nodes and 5429 edges. Each node is corresponding to a scientific publication from one of seven categories. An edge is created if there exist a reference/ citation relationship between two papers. A word dictionary is used to describe a paper, where each node can be represented by a 1433-dim word vector with 1/ 0 value in each element.

Citeseer This dataset (Sen et al. 2008) contains 3312 nodes and 4732 edges in total. The setting of Citeseer is similar to that of Cora but all papers (nodes) are collected from six categories. For each node, a 3703-dim word vector is used to represent the paper.

[1]The official leader board of the Ogbn-arxiv dataset: https://ogb.stanford.edu/docs/leader˙nodeprop/#ogbn-arxiv

Dataset	# Nodes	# Edges	# classes	# features	# Train/ Val/ Test Split	Label Rate (%)
Cora	2708	5429	7	1433	140/500/1000	5.17
Citeseer	3312	4732	6	3703	120/500/1000	3.62
Pubmed	19717	44338	3	500	60/500/1000	0.30
Ogbn-arxiv	169343	2501829	40	128	90941/29799/48603	53.70

Table 1: The statistics of different datasets.

Pubmed This dataset (Namata et al. 2012) is extracted from the PubMed database, where each collected paper is from one of three categories. It is made up of 19717 nodes and 44338 edges. Different from Cora and Citeseer, each paper is described by a 500-dim TF/IDF feature vector.

Ogbn-arxiv

This dataset (Hu et al. 2020) is a directed graph collected from MAG (Wang et al. 2020a). Each node is associated with an arXiv paper from one of the forty subject areas. A 128-dim feature vector generated from skip-gram model is used to represent a node. As each node is also labeled with the year it get published, the dataset is split into three parts for train/ validation/ test. Specifically, the papers published until year 2017 are labeled as training nodes, which with size 90941. The papers published in year 2018 and 2019 are used as validation and test set, which contain 29799 and 48603 nodes respectively.

Models for Comparison

We compare our proposed methods with many existing works, including a model doesn't incorporate graph structure information (MLP), graph learning models without GNN (node2vec and node2vec + raw features), state of the art GCL methods (DGI, BGRL and MVGRL) and typical semi-supervised GNN models (GCN, GAT, GraphSAGE, JK-Net and SGC). As time limited, we only implement MLP, node2vec and those typical semi-supervised GNN models. The implementation of GLC models will be left as our future works.

We briefly introduce each model in the following.

MLP Multi-Layer Perceptron (MLP) is a neural network that stacks multiple fully-connected layers. In our implementations, we use a 2-layer MLP for comparison. The hidden layer is configured with 16 neurons, which has the same hidden dimension as in GCN (Kipf and Welling 2016).

Node2vec Node2vec is an unsupervised graph learning model for node embeddings. Specifically, it samples a series of node sequence by a biased random walk, which draw analogy to 'word sequences' in the Natural Language Processing (NLP). Therefore, the word embedding technique, word2vec, can be applied to the graph structure data to learn an embedding for each node. In our experiments, we learn a 128-dim node2vec embedding for each node. An additional MLP, which follows the same settings as mentioned above, is trained to complete the node classification task, with the 128-dim node embedding as inputs. In Ogbn-arxiv dataset, we also compare with the model that with node2vec embedding and raw node features as inputs.

DGI Deep Graph Infomax (DGI) (Veličković et al. 2018) is an unsupervised node representation learning method. It relies on maximizing mutual information between the node (local) representations and graph (global) representations. Unlike Node2Vec, it does not rely on random walk objectives and thus does not focus on preserving homophily information in the original graph. It also follows the contrastive training paradigm where node and graph representations computed from the original graph are considered as positive pairs. Random node shuffling is applied to the original graph to generate a corrupted graph. The node representations computed from the corrupted graph and the graph representations computed from the original graph are considered as negative pairs.

BGRL BGRL (Thakoor et al. 2021) is also a self-supervised learning that is based on contrastive learning. Inspired by BYOL (Grill et al. 2020), BGRL eliminates the need of negative samples by bootstrapping the output of a delayed version of its encoder, without the need of negative examples. For positive examples, BGRL uses node features masking and random edges removal to generate augmented views of a graph. The embeddings of the same node in the two augmented graphs are considered as positive examples.

