

SELF-GNN: Self-supervised Graph Neural Networks without explicit negative sampling

Zekarias T. Kefato

KTH Royal Institute of Technology
Stockholm, Sweden
zekarias@kth.se

Sarunas Girdzijauskas

KTH Royal Institute of Technology
Stockholm, Sweden
sarunasg@kth.se

ABSTRACT

Real world data is mostly unlabeled or only few instances are labeled. Manually labeling data is a very expensive and daunting task. This calls for unsupervised learning techniques that are powerful enough to achieve comparable results as semi-supervised/supervised techniques. Contrastive self-supervised learning has emerged as a powerful direction, in some cases outperforming supervised techniques.

In this study, we propose, SELF-GNN, a novel contrastive self-supervised graph neural network (GNN) without relying on explicit contrastive terms. We leverage Batch Normalization, which introduces implicit contrastive terms, without sacrificing performance. Furthermore, as data augmentation is key in contrastive learning, we introduce four feature augmentation (FA) techniques for graphs. Though graph topological augmentation (TA) is commonly used, our empirical findings show that FA perform as good as TA. Moreover, FA incurs no computational overhead, unlike TA, which often has $O(N^3)$ time complexity, N – number of nodes.

Our empirical evaluation on seven publicly available real-world data shows that, SELF-GNN is powerful and leads to a performance comparable with SOTA supervised GNNs and always better than SOTA semi-supervised and unsupervised GNNs. The source code is available at <https://github.com/zekarias-tilahun/SelfGNN>.

CCS CONCEPTS

• Information systems → Social networks; • Computing methodologies → Learning latent representations; Neural networks.

KEYWORDS

graph neural networks, self-supervised learning, contrastive learning, graph representation learning, graph data augmentation

ACM Reference Format:

Zekarias T. Kefato and Sarunas Girdzijauskas. 2021. SELF-GNN: Self-supervised Graph Neural Networks without explicit negative sampling. In *SSL '21: The International Workshop on Self-Supervised Learning for the Web, April 19–23, 2021, SSL, Lj*. ACM, New York, NY, USA, 8 pages. <https://doi.org/10.1145/nnnnnnn.nnnnnnn>

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

SSL '21, April 19–23, 2021, SSL, Lj

© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-XXXX-X/18/06... \$15.00
<https://doi.org/10.1145/nnnnnnn.nnnnnnn>

1 INTRODUCTION

Self-Supervised learning (SSL) has emerged as a powerful representation learning paradigm bridging the gap between unsupervised and supervised learning methods. Recent developments across different fields, such as Computer Vision (CV) and Natural Language Processing (NLP), achieve promising results using SSL techniques in some cases performing better than supervised methods [5, 11]. These methods are powered by the so-called *contrastive learning* (CL), which learns by contrasting augmented view of positive and negative objects. That is, given augmented views of the same objects, the CL framework learns by maximizing the compatibility between the respective representation of the views and pushing them apart from representations of a contrastive term (negative sample).

Recent studies [6, 11, 23] in CV propose methods that do not require explicit contrastive terms, while achieving better performance than those with explicit contrastive terms. As a result, now it is fairly understood that for some problems in CV, contrastive learning using either explicit or implicit negative samples performs at least as well as supervised methods [11, 23, 24]. However, it is not yet clear whether contrastive terms should be explicit in GNNs.

Motivated by these findings, this study proposes SELF-GNN, a contrastive self-supervised algorithm for graph neural networks with implicit contrastive terms. SELF-GNN imitates a Siamese network [1], which has been widely used in recent contrastive self-supervised learning methods [5, 6, 11, 13, 14, 23, 24, 26]. While SELF-GNN bares some resemblance to [6, 11], its unique characteristic is in contrast to virtually all SSL methods for graphs that require explicit negative sampling. SELF-GNN uses implicit negative samples that come as a result of batch normalization [23]. Furthermore, SELF-GNN is agnostic to the type of GNN.

As data augmentation is the key component of CL, we introduce four node feature augmentation (FA) strategies, of which some are motivated by related techniques in CV. Though, previously focus has been given to topological augmentation (TA), our empirical finding shows that both TA and FA provide competitive results. However, TA techniques normally require expensive matrix operation with $O(N^3)$ time complexity, while, the proposed FA techniques are simply constant time operations.

The key contribution of this study is the introduction of a new class of self-supervised approaches for GNNs that requires no explicit negative sampling. In addition, we have introduced four FA techniques that offer competitive performance as TA.

Extensive empirical evaluation using seven real-world datasets demonstrate that SELF-GNN achieves comparable performance with

supervised GNNs. Our experiments show that compared to semi-supervised and unsupervised GNN methods it consistently achieves better performance.

2 RELATED WORK

As this study focuses on the intersection of Graph Neural Networks (GNN) and Self-Supervised Learning (SSL), in the following we cover related studies from both topics.

2.1 Graph Neural Networks

Graph Neural Networks represent a family of powerful graph representation learning algorithms [4, 12, 13, 18, 19, 21, 28]. Though they come in different flavors, most of them share an essential framework where representations are learned by propagating node features along the topology of the graph. Unlike other neural networks, such as MLP and CNN, GNNs enable us to construct arbitrary architectures defined by the graph topology. That is, the graph topology itself is what defines the neural network architecture. Therefore, applying consecutive layers of GNNs allows node information to propagate to distant neighbors. Due to the vast amount of literature on GNNs, we shall cover a selected few only.

