Jointly Learnable Data Augmentations for Self-Supervised GNNs

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

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

Hannes Stärk
Technical University of Munich
Munich, Germany
hannes.staerk@tum.de

ABSTRACT

Self-supervised Learning (SSL) aims at learning representations of objects without relying on manual labeling. Recently, a number of SSL methods for graph representation learning have achieved performance comparable to SOTA semi-supervised GNNs. A Siamese network, which relies on data augmentation, is the popular architecture used in these methods. However, these methods rely on heuristically crafted data augmentation techniques. Furthermore, they use either contrastive terms or other tricks (e.g., asymmetry) to avoid trivial solutions that can occur in Siamese networks.

In this study, we propose, GraphSurgeon, a novel SSL method for GNNs with the following features. First, instead of heuristics we propose a learnable data augmentation method that is jointly learned with the embeddings by leveraging the inherent signal encoded in the graph. In addition, we take advantage of the flexibility of the learnable data augmentation and introduce a new strategy that augments in the embedding space, called post augmentation. This strategy has a significantly lower memory overhead and runtime cost. Second, as it is difficult to sample truly contrastive terms, we avoid explicit negative sampling. Third, instead of relying on engineering tricks, we use a scalable constrained optimization objective motivated by Laplacian Eigenmaps to avoid trivial solutions.

To validate the practical use of GraphSurgeon, we perform empirical evaluation using 14 public datasets across a number of domains and ranging from small to large scale graphs with hundreds of millions of edges. Our finding shows that GraphSurgeon is comparable to six SOTA semi-supervised and on par with five SOTA self-supervised baselines in node classification tasks.

CCS CONCEPTS

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

KEYWORDS

GNN, Self-Supervised Learning, Data Augmentation

1 INTRODUCTION

Owing to its comparable performance to semi-supervised learning, self-supervised learning (SSL) has been widely adapted across a number of domains. In SSL, we seek to learn representations of objects (e.g. images and graphs) without relying on manual labeling. A particular interest in this study is SSL for graph neural networks (GNN) - ssl-gnn.

Earlier efforts in SSL concentrated on devising a pretext task and training a model to extract transferable features. However, it is challenging to learn a generalized representation that is invariant to the pretext task and useful for a downstream task [22]. As a result, a new framework based on a Siamese network [5] has become the de facto standard, and current SOTA results are achieved using different flavors of such networks [7, 8, 11, 15, 18–20, 22, 34–36, 40].

Self-supervised techniques using Siamese networks learn representations that benefit from data-augmentation (perturbation) [36], and devising suitable augmentation techniques is one of the main challenges. Although there are well established data-augmentation techniques (e.g., for images), there are no standard techniques for other modalities, such as text and graph data. As a result, existing methods depend on different heuristics and/or trial and error [34].

In a Siamese framework, there are two networks, commonly referred to as a *student* and *teacher* network, and their inputs are differently augmented views of the same object. Learning is carried out by maximizing the agreement between the outputs of the two networks. However, a contrastive term (negative samples) is required to prevent a trivial solution (e.g., constant output, collapse) [7, 11, 18, 19, 27, 34, 39, 40]. On the other hand, because it is usually difficult to sample truly contrastive (negative) terms, techniques were proposed without the need for explicit negative samples. These methods rely on engineering tricks (e.g., asymmetry) to avoid a trivial solution [8, 12, 15, 20, 28]. Three strategies are commonly used, which are asymmetry in the (1) architecture (2) weights, and (3) update rules [15].

In this study, we propose an SSL model for GNNs based on the Siamese architecture called GraphSurgeon (**s**elf-s**u**pervised **G**NN that jointly l**e**arns t**o** augme**n**t). Unlike most existing methods, GraphSurgeon requires neither data augmentation using heuristics nor explicit negative samples. We design an SSL architecture in such a way that data augmentation is jointly learned with the graph representation. Moreover, by using a principled constrained optimization objective, we avoid the need for explicit negative samples.

To augment a given node v, we simply use two augmentation heads $v_1 = f_{\theta_1}(v)$ and $v_2 = f_{\theta_2}(v)$, parameterized by two separate sets of weights θ_1 and θ_2, which produce two views v_1 and v_2 of v. For example, $f_{\theta_1}(v)$ and $f_{\theta_2}(v)$ could be simple MLP heads. v_1 and v_2 are fed to a shared GNN, $h_\theta(\cdot)$, parameterized by a set of weights θ. Here, the key idea is that the parameters of the augmentation heads,

θ_1 and θ_2, are jointly learned with θ. Because the two networks are symmetric and equivalent, we use the terms *left* and *right*, instead of student and teacher.