MVGRL MVGRL (Hassani and Khasahmadi 2020) is also a contrastive-learning based model. It maximizes the mutual information between representations encoded from different structural views of graphs. MVGRL uses graph diffusion and random node sampling to generate augmented graphs. Node representations and graph representations computed from augmented graphs are considered as positive examples. Similar to DGI, MVGRL also relies on negative samples. Towards this end, MVGRL generates corrupted graphs by applying random node shuffling. Node representations computed from the corrupted graph and the graph representations computed from the original or augmented graph are considered as negative pairs.

GCN Graph Convolutional Network (GCN) (Kipf and Welling 2016) is a semi-supervised GNN model for graph learning. GCN adopts a simplified spectral graph convolutional operation in each layer, which essentially aggregates the neighbors' information of a target node with the normalized adjacency matrix. By stacking multiple GCN layers, graph-related patterns within the multi-hop neighborhood can be embedded into the node representation, which can be supervised by some given node labels. In our implementations, we follow (Kipf and Welling 2016) to build a two-layer GCN model. The hidden dimension of each layer is 16 for Cora, Citeseer and Pubmed datasets. In Ogbn-arxiv, we

directly report the performance of GCN in the leaderboard, which is configured with three GCN layers with 256 hidden dimensions.

GraphSAGE GraphSAGE (Hamilton, Ying, and Leskovec 2017) adopts a spatial-based aggregating strategy for each layer. For a target node, information is directly aggregated from its neighbors following different kind of aggregators. Neighbor sampling strategy can sometimes be applied in the aggregation process when there are too many many neighbors for a node, which makes it capable to deal with large-scale graphs. In our experiments, we implement two kinds of GraphSAGE by using different aggregators. The one uses 'gcn' aggregator is denoted as SAGE(g), while the other using 'mean' aggregator is denoted by SAGE(m). In our implementation, to align with GCN, we stack two GNN layers for GraphSAGE and set the hidden dimension to be 16 for experiments on Cora, Citeseer and Pubmed. In Ogbn-arxiv, we report the performance of GraphSAGE in the leaderboard, which is a three-layer model with 256 hidden dimensions.

GAT Graph Attention Network (GAT) (Veličković et al. 2017) also adopts a spatial-based aggregation strategy to learn structural information. However, different from GraphSAGE, it additionally introduce an attention mechanism into the aggregation process. Attention heads are applied to assign attention weights for each neighbors of a target node in the aggregation, which distinguishes the importance of different nodes. In our implementations, we follow the work of (Veličković et al. 2017), which builds a 2-layer GAT model for Cora, Citeseer and Pubmed. It is configured with 8 attention head, each of which is with a 8-dim embedding. In Ogbn-arxiv, we follow the common configurations of GAT-variants in the leaderboard. Concretely, a three-layer model with 3 attention heads , each of which is with 256 dimensions.

JK-Net Jumping-Knowledge Network (JK-Net) (Xu et al. 2018) is a general framework that can be implemented based on different GNN backbones. In JK-Net, a bypass-link is added between each GNN layer and the final GNN layer. Node representations generated in each layer can therefore be concatenated together, which yields a comprehensive embedding that captures information of different hops of information. In our experiments, we implement JK-Net with GCN as the backbone. We also add a one-layer MLP as classifier after the final GNN layer. For Cora, Citeseer and Pubmed, a two-layer JK-Net with 16 hidden dimensions is constructed. For Ogbn-arxiv, we report the performance listed in the official leaderboard.

SGC Simplified Graph Convolution (SGC) (Wu et al. 2019) simplifies the general GNN framework by removing all linear transformations and non-linear activations in each layer. This induces a light-weight model and significantly speed up the training and inferences. In our experiments, we build a two-layer SGC for Cora, Citeseer and Pubmed and a three-layer model for Ogbn-arxiv.

Implementation Details

In this section, we specify the details of our model implementations for experiments.

Dataset Splits For Cora, Citeseer and Pubmed datasets, we follow the settings of (Kipf and Welling 2016; Veličković et al. 2017; Wu et al. 2019) to construct the train/ validation/ test splits. Specifically, for each class, 20 nodes are picked up to form the training set. Another 500 nodes and 1000 nodes are collected for validation and test set respectively. For the Ogbn-arxiv dataset, we follow the official train-validation-test split to evaluate our model. Detailed statistics refers to Table 1.