A basic propagation rule has the form defined in Eq. 1.

$$X^{l+1} = \text{ReLU}(HX^lW^l) \quad (1)$$

where, $H \in \mathbb{R}^{N \times N}$ is a particular materialization of the topology, for example a symmetrically renormalized adjacency matrix, A ($H = \text{symrnorm}(A)$), N number of nodes, $W^l \in \mathbb{R}^{F^{l-1} \times F^l}$ and $X^l \in \mathbb{R}^{N \times F^l}$ are the parameters and activations of the l^{th} layer, respectively, F^l is the number of units of the l^{th} layer. W^l is shared by all nodes and this variant is commonly referred to as Graph Convolutional Network (GCN) [19]. A key limitation of GCN is that, it gives equal importance to features when aggregating. Graph Attention Network (GAT) [31] is later proposed to address this by introducing learnable attention weights of neighborhood features according to their importance.

GCN and GAT require full-batch training, that is, the entire graph (H) should be loaded to memory, which makes them to be transductive and not scalable to large networks. GraphSage [12] alleviates these limitations and proposed a technique based on neighborhood sampling. There, every node apriori specifies a selected few nodes that propagate information. Nonetheless, this has introduced a memory overflow issue for a deeper model, which comes as a result of neighborhood explosion problem. Though, improved methods through layer sampling were proposed by other studies [4, 34], the problem has not been alleviated completely.

Followup studies propose subgraph sampling methods, using clustering (CLUSTERGCN [7]) and graph sampling (GRAPHSAINT [33], MVS-GNN [9]) that are both scalable and do not suffer from the neighborhood explosion problem. These studies pioneer a more efficient mini-batch training than their predecessors, using subgraphs as batch, and they are agnostic to the type of GNN. The key idea is that, the subgraphs will be considered by their own merit and any type of GNN model can be applied on top of them.

2.2 Self-Supervised Learning

Self-supervised learning (SSL) has catalyzed representation learning across disciplines, such as CV, NLP, and GRL, owing to the critical problem of data labeling it tackles. Real-world data is rarely labeled; and often, as a result of its sheer volume, labeling it is very expensive. SSL has emerged as a standard representation learning approach, reaching to and sometimes surpassing the level of pretrained models learned through supervision [5, 6, 11, 13, 24].

Powerful models, like BERT [10] and GPT [2], has revolutionized SSL through the so-called *pre-training and fine-tuning* framework. That is, they pretrain a model based on Transformers on a cheaper pretext task and will fine-tune it on a specifically difficult/expensive task of interest. Difficulty or expensiveness here refers to the lack of sufficient labeled data or in general training data.

Alternatively, Contrastive Learning (CL) has gained popularity in the CV community as a variant of SSL for visual representation [5, 6, 11, 14, 26]. CL is based on data augmentation of a self and contrastive term, where learning is carried out by maximizing similarities of the representations of the augmented views of the same object and minimizing similarity with respect to the contrastive object.

GRL has entertained both the Pre-train fine-tune [15, 16, 22] and CL [13, 30] paradigms. In the former, in general they closely follow the strategies of BERT and GPT families. The later methods rely on augmented views of the original graph topology and design an objective that leverages the augmented view for contrasting. DGI [30] is one CL method that uses a corrupted view of the original graph as an augmentation. Another method, MvGRL [13] on the other hand builds a higher-order network as an augmentation. Though our approach can fall under CL methods, however unlike other methods we do not require explicit negative sampling.

3 METHOD

We start by introducing the preliminaries upon which we build further discussion of the proposal.

3.1 Preliminaries

We consider an undirected graph, $G = (V, E, X)$, with a set of nodes V , and set of edges E , where $N = |V|$ and $M = |E|$. $X \in \mathbb{R}^{N \times F}$ is a node feature matrix, where F is the number of features. The adjacency matrix representation is given by $A = [0, 1]^{N \times N}$, where $A[i, j] = 1$ if $(i, j) \in E$ and 0 otherwise. A diagonal matrix $D \in \mathbb{R}^{N \times N}$ defines the degree distribution of A , and $D[i, i] = \sum_{j=0}^N A[i, j]$. Finally, we define a symmetrically renormalized adjacency matrix as $\tilde{A} = D^{-1/2}(A + I)D^{-1/2}$, and I is the identity matrix.

In essence, CL brings together representations of two augmented views of the same object and repulses them apart from the representation of augmented views of a contrastive object, negative sample.

A question that has been carefully investigated in CV and yet to be understood clearly in graph representation learning is whether we can drop negative samples or use them implicitly without compromising performance [11]. In CV, it has been shown that contrastive learning with implicit negative sampling [11] or hard negative sampling [24] can achieve as good a result as a pre-training with supervision. This motivates us to investigate the former direction and shed some insights by avoiding explicit negative sampling in GRL, as the later is shown to be useful in GRL. As a result, we

rely on batch normalization (BATCHNORM) that introduces implicit negative samples [29].

Note that, existing self-supervised methods for graphs rely on one or more of the following techniques:

- (1) Pre-training and fine-tuning paradigm [15, 16]
- (2) CL with explicit negative samples [13]
- (3) Self-supervised learning with few class labels [27]

In contrast with 1 and 2, our approach requires only positive augmented views as input, the contrastive term will be provided through a BATCHNORM, and in contrast with 3, no label information, whatsoever, is used during training.

As a standard contrastive learning framework requires a data augmentation strategy, we first introduce the graph data augmentation techniques we use in this study.

3.2 Graph Data Augmentations

Data augmentation in other domains, such as CV, has been extensively studied. As a result a set of standard augmentation techniques, such as cropping, rotating, blurring are commonly used. However, there are no established data augmentation techniques for graphs [13]. In this study, we propose two family of data augmentation techniques based on topology and features. To the best of our knowledge, this is the first study proposing graph data augmentation techniques based on both topology and features..