Furthermore, because of the flexibility of the learnable augmenters, we introduce an alternative new strategy called *post-augmentation*. Post-augmentation applies augmentation in a latent space. That is, we first encode (e.g., using a GNN, CNN, or Transformer) and augment the encoded representations. This is in contrast to the standard practice which we refer to as *pre-augmentation*, where we first augment the input features and then encode them. Compared to the pre-augmentation strategy, we show that post-augmentation significantly decreases memory use and computation time. Fig. 1 shows the pre and post architectures. Note that post-augmentation is difficult, if not impossible, when using predefined augmentation strategies.

GraphSurgeon is trained based on a loss function that draws inspiration from Laplacian Eigenmaps [2]. Just like any Siamese network, it has a term to maximize the agreement between the outputs of the left and right networks. To avoid trivial solutions, instead of engineering tricks, we use an orthonormality constraint similar to Laplacian Eigenmaps. The constraint encourages positive pairs to be similar while preventing trivial solutions.

Finally, we perform an empirical evaluation using publicly available datasets and compare GraphSurgeon with strong SOTA baselines on three types of node classification tasks. The results show that GraphSurgeon is comparable to semi-supervised methods, on average just 2 percentage points away from the best performing ones. However, it is on par with the ssl-gnn baselines.

Our contributions are summarised as follows:

- We propose a ssl-gnn called GraphSurgeon that jointly learns to embed and augment graph data.
- We introduce an alternative new augmentation strategy that happens in the latent space as opposed to the input space and it leads to a significant improvement in resource usage, regarding GPU memory and run-time.
- GraphSurgeon learns meaningful representations based on a constrained optimization objective that has theoretical motivation as opposed to engineering tricks to prevent trivial solutions.
- We carried out extensive experiments using 14 publicly available datasets ranging from small to large, spanning a number of domains and we show that GraphSurgeon is comparable to six SOTA supervised GNNs and on par with five SOTA ssl-gnn architectures.

To the best of our knowledge, this is the first attempt towards ssl-gnn that jointly learns data augmentation and also does not require explicit contrastive terms. In addition, one can easily adopt the techniques for other domains, such as CV.

2 RELATED WORK

We briefly discuss ssl-gnn methods based on Siamese networks in terms of their data-augmentation strategies and their architecture choices. Then we give an overview of GNNs.

Data Augmentation. Although there are well-established data augmentation techniques in the CV domain, this is not the case for the graph domain [18, 34]. Different heuristics, based on high-order
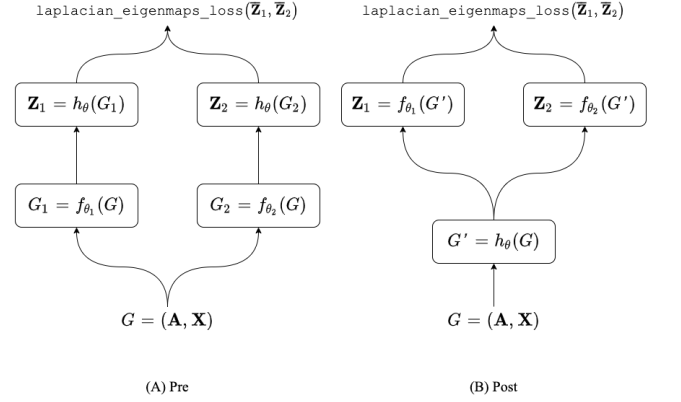
Figure 1: The architecture of GraphSurgeon, (A) first generates two augmented views of an input signal from two learned functions f_{θ_1} and f_{θ_2} and then passes them to a shared GNN encoder h_θ. Meanwhile, (B) first encodes the input signal and passes it to the two learned augmenters, f_{θ_1} and f_{θ_2}.

networks, perturbation of topology and attributes have been proposed [3, 18, 20, 30, 35]. It is not clear what the relative benefit of these augmentation strategies is, and little is known regarding the relevance of each strategy with respect to different downstream tasks. There are also conflicting observations regarding the composition of more than two augmentation strategies [18, 34]. A recent study [34] proposes a different strategy that dynamically chooses augmentation techniques while learning the graph representation through a joint optimization objective.

Our study differs from this line of research, first as there are no predefined data augmentations whatsoever, and second, the augmentation is jointly learned (*not chosen*) with the graph representation.