Network Architecture CoCoS can be divided into two parts, i.e., the GNN backbone for embedding learning and the discriminator for contrastive learning. In our experiments, we applied CoCoS to all aforementioned GNN backbones, where configurations are all the same as above. As for the discriminator, we use a two-layer MLP In Cora, Citeseer and Pubmed datasets, the hidden layer dimension is 64 while in Ogbn-arxiv we set it as 256.

Hyperparameter Settings In Cora, Citeseer and Pubmed, all backbones are trained for 300 epochs, with learning rate 0.01 and weight-decay 5×10^{-4}. In Ogbn-arxiv, we train all backbones for 1000 epochs, with the learning rate 0.002 and weight-decay 5×10^{-4}. One exception is SGC, where we follow the work of (Wu et al. 2019), which sets the learning rate and weight-decay to be 0.2 and 5×10^{-5} respectively.

As for CoCoS, we follow the grid-search strategy to find a better combination of hyperparameters. Specifically, in Cora, Citeseer and Pubmed, the learning rate is set to be 0.05, 0.02, 0.01, 0.005, each of which is associated with a 5×10^{-4} weight-decay. The adjustable coefficient α in Eq. 11 is set to be 0.2, 0.4, 0.6 and 1.0. For each experiment, we fine-tune the pretrained GNN models for 200 epochs. The estimated labels will be overridden by the current training model for every 10 epochs. In the end, we find that the model yields a relatively better performance with 0.05 learning rate and $\alpha = 0.6$ weight-decay. Therefore, the performances of all variants of CoCoS reported in this paper are all based on such a group of hyperparameters. In Ogbn-arxiv, we do not conduct hyperparameter searching. Instead, we set the learning rate to be 0.002 and let $\alpha = 0.6$. The model is then trained for 300 epochs. The estimated labels will be overridden by the current training model's prediction for every 10 epochs.

Training Details CoCoS is taken as an enhancement technique for current GNN-based models. Therefore, we will first trained a GNN model on the datasets using the limited node labels. CoCoS will then be applied to continue to fine-tune such a pretrained model. Note that in Ogbn-arxiv, there are totally 40 different nodes in the graph. The model may be unstable in the training process if we shuffle all classes of nodes in one go. To solve this problem, we split each epoch into ten steps. For each step, we only randomly shuffle four classes of nodes. This helps reduce some randomness and makes the training more stable. In our experiments, we run

Model	Cora	Citeseer	Pubmed
GCN	82.12	71.07	78.79
GCN-CS(O)	87.45	78.22	83.09
GAT	81.95	70.68	78.29
GAT-CS(O)	87.57	79.41	82.35
SAGE(g)	82.06	70.66	78.29
SAGE(g)-CS(O)	88.09	78.94	83.92
SAGE(m)	81.55	69.76	78.49
SAGE(m)-CS(O)	89.59	82.35	85.30
JK-Net	80.69	68.03	78.65
JK-Net-CS(O)	88.89	80.04	84.71
SGC	80.55	72.00	78.86
SGC-CS(O)	87.03	76.88	87.29

Table 2: Node classification performances with oracles (%).

each model for ten times and record the test accuracy corresponding to the best validation accuracy that the model has achieved. We finally report the mean test accuracy of all ten runs. For Ogbn-arxiv, we also report the validation accuracy and standard deviations following the requirements of the OGB leaderboard.

All our models are implemented based on the PyTorch framework with the DGL library. All datasets are loaded through DGL. Our experiments are conduct on a single Linux machine, with an AMD Ryzen Threadripper 3970X CPU @ 3.7GHz, 256 GB RAM, and a NVIDIA RTX TITAN GPU with 24 GB RAM.

Supplementary Experimental Results

Context Sharing Using Oracles with Different GNN Backbones

To verify that context sharing can really help the model learn better or more discriminative representations, we use the oracles, i.e., ground-truth labels, of unlabeled data to guide the intra-class features shuffling. Also, to validate that such a strategy is not only effective to a specific model, but also workable for many other GNNs, we apply it to all of the aforementioned semi-supervised GNN baselines.

The experimental results are shown in Table 2. As we can observe from the results, all baseline models are significantly improved if we can use oracles for intra-class nodes shuffling. Such a result also reveals that context sharing can also benefit those typical GNN models that are most popular. This motivates us to find a substitution of the oracles even if it may introduce some noises. Minor noises are expected not to affect the model training too much. This may owe to the nature of GNN, which learns knowledge by aggregate information from neighbors. Therefore, we can expect that the prediction result of a node can be more accurate if most of its neighbors' label are correctly estimated. Such an intuition inspired us to use the prediction results of a pretrained model as the initial estimations for unlabeled nodes.