Topology augmentation. The aim of a topological data augmentation is to uncover a different topological view of the original graph by exploiting the properties of the graph structure. A popular augmentation technique uses a higher-order network that is computed through a general graph diffusion process [20]. Employing high-order networks obtained through a diffusion process have been shown to improve GNNs performance [20]. In this study, we use two flavors of the popular PAGERANK algorithm, which are PAGERANK based on rooted random walks (commonly referred as personalized PAGERANK–PPR), and heat-kernel (HK) PAGERANK [8], given in equations 2, 3 respectively. We propose a third high-order network construction technique based on Katz-index as shown in Eq. 4.

$$H^{PPR} = \alpha(I - (1 - \alpha)\tilde{A})^{-1} \quad (2)$$

$$H^{HK} = \exp(tAD^{-1} - t) \quad (3)$$

$$H^{katz} = (I - \beta\tilde{A})^{-1}\beta\tilde{A} \quad (4)$$

where α is a teleportation probability and t is diffusion time, and β is a decay parameter. Katz-index is the weighted sum of the set of all paths between a pair of nodes, paths are penalized according to their length. The attenuation factor (β) governs the penalization.

Feature Augmentation. While graph data augmentation in SOTA largely focuses on the topological view, little attention has been given to feature augmentation. Furthermore, a study [13] has argued against feature augmentation techniques, such as masking, and adding noise. In this study, we propose the following feature augmentation techniques other than masking and noising.

- (1) *Split*: the first feature augmentation technique is inspired by cropping for image augmentation, and it creates an augmented view of the features by splitting them into two as $X = X[:, : F/2]$ and $X' = X[:, F/2 :]$. That is, the first half of the feature dimension is used to build one view and while

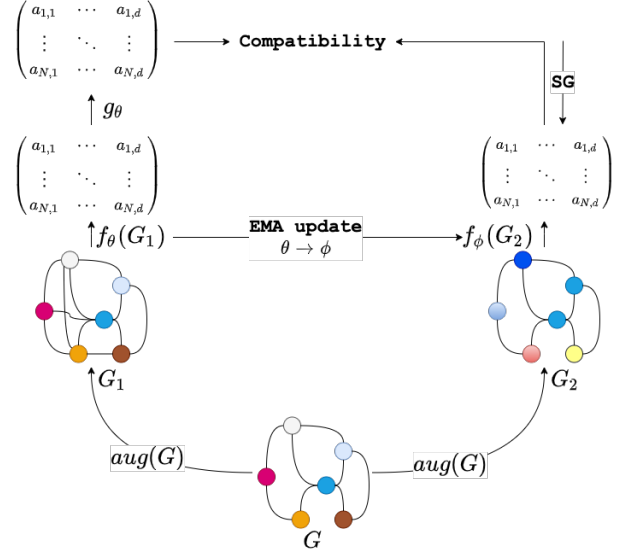


Figure 1: SELF-GNN’s architecture, $aug(\cdot)$ is a graph data augmentation function, f_θ and f_ϕ are GNN encoders parameterized by θ and ϕ , and g_ψ is a MLP, SG (stop gradient), and EMA (exponential moving average). G_1 and G_2 are the topological and feature augmentations of the graph, G , respectively.

the remaining half is used to build the second view. Note that both views are constructed simultaneously.

- (2) *Standardize*: motivated by scaling in CV, this technique simply applies a standardization of the features by applying a z -score standardization as $X' = (\frac{X^T - \bar{x}}{s})^T$, where $\bar{x} \in \mathbb{R}^{F \times 1}$ and $s \in \mathbb{R}^{F \times 1}$ are the mean and standard deviation vectors associated to each feature. Though the values are scaled, the signal encoded in X' is equivalent to that of X , hence providing a scaled view of the original features.
- (3) *Local Degree Profile (LDP)*: several real-world graphs come with no features associated to nodes, and [3] propose a mechanism for building node features based on five statistics computed from its local degree profile, $X' \in \mathbb{R}^{N \times 5}$. We use these features to generate an augmented view of the graph where LDP is the node feature. We apply zero padding on X' , so that the feature dimension of X and X' match, $X' \in \mathbb{R}^{N \times F}$.
- (4) *Paste*: is a feature augmentation technique that simply combines X with the LDP features, such that the augmented feature $X' \in \mathbb{R}^{N \times (F+5)}$. In this case a zero padding is applied on the original features, such that $X \in \mathbb{R}^{N \times (F+5)}$.

In Fig. 1, $aug(G)$ allows us to sample and apply augmentation on the topology or features of the graph, G .

3.3 Architecture

The proposed architecture, shown in Fig. 1, mimics a Siamese network [1] that is commonly used in recent contrastive self-supervised models for representation learning [5, 6, 11, 13, 23, 24, 26]. It has two parallel networks, referred to as a student (left hand side) and teacher (right hand side) networks [6, 11]. The first component

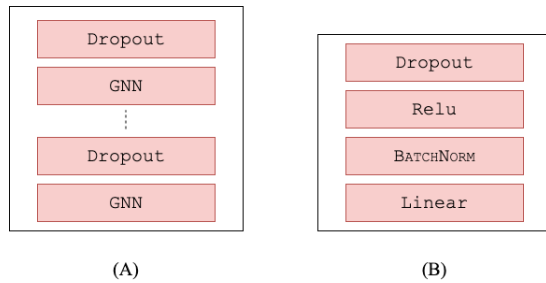


Figure 2: The architecture of (A) the encoder, f , and (B) prediction or projection block, g_θ . The forward propagation is from the lower to upper layers.

of both networks is similar, they both use stacked GNN encoder functions f_θ and f_ϕ , parameterized by the sets of parameters θ and ϕ (shown in Fig. 2(A)). The encoder produces the representation of the input graph, and this is what we use for downstream graph analytic tasks. f is referred to as the representation block, and $X_1 = f_\theta(G_1)$, $X_2 = f_\phi(G_2)$. Note that the architecture is GNN encoder agnostic, thus any type of GNN can be used.