Architectures. The key difference between existing architectures arises from the need to prevent trivial solutions. To this end, existing studies [18, 27, 30, 31, 34, 35, 39, 40] often rely on contrastive architectures. However, as sampling truly contrastive terms is a difficult task, other studies have used asymmetric architectures to prevent trivial solutions. Originally proposed for CV [12, 15], asymmetric methods [8, 20, 28] have empirically shown that having different student and teacher networks and a stop gradient operation is sufficient to prevent collapse. Though there are three possibilities for asymmetry, not all of them are necessary to prevent trivial solutions [8, 12]

In a similar line of research, a recent study [3, 36] has introduced the notion of redundancy reduction to prevent collapse without requiring the asymmetry trick. Our study advocates a similar notion as [1, 36], in the sense that it has no explicit contrastive terms and does not use the asymmetry trick. On the other hand, we maximize agreement between positive pairs (as opposed to redundancy reduction), and to avoid collapse we capitalize on a constrained optimization objective motivated by Laplacian Eigenmaps [2].

Graph Neural Network (GNN). Because there is a plethora of GNN architectures, our discussion focuses on a brief description of the essentials of GNNs. Unlike previous neural network architectures, such as RNNs and CNNs, GNNs are highly flexible in terms of the type of data structure they can accommodate. The GNN formulation allows us to learn representations of objects for a multitude of data structure, such as grids, graphs, groups, geodesics and gauges [6, 16]. Contrary to the rigid computational graphs of RNN and CNN, in a GNN it is the input data that determines the computation graph. A GNN can be viewed as a message passing framework [14], where the messages are vectors that are updated by a neural network head [16], which is commonly shared by all nodes. The propagation rule for the "vanilla" GNN (GCN) [21] is defined as

$$X^{(l+1)} = \sigma(\tilde{A} X^{(l)} W^{(l)}) \tag{1}$$

where σ is an activation function, e.g., ReLU, $\tilde{A}$ is a symmetrically normalized adjacency matrix, X is node feature matrix ($X^{(0)} = X$), $X^{(l)}$ is the node feature at the lth layer, and $W^{(l)}$ is the weight of the lth layer.

Because of the limitations of the above GNN model, such as scalability and oversmoothing, several extensions have been proposed [4, 9, 10, 13, 14, 17, 23, 32, 33, 37, 38]. Of particular interest to this study are architectures for large scale graphs. The popular ones are neighborhood sampling [10, 17] and subgraph sampling [13, 37]. Instead of a full-batch training, which is used in the "vanilla" architecture, a neighborhood (layer) sampling uses a sampled set of neighbors for message passing. On the other hand, subgraph sampling techniques in a nutshell build sub-graphs as batches and apply a full-batch GNN over subgraphs (using clustering [13] or subgraph sampling [37]).

3 GRAPHSURGEON

We consider an undirected graph G with a set of nodes V and edges E, where $N = |V|$ and $M = |E|$. The adjacency matrix representation of G is given by $A \in [0,1]^{N \times N}$; and $\tilde{A} = D^{1/2}(A + I_N)D^{1/2}$ denotes the symmetrically normalized adjacency matrix. Where $D_{i,i} = \sum_j A_{i,j} + 1$ is the degree matrix and I_N is and Identity matrix of N diagonal entries. Each node u is associated with an attribute signal $x_u \in \mathbb{R}^F$, and the set of signals for all the nodes in the graph is given by $X \in \mathbb{R}^{N \times F}$.

Though the proposed method is agnostic to the type of GNN architecture, we assume a kind of GNN, parameterized by $\theta = \{W^{(l)} \in \mathbb{R}^{F_{l-1} \times F_l} : l = 1, \ldots, L\}$ is given. For example, the GCN model by Kipf and Welling [21] with the propagation rule in Eq. 1. Hence, we consider a generic L–layer message passing GNN, h_θ, parameterized by θ. Similar to Eq. 1, at every layer l in h_θ, each node propagates messages to its neighbors using a particular materialization of G, e.g., $\tilde{A}$. To add expressivity, the propagation is followed by a linear transformation using $W^{(l)}$ and then a non-linearity using ReLU.

Given a GNN h_θ, a certain materialization of G, and a node feature matrix X, we propose, GRAPHSURGEON, a novel ssl-gnn model based on the Siamese network [5]. The overview of the model is shown in Fig. 1. For ease of discussion, we consider A as the materialization of G. We express the GNN encoder using two equivalent formulations, $h_\theta(A, X)$ and $h_\theta(G)$, interchangeably.