Node Classifications on Ogbn-arxiv Dataset

We additionally report the validation accuracy and standard deviation of CoCoS on the Ogbn-arxiv dataset, which aligns with the official leaderboard. The results are shown in Table 3. CoCoS outperforms all other compared models in both test and validation accuracy, which demonstrates its effectiveness.

Model	Test Accuracy	Validation Accuracy
MLP	55.50 ± 0.23	57.65 ± 0.12
n2v+feat	70.07 ± 0.13	71.29 ± 0.13
BGRL	71.51 ± 0.11	72.61 ± 0.15
GCN	71.74 ± 0.29	73.00 ± 0.17
GAT	72.24 ± 0.16	73.30 ± 0.03
SAGE(g)	71.32 ± 0.17	72.51 ± 0.04
SAGE(m)	71.49 ± 0.27	72.27 ± 0.16
JK-Net	72.19 ± 0.21	73.35 ± 0.07
SGC	69.99 ± 0.18	70.86 ± 0.08
CoCoS	$\mathbf{72.95} \pm 0.19$	$\mathbf{74.35} \pm 0.06$

Table 3: The classification accuracies on Ogbn-arxiv (%). Note that the performance of all other compared models is obtained from the Ogbn-arxiv official leaderboard or from their published papers, except GAT and SAGE(g), which are implemented by ourselves.

Model	Cora	Citeseer	Pubmed
GCN-CS	82.83(+0.71)	72.30(+1.23)	78.93(+0.14)
GCN-Both	83.20(+1.08)	73.13(+2.06)	79.68(+0.89)
GAT-CS	81.77(−0.18)	72.36(+1.68)	77.36(−0.93)
GAT-Both	82.64(+0.69)	72.42(+1.74)	78.87(+0.58)
SAGE(g)-CS	82.73(+0.67)	71.92(+1.26)	79.34(+0.69)
SAGE(g)-Both	83.35(+1.29)	72.70(+2.04)	79.51(+0.86)
SAGE(m)-CS	82.60(+1.05)	72.61(+2.85)	78.21(−0.28)
SAGE(m)-Both	83.05(+1.50)	72.96(+3.20)	79.93(+1.44)
JK-Net-CS	82.54(+1.85)	71.25(+3.22)	79.55(+0.09)
JK-Net-Both	83.17(+2.48)	71.85(+3.82)	79.95(+1.30)
SGC-CS	80.93(+0.38)	72.56(+0.56)	78.95(+0.09)
SGC-Both	80.93(+0.38)	72.56(+0.56)	78.95(+0.09)

Table 4: The classification accuracies when using different context sharing strategies for different GNN baselines (%). Values in the parentheses are the improvements with respect to the corresponding baseline.

Context Sharing on Different GNN Backbones

In this ablation experiment, we investigate the effect of different strategies of context sharing configurations (Eq. 2 and Eq. 10 of the main paper) on different GNN backbone models. The model trained by Eq. 2 is denoted by a '-CS' suffix while the model trained by Eq. 10 are marked by a '-Both' suffix.

Experimental results are shown in Table 4. We can observe that in most cases, both of these two strategies can improve the performance of the corresponding baseline model by different extent. Exceptions only occur when using Eq. 2 to train GAT on Cora and Pubmed and SAGE(m) on Pubmed, where classification accuracies of these models drop slightly. This may due to some chaos raised by the randomness of intra-class feature shuffling. In contrast, when introducing the supervised part (Eq. 10) to train models, the performance of classification accuracy can be better.