The second block in most Siamese networks [5, 6, 11] is a projection head (Shown in Fig. 2(B)). In this study, we have investigated the necessity of this head and empirically found out that for GRL it adds no qualitative gain, unlike as it is suggested for CV tasks [5]. Therefore we drop the projection head.

Finally, one key difference between the student and the teacher network is that, the student network applies a prediction block, a MLP or the function g_θ over the output of the representation block, f_θ . Its architecture is shown in Fig. 2(B). The prediction block act as the brain of the student who is trying to learn what is produced by the teacher network, $g_\theta(X_1) \approx X_2$

Besides their architecture, the second key difference between the two networks is that, the parameters of the student network are updated through a back-propagation of gradients. Whereas, the parameters of the teacher network are updated using an exponential moving average (EMA) of the parameters of the student network.

Therefore, the model is trained using two joint operations, which are (1) student parameter updates using gradients and (2) teacher parameter updates using an EMA of the student parameters. In the first case, the loss function specified using the mean squared error as in Eq. 5 is used to obtain the gradients.

$$\mathcal{L}_\theta = 2 - 2 \cdot \frac{\langle g_\theta(X_1), X_2 \rangle}{\|g_\theta(X_1)\|_F \cdot \|X_2\|_F} \quad (5)$$

The key idea pointed out in [11] is that gradients are computed with respect to θ only, and as indicated by the SG symbol on Fig. 1, the gradient flow on the teacher network is blocked. Instead, the EMA of the student parameters is used as in Eq. 6 in order to update the teacher parameters.

$$\phi \leftarrow \tau\phi + (1 - \tau)\theta \quad (6)$$

where τ is a decay rate.

Though it seems counter-intuitive that one can learn using positive examples only, however, it has been analytically proven that the prediction block, g_θ , and the BATCHNORM introduce implicit

Datasets	Nodes	Edges	Features	Classes
Cora	2,708	5,278	1,433	7
Citeseer	3,327	4,552	3,703	6
Pubmed	19,717	44,324	500	3
Photo	7,487	119,043	745	8
Computers	13,381	245,778	767	10
CS	18,333	81,894	6,805	15
Physics	34,493	247,962	8,415	5

Table 1: Summary of the datasets

negative samples [29]. Thus, similar to those that explicitly sample contrastive terms either from the batch [5] or entire dataset [13, 30], the implicit contrastive terms prevents the model from converging to the trivial solution of collapsing into a constant point.

3.4 Improving SELF-GNN

SELF-GNN in its current form has two efficiency problems (1) the TAs involve matrix inversion, which incurs $O(N^3)$ run time, and (2) the TAs lead to a significant increase in the number of edges. The second problem is not favorable for full-batch GNNs in terms of GPU memory usage. However, we propose a strategy to mitigate the severity of the aforementioned problems.

For this reason, we use subgraph sampling used in CLUSTERGCN [7]. That is, first we create initial clusters using METIS [17], and then we randomly merge some clusters to create a final set of subgraphs (batches). Then, instead of the complete graph, we apply the TAs over each subgraph independently and SELF-GNN is trained using batches. This variant of SELF-GNN is dubbed CLUSTERSELF-GNN. Section 4 empirically corroborates the suggested improvement.

4 EMPIRICAL EVALUATION

We evaluate the performance of SELF-GNN using publicly available real-world datasets and compare its performance against different GNN architectures and SSL baselines.

4.1 Datasets

We use seven datasets provided by the PyTorch Geometric library¹. Three popular citation networks (Cora, Citeseer, Pubmed), two author collaboration networks [25] (CS, Physics) from two categories of the Arxiv database, and co-purchased products network [25] (Photo, Computers) from two categories of Amazon. The summary of the datasets is given in Table 1.

4.2 Baselines

We compare the proposed method against the original semi-supervised and self-supervised GNN models. We select three popular semi-supervised GNNs, which are GCN [19], GAT [31], and GRAPH-SAGE [12], and two self-supervised GNNs, MvGRL [13] and DGI [30]. Since MvGRL requires topological data augmentation like ours, we use three variants MvGRLPPR, MvGRLHK, and MvGRLKATZ denoting the associated augmentation techniques, which are personalized PageRank, heat kernel, and Katz-index, respectively. A brief discussion of the baselines can be found in Section 2.

¹<https://pytorch-geometric.readthedocs.io/en/latest/>

4.3 Experimental Setting

Dataset: a public split (Planetoid split [32]) of the citation networks is already available, and hence we use this split for the citation datasets. Since no public split is available for the rest, we randomly split them as (70/10/20–train/validation/test) by respecting the class proportion in each split.

Hyperparameters: all the algorithms are trained for 1,000 epochs, and for all of them the validation set is used to choose the best epoch for evaluating the model using the test set. SELF-GNN has the same architecture and setting in all the experiments. That is, the encoder has two GNN layers with 512 and 128 units each followed by a dropout layer, and uses the same dropout and learning rate of 0.2 and 0.0001, respectively. Since the representation block has 128 output units, we use 128 as the representation size for all the baselines. The hyperparameters of the topological augmentations are set to $\alpha = 0.15$, $t = 3$, and $\beta = 0.1$.

Machine: we use 2 NVidia Quadro RTX 5000 with NVLink, each with 3072 CUDA cores, 16 GB GDDR6 memory, and Ubuntu 19.10.