3.1 Learning Data Augmentation

The core component behind SSL models based on a Siamese network is data augmentation, without it, such models have poor empirical performance [39]. However, existing augmentation strategies are often based on heuristics and/or trial and error [34]. That is, given a graph G, two augmented views $G_1 = t_1(G)$ and $G_2 = t_2(G)$ are generated by applying augmentation techniques $t_1 \sim \mathcal{T}$ and $t_2 \sim \mathcal{T}$, sampled from a set of predefined techniques, $\mathcal{T}$.

Automatically learning augmentations has not been well explored yet. In this study, we propose a simple yet novel data augmentation technique that is learned based on the signal encoded in the graph. Therefore, we replace t_1 and t_2, with trainable functions f_{θ_1} and f_{θ_2} parametrized by θ_1 and θ_2.

One can model f to learn either a different view of the topology A, or attribute signal X. In this study, we address the latter, and the former will be covered in future work. Therefore, we model f as an MLP, *i.e.*, two augmented views are generated as $X_1 = f_{\theta_1}(X)$, $X_2 = f_{\theta_2}(X)$, where $X_1 \in \mathbb{R}^{N \times D}$ and $X_2 \in \mathbb{R}^{N \times D}$ and $\theta_1 = \{W_1^{(l)} : l = 1, \ldots L_1\}$ and $\theta_2 = \{W_2^{(l)} : l = 1, \ldots L_2\}$, where L_1 and L_2 are the number of layers of f_{θ_1} and f_{θ_2}. In order for the graph signal to govern the learned augmentations X_1 and X_2, the key design choice in GRAPHSURGEON is to jointly learn θ_1 and θ_2 with θ.

A standard ssl-gnn technique follows a *pre-augmentation* architecture (in short, *pre*), i.e. first the input is augmented and then fed to a GNN encoder, Fig 1 (A). Our design, however, gives us the flexibility to change the order of augmentation to *post-augmentation* (in short, *post*), such that we first encode and then augment, Fig. 1 (B). The *post* architecture is motivated by efficiency in terms of run time and memory footprint.

While both architectures have the same number of parameters, *post* requires less computation time and memory during training. This is the case since with *pre*, the GNN encoder processes two separate representations for each node, leading to twice as many computations for the encoder. In *post*, the encoder only processes one representation for each node and the augmentation happens afterwards. With the GNN encoder making up the largest part of the architecture, this means that *pre* needs almost two times more resources than *post*.

3.2 Joint Training of the Parameters

The following discussion assumes the pre-augmentation architecture. The high-level overview for the training function of GRAPH-SURGEON is given in Listing 1. GRAPHSURGEON first generates augmented views $G_1 = (A, X_1) = f_{\theta_1}(G)$ and $G_2 = (A, X_2) = f_{\theta_2}(G)$, f_1(data.x) and f_2(data.x) in the pseudocode. Then it encodes the views using a shared GNN, h_θ, to generate the corresponding latent representations $Z_1 = h_\theta(G_1)$ and $Z_2 = h_\theta(G_2)$ (h(data.adj, x1) and h(data.adj, x2) in the listing).

Our goal in an ssl-gnn framework is to maximize the agreement between these two representations. To this end, we closely follow Laplacian Eigenmaps [2] and minimize the mean squared error between the normalized representations (unit vectors) of two data points. Though in Laplacian Eigenmaps the two data points correspond to different objects (e.g., two different nodes, images), in our case, these are just the unit embedding vectors of the two augmented views, which are $\bar{Z}_1$ and $\bar{Z}_2$ (z1_unit and z2_unit in

```python
# h: GNN Encoder
# f_1: First augmenter
# f_2: Second augmenter
# gamma: weight for the regularization
# B: Number of nodes in a batch
# D: Number of features after augmentation
# F_l: The final size of the embedding dimension

for data in loader:
    # data contains the graph data, that is, adj: a sparse adjacency
    # matrix and x: the node features. It could be a full-batch data
    # or a block-diagonal of b batches of subgraphs with the
    # corresponding node features.

    if pre_augment:
        # data.x.shape = BxF
        # data.adj.shape = BxB
        x1 = f_1(data.x) # x1.shape = BxD
        x2 = f_2(data.x) # x2.shape = BxD
        z1 = h(data.adj, x1) # z1.shape = BxF_l
        z2 = h(data.adj, x2) # z2.shape = BxF_l
    else:
        z = h(data.adj, data.x) # z.shape = BxD
        z1 = f_1(z) # z1.shape = BxF_l
        z2 = f_2(z) # z2.shape = BxF_l

    # Transforming embedding vectors to unit vectors
    z1_unit = torch.nn.functional.normalize(z1, dim=1, p=2)
    z2_unit = torch.nn.functional.normalize(z2, dim=1, p=2)

    mse = mse_loss(z1_unit, z2_unit) # Equation 2
    if use_improved_loss:
        I = torch.eye(F_l)
        constraint = (
            (z1_unit.t().matmul(z1_unit) - I).norm(p='fro') +
            (z2_unit.t().matmul(z2_unit) - I).norm(p='fro')
        ) # Column orthonormality constraint
    else:
        I = torch.eye(B)
        constraint = (
            (z1_unit.matmul(z1_unit.t()) - I).norm(p='fro') +
            (z2_unit.matmul(z2_unit.t()) - I).norm(p='fro')
        ) # Row orthonormality constraint

    optimizer.zero_grad()
    loss = mse + gamma * constraint # Equation 4
    loss.backward()
    optimizer.step()
```