Model	Cora	Citeseer	Pubmed
CoCoS-c-F	**84.49**(+2.37)	**73.62**(+2.55)	80.93(+2.14)
CoCoS-c-T	84.07(+1.95)	73.39(+2.32)	81.24(+2.45)
CoCoS-c-M	84.18(+2.06)	73.38(+2.31)	81.03(+2.24)
CoCoS-c-S	84.48(+2.36)	73.57(+2.50)	80.39(+1.60)
CoCoS-c-FS	84.15(+2.03)	73.57(+2.50)	80.92(+2.13)
CoCoS-c-MS	84.02(+1.90)	73.56(+2.49)	80.81(+2.02)
CoCoS-a-F	83.43(+1.49)	72.78(+2.10)	80.16(+1.87)
CoCoS-a-T	83.07(+1.12)	72.68(+2.00)	80.45(+2.16)
CoCoS-a-M	83.44(+1.49)	72.27(+1.59)	80.37(+2.08)
CoCoS-a-S	83.69(+1.74)	72.69(+2.01)	78.95(+0.66)
CoCoS-a-FS	83.81(+1.86)	72.98(+2.30)	79.85(+1.56)
CoCoS-a-MS	83.21(+1.26)	73.03(+2.35)	79.59(+1.30)
CoCoS-sg-F	83.89(+1.83)	73.21(+2.55)	81.67(+3.02)
CoCoS-sg-T	83.50(+1.44)	73.28(+2.62)	81.57(+2.92)
CoCoS-sg-M	83.49(+1.43)	73.20(+2.54)	81.25(+2.60)
CoCoS-sg-S	84.33(+2.27)	72.87(+2.21)	80.62(+1.97)
CoCoS-sg-FS	83.77(+1.71)	73.10(+2.44)	81.59(+2.94)
CoCoS-sg-MS	84.02(+1.96)	73.23(+2.57)	80.95(+2.30)
CoCoS-sm-F	83.71(+2.16)	72.67(+2.91)	81.38(+2.89)
CoCoS-sm-T	83.44(+1.89)	73.30(+3.54)	**82.06**(+3.57)
CoCoS-sm-M	83.26(+1.71)	73.39(+3.63)	81.49(+3.00)
CoCoS-sm-S	83.55(+2.00)	73.50(+3.74)	80.22(+1.73)
CoCoS-sm-FS	83.58(+2.03)	73.44(+3.68)	80.68(+2.19)
CoCoS-sm-MS	83.72(+2.18)	73.05(+3.29)	81.20(+2.71)
CoCoS-j-F	83.71(+3.02)	71.57(+3.54)	80.51(+1.86)
CoCoS-j-T	83.46(+2.77)	71.67(+3.64)	80.74(+2.09)
CoCoS-j-M	83.59(+2.90)	71.46(+3.43)	80.50(+1.85)
CoCoS-j-S	84.03(+3.34)	71.74(+3.71)	80.34(+1.69)
CoCoS-j-FS	83.59(+2.90)	71.47(+3.44)	80.68(+2.03)
CoCoS-j-MS	83.60(+2.91)	71.67(+3.64)	80.64(+1.99)
CoCoS-sc-F	81.65(+1.10)	72.86(+0.86)	79.59(+0.73)
CoCoS-sc-T	81.19(+0.64)	72.09(+0.09)	79.33(+0.47)
CoCoS-sc-M	81.31(+0.76)	72.32(+0.32)	79.33(+0.47)
CoCoS-sc-S	81.18(+0.63)	71.78(−0.22)	79.30(+0.44)
CoCoS-sc-FS	81.33(+0.78)	72.39(+0.39)	79.35(+0.49)
CoCoS-sc-MS	81.13(+0.58)	72.33(+0.33)	79.35(+0.49)

Table 5: The classification accuracies when training with different contrastive pairs using different GNN backbone models (%). Values in the parentheses are improvements with respect to the corresponding baseline.

This reveals that Eq. 10 can help correct some inconsistency raised by the random shuffling operations, which stabilizes the training and results in yielding a more powerful and robust representation for each node.

Effect of Different Contrastive Augmentations with Different GNNs

In this ablation study, we investigate the effect of different combinations of contrastive pairs. Experiments are conducted on all aforementioned GNNs. Following the notations of the main content of our paper, we use lowercase letters to indicate different kinds of GNN backbones, and capital letters to denote different kinds of contrastive combinations.

We report the experimental results in Table 5. Obviously, CoCoS consistently enhances different types of GNN models when trained using different combinations of contrastive pairs. All classification accuracies are significantly improved except the variant on SGC. This may due to the fact that SGC only has very limited model size, which essentially prevents it to learn more advanced patterns from graphs. In addition, from Table 5, we can observe that the variants of CoCoS-c perform better in Cora and Citeseer datasets, while variants of CoCoS-sg and CoCoS-sm yield a relatively better performance in Pubmed dataset. CoCoS-sm-T can even reach the best with 82.06% accuracy in the Pubmed dataset, which is much better than all other GCL models and semi-supervised GNN baselines. This also reveals that the learning capacity of some existing GNN models are under-exploited. Instead, CoCoS-enhanced model can yield a better performance, which demonstrates its effectiveness on model learning improvement.