Source codes: except DGI and MVGRL baselines, where we use the source code provided by the authors, for the rest of the baselines we use the PyTorch Geometric² library.

Workload: For all the datasets the standard work-load is node classification, and the evaluation metric is accuracy.

4.4 Results

In a series of experiments, we compare SELF-GNN against the original semi-supervised GNNs and self-supervised GNNs on node classification and report the accuracy.

Experiment 1: Comparison with the Original GNNs

In this experiment, the goal is to investigate how well SELF-GNN compares to the original GNN models (dubbed Original). Note, the original architectures are supervised using a small fraction of the labels, and in contrast our model has absolutely no supervision during training. Once node embeddings are learned, we use a logistic regression classifier to classify the test set. We use 5-fold cross validation, and the results reported in Table 2 are the mean on the 40% fold. SELF-GNN gives a comparable result as the original methods, in fact achieving better performance for the citation networks.

Interestingly, albeit expected, the original methods are relatively hungry for labeled data. For the citation networks, only a fraction of the training nodes were labeled, that is a rate of 0.0563, 0.0569, 0.0030 is used for Cora, Citeseer, and Pubmed, respectively. Whereas, for the rest of the datasets all the training nodes are labeled, fully supervised, and in these cases unsurprisingly SELF-GNN performs lower than the original models.

Furthermore, we observe that the feature augmentation techniques lead to equivalent performance as topological augmentation techniques, except LDP in most cases and PASTE for the Pubmed datasets. From the topological augmentations, PPR and HK tend to perform better than KATZ, except for the Photo dataset. On the other hand, Split and Standard consistently perform better than the other feature augmentation techniques.

Finally, for the relatively large graphs (CS and Physics), where the topological augmentation produces more than a million edges,

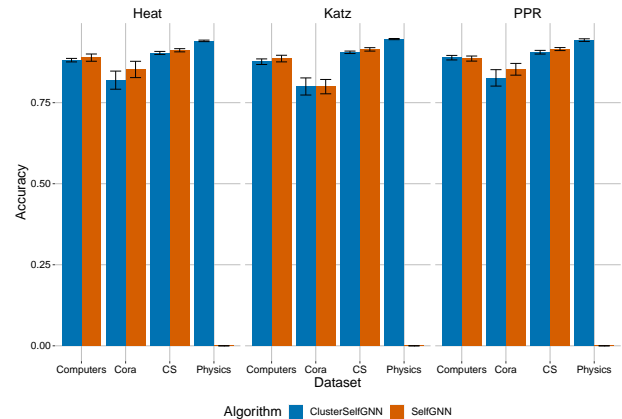


Figure 3: Performance comparison between the subgraph sampling (CLUSTERSELF-GNN) and full batch (SELF-GNN) based models

SELF-GNN runs out of GPU memory, as indicated by the dashed entries in the table.

Experiment 2: Improving Performance

This experiment demonstrates the proposed remedy for the out-of-memory problem pointed out in the previous experiment. We show a comparison between the CLUSTERSELF-GNN and SELF-GNN to highlight the fact that CLUSTERSELF-GNN can deal with this issue just by sacrificing a small drop in terms of performance, as shown in Fig. 3. The comparison on the datasets other than Physics highlights that CLUSTERSELF-GNN is closely comparable with SELF-GNN. However, the main purpose of CLUSTERSELF-GNN is for relatively large graphs that do not fit in the GPU memory. Thus, we use the Physics dataset, where SELF-GNN runs out of memory and demonstrate CLUSTERSELF-GNN’s power as in the figure. The results reported for CLUSTERSELF-GNN, with all the three topological augmentations, are in par with the full-batch feature augmentations.

Experiment 3: Comparison with Self-supervised GNNs

Next, we perform an experiment to compare SELF-GNN against two of the self-supervised methods DGI [30] and MVGRL [13]. Details on the settings of this experiment are given in Appendix A. Table 3 reports the results of this experiment, and we can see that SELF-GNN mostly performs better than the baselines. In MVGRL paper [13], the reported results for Cora, Citeseer, and Pubmed are 86.8 ± 0.5 , 73.3 ± 0.5 , and 80.1 ± 0.7 , respectively. However, first the representation dimension is 512 and second it is not stated whether they use a public or their own split of the datasets. As pointed out in a study [25], different splits on the three datasets lead to a remarkably huge difference in performance.

4.5 Ablation Study

4.5.1 Effect of BATCH-NORM. As the key to avoiding explicit negative sampling is BATCH-NORM, in the first experiment we analyze its effect on the performance of SELF-GNN. As a result, we train SELF-GNN with and without BATCH-NORM.