Model	Cora	Citeseer	Pubmed
DGI	82.30	71.80	76.80
MVGRL	**86.80**	73.30	80.10
CoCoS	83.75	**73.44**	**81.58**

Table 6: Accuracies of nodes classification in different datasets (%). Experimental settings of CoCoS are aligned with GCL methods.

Performance of CoCoS When Following the Settings of GCL Methods

In our experiments, we do not re-implement GCL methods but compare with the performances reported in their published paper. However, one should be noted that the network settings as well as some training settings of these GCL models are different from CoCoS. For example, the hidden dimension of GCL models such as DGI and MVGRL is set as 512, while in CoCoS reported above, this value is only 16. To make a more fair comparison, we conduct a series of additional experiments, which aligns the setting with the aforementioned GCL methods. Specifically, we use GCN as the backbone model for CoCoS, with a 512-d hidden layer. The dimension of the hidden layer in the discriminator is set as 128. We run 300 epochs to pretrain the GCN backbone, with a learning rate of 0.001, while fine-tune the pretrained model with a learning rate of 0.05 for 200 epochs. We run the model for 50 times and report the mean of test accuracy.

The results are shown in Table 6, where CoCoS is the runner-up in Cora dataset and dominates other GCL models in both Citeseer and Pubmed datasets. Such an experiment demonstrates the effectiveness of CoCoS. However, we observe from the experiments' log that the backbone model, GCN, may fall into overfitting during the pretraining stage in the Cora dataset. This indicates that the knowledge learned by the pretrained model may not be informative enough, which may result in a sub-optimal performance when continuing to fine-tune on the the backbone model. In contrast, GCL methods will learn from the entire graph instead of those labeled nodes (and their neighbors) from the very beginning. Such an advance reduces the risk of falling into overfitting even if the model are configured with a large number of learnable parameters. This may explain the phenomenon that MVGRL performs better than CoCoS in Cora. Even though, CoCoS can still beat other GCL methods in Cora and dominates the other two datasets with larger size, which demonstrates its effectiveness.

Additional Experiments

To further explore the strength and weakness of CoCoS, we conduct some additional experiments on four other datasets. The statistics of each dataset are illustrated in Table 7.

Datasets

Amazon-Computer and Amazon-Photo These two datasets (Shchur et al. 2018) are co-purchase network that is constructed from Amazon. Each node stands for a good and an edge indicates the two corresponding goods are frequently purchased together. A node is represented by a bag-of-word feature vector, which is derived from the product reviews. The goal is to classify each node into one of the ten good categories. We abbreviate Amazon-Computer as AMZ-C and Amazon-Photo as AMZ-P in the followings.

Coauthor-CS and Coauthor-Physics These two datasets (Shchur et al. 2018) are coauthor networks of academic papers from two different domains. Each node is corresponding to a unique author, and an edge between two nodes indicates that the two authors co-authored a paper. Node features are sparse bag-of-word vectors based on the paper keywords of the corresponding author. The task is to map each author to their respective study field (i.e., node classifications). We abbreviate Coauthor-CS as Co-CS and Coauthor-Physics as Co-Phy in the followings.

Implementation Details

Data Splits and Evaluation Protocols We apply the same data split strategy on all these four datasets for our additional experimental evaluations. Specifically, we randomly sample 10% of nodes in each dataset for training and 10% for evaluation. The remaining 80% of nodes are left for testing. For each run of experiments, we record the test accuracy of the model that with the best evaluation accuracy. We repeat the experiments for twenty runs and report the average number over all twenty rounds.

Hyperparameter Settings In the experiments, we apply the same set of hyperparameters for each model on these four datasets. For baseline models, i.e., MLP, node2vec, GraphSAGE, GCN and JKNet, we set the total training epochs to 150, dropout rate to 0.6, weight decay to 5×-4. Each model consists of 2 layers, where the dimension of the hidden layer is 64. For CoCoS-enhanced models, we set the total training epochs to 400, $\eta = 20$, $\alpha = 0.6$ and use the combination of 'M' and 'S' (the meaning of them refers to the content of our main paper) for positive contrastive pairs. The discriminator consists of 2 layers, where the hidden dimension is 64. Other settings are kept the same as that of the corresponding backbone.

Classification Results and Observations

The experimental results on the four additional datasets are shown in Table 8. From the results, we can observe that CoCoS shows different effects on the backbone models on different datasets. In the two coauthor datasets, models enhanced by CoCoS are slightly improved, while the corresponding accuracies on the two co-purchase datasets drop slightly.

The benefits brought by CoCoS in these four datasets are not as significant as that of the four standard citation networks we aforementioned in the main paper, and some of them even get worse. By analyzing the characteristics of each dataset, we speculate some possible reasons that may lead to such phenomenon.

For Amazon co-purchase datasets, graph topology is supposed to be more informative when inferring the role of each node, while node features contribute small. Evidences are that there is a huge performance gap between MLP and node2vec, but the performance gap between node2vec and other GNN-based baselines are much smaller. (Note that MLP only takes node features into account, while node2vec generates node embeddings only based on the graph topology.) In other words, features of nodes within the same class may not be strong-correlated. However, a key operation to enhance a model is to apply context sharing, which supposes intra-class node features are somewhat mutually interchangeable. Therefore, when applying CoCoS to models on these two datasets, the context sharing strategy may not work but even introduce more noises. This results in little performance improvement or even worsen the model.

On the other hand, the situation of the two coauthor datasets can be contrary. We can observe from Table 8 that the performance gap between node2vec and MLP are much larger than that between MLP and other GNN-based models. When only using node features for prediction (i.e., the MLP model), it is good enough to obtain a relatively high accuracy number (over 90% for both datasets). In other words, node features can be much more informative in these two coauthor datasets, while graph topological structure counts much less. GNN-based methods are therefore not much more effective than MLP-based models. However, CoCoS is mainly designed to enhance learning on graph data. When GNN-based models cannot benefit from capturing the graph topological information, the enhancement techniques of CoCoS may not come into play. This explains the insignificant improvement of CoCoS on these two datasets.

Discussions

With the above experiments and observations, we can have a more clear view on CoCoS. On one hand, CoCoS can be effective to enhance the performance of GNN baselines for graph learning, as shown in the experiments on those citation datasets (i.e., Cora, Citeseer, Pubmed and Ogbn-arxiv). On the other hand, it will not always work for all if some of the conditions or assumptions are not satisfied, as shown in the experimental results on the Amazon co-purchase datasets and the coauthor datasets. Taking the above into account, we can conclude the strength of CoCoS and also some inapplicable scenarios as follows:

1. CoCoS is able to mine the intra-class correlations of node features through the context sharing scheme. By applying to GNN-based models, correlation knowledge can be jointly integrated with the graph topological information, which enhances the learning capacity of the model.
2. One key strength of CoCoS is to better integrate both node features and graph topological information for graph

Dataset	# Nodes	# Edges	# classes	# features	# Train/ Val/ Test Split (%)	Label Rate (%)
Amazon-Computer	13752	245861	10	767	10/10/80	10
Amazon-Photo	7650	119081	10	745	10/10/80	10
Coauthor-CS	18333	81894	15	6805	10/10/80	10
Coauthor-Physics	34493	247962	5	8415	10/10/80	10

Table 7: The statistics of datasets for additional experiments.

Model	Co-CS	Co-Phy	AMZ-C	AMZ-P
MLP	91.58	95.17	72.14	82.01
n2v	84.81	91.92	84.90	89.34
GCN	92.39	95.68	89.46	92.77
SAGE(g)	92.22	95.48	89.16	92.71
SAGE(m)	94.08	96.17	89.77	93.89
JK-Net	92.27	95.62	89.42	92.54
CoCoS(c)	93.00	95.88	88.93	92.84
CoCoS(sg)	92.59	95.66	88.61	92.72
CoCoS(sm)	94.40	96.29	89.01	93.80
CoCoS(j)	92.92	95.95	89.49	92.92

Table 8: Node classification accuracies on additional datasets (%).

learning. However, when one (or both) of these two factors is (are) unavailable or less informative, the performance gain of applying CoCoS may not be significant.

References

Bui, T. D.; Ravi, S.; and Ramavajjala, V. 2018. Neural graph learning: Training neural networks using graphs. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, 64–71.

DeVries, T.; and Taylor, G. W. 2017. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*.

Feng, W.; Zhang, J.; Dong, Y.; Han, Y.; Luan, H.; Xu, Q.; Yang, Q.; Kharlamov, E.; and Tang, J. 2020. Graph Random Neural Network for Semi-Supervised Learning on Graphs. In *NeurIPS'20*.