²<https://pytorch-geometric.readthedocs.io/en/latest/>

Dataset	GNN Type	Algorithms							
		Original	SELFNN variants						
			PPR	HK	KATZ	SPLIT	STANDARD	PASTE	LDP
Cora	GCN	79.9±0.003	85.3±0.02	85.2±0.02	79.9±0.02	84.4±0.02	84.6±0.03	81.9±0.02	78.9±0.02
	GAT	79.1±0.004	84.2±0.02	84.9±0.02	83.8±0.01	85.0±0.01	85.1±0.02	82.2±0.03	80.7±0.01
	GRAPHsAGE	78.7±0.01	84.7±0.01	84.9±0.01	79.9±0.01	81.7±0.01	84.5±0.01	81.9±0.01	68.6±0.01
Citeseer	GCN	65.7±0.01	70.6±0.01	70.5±0.02	65.6±0.02	72.3±0.01	69.8±0.01	71.2±0.02	61.2±0.01
	GAT	66.3±0.01	71.4±0.02	71.2±0.02	66.5±0.03	71.4±0.01	68.0±0.03	70.3±0.01	60.1±0.02
	GRAPHsAGE	63.6±0.01	70.5±0.02	68.8±0.01	63.8±0.01	72.3±0.01	65.7±0.01	69.7±0.01	54.5±0.02
Pubmed	GCN	79.0±0.005	82.1±0.01	82.0±0.01	81.0±0.01	81.3±0.01	82.3±0.01	76.0±0.01	77.1±0.01
	GAT	77.5±0.005	82.7±0.01	82.6±0.01	80.8±0.01	81.1±0.02	80.6±0.02	76.1±0.01	76.7±0.02
	GRAPHsAGE	76.2±0.01	80.2±0.01	80.4±0.01	80.7±0.02	79.5±0.01	80.8±0.02	72.2±0.02	68.6±0.02
Photo	GCN	94.3±0.002	92.9±0.01	93.4±0.01	93.3±0.01	92.1±0.01	92.0±0.01	91.6±0.01	90.4±0.01
	GAT	94.4±0.005	92.7±0.01	93.1±0.01	93.2±0.01	93.2±0.01	92.2±0.01	91.7±0.01	91.7±0.01
	GRAPHsAGE	95.8±0.001	92.4±0.004	92.7±0.01	93.8±0.01	92.1±0.004	92.0±0.01	89.9±0.01	85.1±0.01
Computers	GCN	90.9±0.003	88.6±0.01	88.8±0.01	88.6±0.01	88.8±0.01	87.2±0.01	87.9±0.01	86.6±0.01
	GAT	90.9±0.004	88.6±0.01	88.8±0.01	88.3±0.01	88.6±0.01	88.0±0.01	87.6±0.01	87.9±0.01
	GRAPHsAGE	91.2±0.003	87.2±0.01	87.0±0.01	87.4±0.01	87.2±0.01	86.6±0.01	86.1±0.01	82.4±0.004
CS	GCN	93.2±0.001	91.5±0.004	91.1±0.004	91.4±0.004	92.9±0.001	91.7±0.01	91.9±0.01	86.3±0.01
	GAT	92.8±0.002	91.2±0.004	91.4±0.004	91.1±0.004	91.9±0.01	90.7±0.004	92.3±0.01	85.5±0.01
	GRAPHsAGE	94.0±0.001	–	–	–	93.0±0.001	90.4±0.003	92.9±0.004	86.7±0.003
Physics	GCN	96.5±0.001	–	–	–	95.5±0.003	94.8±0.0	95.5±0.002	95.0±0.002
	GAT	96.2±0.001	–	–	–	95.2±0.002	94.6±0.002	95.1±0.003	94.6±0.003
	GRAPHsAGE	–	–	–	–	–	–	–	–

Table 2: Comparison of three (GCN, GAT, and GRAPHsAGE) type of the original GNNs against the different variants of SELFNN on node classification experiment. The reported results are the mean classification accuracy along with the standard deviation. For each dataset and each GNN type, the best performing algorithm highlighted in bold. We put a blank when an algorithm fails to finish as a result of running out of GPU memory .

Dataset	Algorithms							
	SELFbFA	SELFPPR	SELFHK	SELFKATZ	MvGRLPPR	MvGRLHK	MvGRLKATZ	DGI
Cora	81.0±0.18	78.98±0.25	79.01±0.31	77.71±0.34	75.3±0.72	74.5±0.71	74.5±0.6	81.0±0.24
Citeseer	67.19±0.37	68.08±0.48	68.01±0.43	66.14±0.32	58.2±0.0	57.5±0.07	61.5±0.0	66.3±0.28
Pubmed	80.59±0.19	78.05±0.15	77.98±0.14	77.40±0.17	73.2±0.37	74.9±0.43	74.5±0.35	79.8±0.19
Photo	90.86±0.1	93.35±0.19	93.38±0.16	93.65±0.18	91.0±0.98	91.4±0.80	90.8±2.30	92.6±0.15

Table 3: Comparison with self-supervised GNNs. The reported results are the mean classification accuracy along with the standard deviation of 50 runs. Bold indicates the best performing algorithm. SELFbFA is the best of feature augmentation, and SPLIT consistently performs better.

Fig. 4 shows our empirical finding. Notice that, without BATCHNORM SELFNN’s performance is not stable and often suffers significantly.

As an alternative to BATCHNORM we have investigated layer normalization (LAYERNORM), and our finding shows that LAYERNORM behaves similar to the no BATCHNORM case in Fig. 4.

4.5.2 *Effect of Projection head.* A projection head is commonly used in most recent CL methods, however, SELFNN’s architecture does not make use of this head. As a result, we use this ablation study to empirically justify our choice in removing the projection head from SELFNN. Fig. 5 shows the result of node classification experiment with (Yes) and without (No) a projection head. As it can

be seen from the figure, no significant improvement is achieved by adding the projection head. In addition, though in some cases it appears that it gives comparable performance, nonetheless it is not stable as indicated by the relatively higher variance.

4.5.3 *Effect of Perturbation on SPLIT.* The SPLIT augmentation requires splitting the feature along the feature dimension. One might ask, rightly so, how sensitive this augmentation technique is to the perturbation of the features. To this end, before splitting we apply perturbation along the feature dimension as $X = X[:, perm]$, where $perm$ is a random permutation of the features. Fig. 6 shows that overall the permutation almost has no impact on the outcome of SELFNN using SPLIT.