Grill, J.-B.; Strub, F.; Altché, F.; Tallec, C.; Richemond, P. H.; Buchatskaya, E.; Doersch, C.; Pires, B. A.; Guo, Z. D.; Azar, M. G.; et al. 2020. Bootstrap your own latent: A new approach to self-supervised learning. *arXiv preprint arXiv:2006.07733*.

Hamilton, W. L.; Ying, R.; and Leskovec, J. 2017. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 1025–1035.

Hassani, K.; and Khasahmadi, A. H. 2020. Contrastive multi-view representation learning on graphs. In *International Conference on Machine Learning*, 4116–4126. PMLR.

Hu, W.; Fey, M.; Zitnik, M.; Dong, Y.; Ren, H.; Liu, B.; Catasta, M.; and Leskovec, J. 2020. Open graph benchmark: Datasets for machine learning on graphs. *arXiv preprint arXiv:2005.00687*.

Huang, Q.; He, H.; Singh, A.; Lim, S.-N.; and Benson, A. R. 2020. Combining label propagation and simple models out-performs graph neural networks. *arXiv preprint arXiv:2010.13993*.

Kipf, T. N.; and Welling, M. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

Li, Q.; Han, Z.; and Wu, X.-M. 2018. Deeper insights into graph convolutional networks for semi-supervised learning. In *Thirty-Second AAAI conference on artificial intelligence*.

Namata, G.; London, B.; Getoor, L.; Huang, B.; and EDU, U. 2012. Query-driven active surveying for collective classification. In *10th International Workshop on Mining and Learning with Graphs*, volume 8, 1.

Sen, P.; Namata, G.; Bilgic, M.; Getoor, L.; Galligher, B.; and Eliassi-Rad, T. 2008. Collective classification in network data. *AI magazine*, 29(3): 93–93.

Shchur, O.; Mumme, M.; Bojchevski, A.; and Günnemann, S. 2018. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*.

Singh, K. K.; Yu, H.; Sarmasi, A.; Pradeep, G.; and Lee, Y. J. 2018. Hide-and-seek: A data augmentation technique for weakly-supervised localization and beyond. *arXiv preprint arXiv:1811.02545*.

Stretcu, O.; Viswanathan, K.; Movshovitz-Attias, D.; Platanios, E.; Ravi, S.; and Tomkins, A. 2019. Graph Agreement Models for Semi-Supervised Learning. In *Advances in Neural Information Processing Systems*, volume 32.

Sun, K.; Lin, Z.; and Zhu, Z. 2020. Multi-stage self-supervised learning for graph convolutional networks on graphs with few labeled nodes. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, 5892–5899.

Thakoor, S.; Tallec, C.; Azar, M. G.; Munos, R.; Veličković, P.; and Valko, M. 2021. Bootstrapped representation learning on graphs. *arXiv preprint arXiv:2102.06514*.

Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Lio, P.; and Bengio, Y. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903*.

Veličković, P.; Fedus, W.; Hamilton, W. L.; Liò, P.; Bengio, Y.; and Hjelm, R. D. 2018. Deep graph infomax. *arXiv preprint arXiv:1809.10341*.

Wang, K.; Shen, Z.; Huang, C.; Wu, C.-H.; Dong, Y.; and Kanakia, A. 2020a. Microsoft academic graph: When experts are not enough. *Quantitative Science Studies*, 1(1): 396–413.

Wang, Y.; Wang, W.; Liang, Y.; Cai, Y.; Liu, J.; and Hooi, B. 2020b. Nodeaug: Semi-supervised node classification with data augmentation. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 207–217.

Wu, F.; Souza, A.; Zhang, T.; Fifty, C.; Yu, T.; and Weinberger, K. 2019. Simplifying graph convolutional networks. In *International conference on machine learning*, 6861–6871. PMLR.

Xu, K.; Li, C.; Tian, Y.; Sonobe, T.; Kawarabayashi, K.-i.; and Jegelka, S. 2018. Representation learning on graphs with jumping knowledge networks. In *International Conference on Machine Learning*, 5453–5462. PMLR.

Zhong, Z.; Zheng, L.; Kang, G.; Li, S.; and Yang, Y. 2020. Random erasing data augmentation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, 13001–13008.

Zhu, Y.; Xu, Y.; Yu, F.; Liu, Q.; Wu, S.; and Wang, L. 2020. Deep graph contrastive representation learning. *arXiv preprint arXiv:2006.04131*.