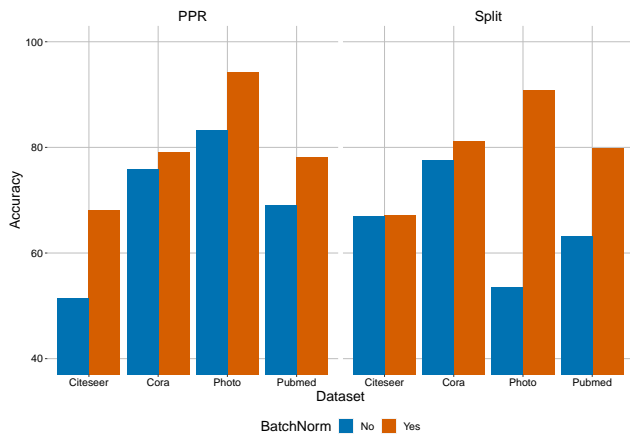


Figure 4: SELF-GNN’s performance with (Yes) and without (NO) BATCHNORM

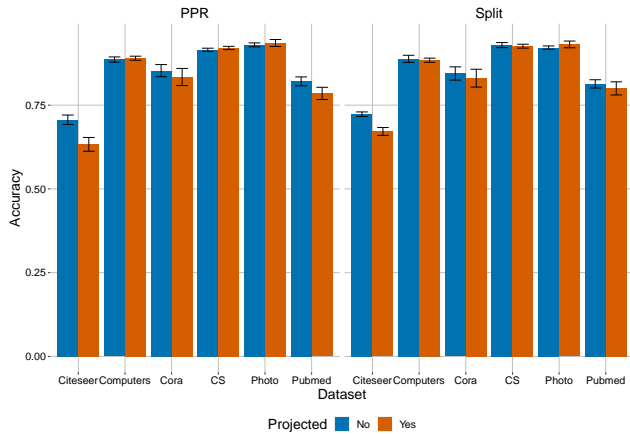


Figure 5: Comparison of SELF-GNN with (Yes) and without (No) a projection head using the PPR and Split data augmentation techniques.

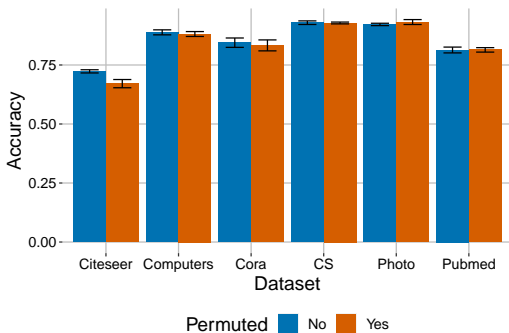


Figure 6: Analysis of the permutation of the features in the Split augmentation.

Dataset	Augmentation	Dropout	Epochs
Cora	SPLIT	0.35	1500
Cora	PPR	0.8	200
Cora	HK	0.7	200
Cora	KATZ	0.7	200
Citeseer	SPLIT	0.2	1500
Citeseer	PPR	0.8	400
Citeseer	HK	0.7	400
Citeseer	KATZ	0.7	400
Pubmed	SPLIT	0.2	1500
Pubmed	PPR	0.4	400
Pubmed	HK	0.4	400
Pubmed	KATZ	0.4	400
Photo	SPLIT	0.2	1500
Photo	PPR	0.65	400
Photo	HK	0.65	400
Photo	KATZ	0.65	400

Table 4: The dropout rate and number of epochs used for experiment three

5 CONCLUSION AND FUTURE WORK

This study proposes SELF-GNN, a novel contrastive self-supervised method for GNNs, which does not require explicit contrastive terms, negative samples. Though negative samples are critical to the success of contrastive learning, we employ batch normalization that is shown to introduce implicit negative samples. Furthermore, we introduce four node feature augmentation techniques that are as effective as topological ones.

We carried out extensive experiments using seven real-world datasets and show that SELF-GNN achieves a comparable performance with supervised GNNs, while performing significantly better than semi-supervised and self-supervised methods.

SELF-GNN relies on two parallel GNNs to be loaded into memory at the same time, this causes a major bottleneck for large networks. Though a clustering based improvement is suggested in this study, a careful and principled work needs to be done to properly address the problem. This is our goal for a future work.

A SETTINGS FOR EXPERIMENT 3

Experiment 3 compares SELF-GNN against SOTA Self-supervised methods; and all of them are trained using the entire dataset.

For evaluation, we closely follow [30]; that is, we train a logistic regression classifier using the training partition (The Planetoid [32] partition for the three citation networks, and our own partition for Photo dataset). The validation set is used for tuning the hyperparameters of our model. Finally, the results reported in Table 3 are using the test partition. The source code provides all these details.

In Table 4, we provide the dropout rates and number of epochs used for this experiment. Other than these two, all the hyperparameters are fixed. We use a two layer GCN [19] with 512, 128 units, and learning rate 0.0001.

We observe that topological augmentations (TA) require strong regularization and small number of epochs. On the other-hand, feature augmentation (FA) methods are gently penalized and require

large number of epochs. We conjecture that this is due to the high-order network used in TA that enables faster feature propagation than FA, which relies on the first-order network.

REFERENCES

- [1] Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. 1993. Signature Verification Using a "Siamese" Time Delay Neural Network. In *Proceedings of the 6th International Conference on Neural Information Processing Systems (Denver, Colorado) (NIPS'93)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 737–744.
- [2] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. arXiv:2005.14165 [cs.CL]
- [3] Chen Cai and Yusu Wang. 2019. A simple yet effective baseline for non-attributed graph classification. arXiv:1811.03508 [cs.LG]
- [4] Jie Chen, Tengfei Ma, and Cao Xiao. 2018. FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling. arXiv:1801.10247 [cs.LG]
- [5] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. 2020. A Simple Framework for Contrastive Learning of Visual Representations. In *Proceedings of the 37th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 119)*. PMLR, Virtual, 1597–1607.
- [6] Xinlei Chen and Kaiming He. 2020. Exploring Simple Siamese Representation Learning. arXiv:2011.10566 [cs.CV]
- [7] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. 2019. *Cluster-GCN: An Efficient Algorithm for Training Deep and Large Graph Convolutional Networks*. Association for Computing Machinery, New York, NY, USA, 257–266. <https://doi.org/10.1145/3292500.3330925>
- [8] F. Chung. 2007. The heat kernel as the pagerank of a graph. *Proceedings of the National Academy of Sciences* 104 (2007), 19735 – 19740.
- [9] Weilin Cong, Rana Forsati, Mahmut Kandemir, and Mehrdad Mahdavi. 2020. *Minimal Variance Sampling with Provable Guarantees for Fast Training of Graph Neural Networks*. Association for Computing Machinery, New York, NY, USA.
- [10] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv:1810.04805 [cs.CL]
- [11] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre H. Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Daniel Guo, Mohammad Gheshlaghi Azar, Bilal Piot, Koray Kavukcuoglu, Rémi Munos, and Michal Valko. 2020. Bootstrap your own latent: A new approach to self-supervised Learning. arXiv:2006.07733 [cs.LG]
- [12] William L. Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems (Long Beach, California, USA) (NIPS'17)*. Curran Associates Inc., Red Hook, NY, USA, 1025–1035.
- [13] Kaveh Hassani and Amir Hosein Khasahmadi. 2020. Contrastive Multi-View Representation Learning on Graphs. arXiv:2006.05582 [cs.LG]
- [14] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. 2020. Momentum Contrast for Unsupervised Visual Representation Learning. arXiv:1911.05722 [cs.CV]
- [15] Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. 2020. Strategies for Pre-training Graph Neural Networks. arXiv:1905.12265 [cs.LG]
- [16] Ziniu Hu, Yuxiao Dong, Kuansan Wang, Kai-Wei Chang, and Yizhou Sun. 2020. *GPT-GNN: Generative Pre-Training of Graph Neural Networks*. Association for Computing Machinery, New York, NY, USA, 1857–1867.
- [17] George Karypis and Vipin Kumar. 1998. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM J. Sci. Comput.* 20, 1 (Dec. 1998).
- [18] Zekarias T. Kefato and Sarunas Girdzijauskas. 2020. Gossip and Attend: Context-Sensitive Graph Representation Learning. In *In Proc. of the 14th AAAI International Conference on Web and Social Media (ICWSM'121)*. arXiv:2004.00413 [cs.LG]
- [19] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. arXiv:1609.02907 [cs.LG]
- [20] Johannes Klicpera, Stefan Weissenberger, and Stephan Günnemann. 2019. Diffusion Improves Graph Learning. arXiv:1911.05485 [cs.SI]
- [21] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. DeepWalk: Online Learning of Social Representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (New York, New York, USA) (KDD '14)*. ACM, New York, NY, USA, 701–710.
- [22] Jiezhong Qiu, Qibin Chen, Yuxiao Dong, Jing Zhang, Hongxia Yang, Ming Ding, Kuansan Wang, and Jie Tang. 2020. *GCC: Graph Contrastive Coding for Graph Neural Network Pre-Training*. Association for Computing Machinery, New York, NY, USA, 1150–1160. <https://doi.org/10.1145/3394486.3403168>
- [23] Pierre H. Richemond, Jean-Bastien Grill, Florent Altché, Corentin Tallec, Florian Strub, Andrew Brock, Samuel Smith, Soham De, Razvan Pascanu, Bilal Piot, and Michal Valko. 2020. BYOL works even without batch statistics. arXiv:2010.10241 [stat.ML]
- [24] Joshua Robinson, Ching-Yao Chuang, Suvrit Sra, and Stefanie Jegelka. 2021. Contrastive Learning with Hard Negative Samples. arXiv:2010.04592 [cs.LG]
- [25] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. 2019. Pitfalls of Graph Neural Network Evaluation. arXiv:1811.05868 [cs.LG]
- [26] Aravind Srinivas, Michael Laskin, and Pieter Abbeel. 2020. CURL: Contrastive Unsupervised Representations for Reinforcement Learning. arXiv:2004.04136 [cs.LG]
- [27] Ke Sun, Zhouchen Lin, and Zhanxing Zhu. 2020. Multi-Stage Self-Supervised Learning for Graph Convolutional Networks on Graphs with Few Labels. arXiv:1902.11038 [cs.LG]
- [28] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. LINE: Large-Scale Information Network Embedding. In *Proc. of the 24th International Conference on World Wide Web (Florence, Italy) (WWW '15)*. 1067–1077.
- [29] Yuandong Tian, Lantao Yu, Xinlei Chen, and Surya Ganguli. 2020. Understanding Self-supervised Learning with Dual Deep Networks. arXiv:2010.00578 [cs.LG]
- [30] Petar Velickovic, W. Fedus, William L. Hamilton, P. Liò, Yoshua Bengio, and R. Devon Hjelm. 2019. Deep Graph Infomax.
- [31] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. arXiv:1710.10903 [stat.ML]
- [32] Zhilin Yang, William W. Cohen, and Ruslan Salakhutdinov. 2016. Revisiting Semi-Supervised Learning with Graph Embeddings. arXiv:1603.08861 [cs.LG]
- [33] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. 2020. GraphSAINT: Graph Sampling Based Inductive Learning Method. arXiv:1907.04931 [cs.LG]
- [34] Difan Zou, Ziniu Hu, Yewen Wang, Song Jiang, Yizhou Sun, and Quanquan Gu. 2019. Layer-Dependent Importance Sampling for Training Deep and Large Graph Convolutional Networks. arXiv:1911.07323 [cs.LG]