Listing 1: A PyTorch and PyTorch Geometric based pseudo code for training GraphSurgeon

the pseudocode). Therefore, we define the objective based on these unit vectors as:

$$\mathcal{L}_\theta = ||\bar{Z}_1 - \bar{Z}_2||_F^2 \qquad (2)$$

However, Eq. 2 admits a trivial solution, that is, collapse into a single point or a subspace [2]. For this reason, we modify E.q. 2 and incorporate an orthonormality constraint inspired by the Laplacian Eigenmaps. Moreover, we want to jointly optimize the parameters of the augmenters. To achieve these goals we update E.q. 2 and

formulate it as a constrained optimization objective as in Eq. 3.

$$\mathcal{L}_{\theta,\theta_1,\theta_2} = ||\bar{Z}_1 - \bar{Z}_2||_F^2 \qquad (3)$$
$$s.t. \quad \bar{Z}_1\bar{Z}_1^T = I_N \quad and \quad \bar{Z}_2\bar{Z}_2^T = I_N$$

Eq. 3 encourages positive pairs across $\bar{Z}_1$ and $\bar{Z}_2$ to be similar to each other, and the orthonormality constraint ensures that each row in $\bar{Z}_1$ or $\bar{Z}_2$ is similar to itself and orthonormal to other rows. Consequently, a trivial solution is avoided [2]. As stated earlier, existing methods often rely on contrastive terms or the asymmetry trick to achieve this.

By using the Lagrangian, we can relax the constrained optimization problem using Eq. 4, to obtain a regularized objective which we call Laplacian Eigenmaps loss

$$\mathcal{L}_{\theta,\theta_1,\theta_2} = ||\bar{Z}_1 - \bar{Z}_2||_F^2 + \gamma\big(||\bar{Z}_1\bar{Z}_1^T - I_N||_F + ||\bar{Z}_2\bar{Z}_2^T - I_N||_F\big) \quad (4)$$

Improved objective. Although E.q. 4 addresses the aforementioned concerns, the matrix multiplications, $\bar{Z}_1\bar{Z}_1^T$ and $\bar{Z}_2\bar{Z}_2^T$, produce $N \times N$ dense matrices. This is not desirable for full-batch GNNs, since storing the resulting matrix in a GPU memory is not feasible. For this reason, we improve the regularization by making a simple change and replacing the row orthonormality regularization with column *orthogonality* regularization. That is, replacing $\bar{Z}_1\bar{Z}_1^T - I_N$ with $\bar{Z}_1^T\bar{Z}_1 - I_{F_L}$ and $\bar{Z}_2\bar{Z}_2^T - I_N$ with $\bar{Z}_1^T\bar{Z}_1 - I_{F_L}$. Since $F_L \ll N$, is usually in the orders of hundreds, we can easily store an $F_L \times F_L$ matrix in the GPU's memory.

Finally, when training GraphSurgeon, gradients are back propagated on both the left and right networks. As a result, all the parameters are updated according to the loss incurred with respect to the signal from the graph. This in turn, allows the parameters of both the encoder, h_θ, and the augmenters, f_{θ_1} and f_{θ_2}, to be governed by the graph signal. The fact that we have jointly trainable augmenters gives us the flexibility to use Fig. 1 (A) and (B) without compromising the qualitative performance. This is difficult, if not impossible, when using predefined augmentation techniques.

3.3 Scalability

The flexibility of GraphSurgeon's architecture allows us to seamlessly integrate it into virtually any kind of GNN architecture. For large-scale graphs with hundreds of millions of edges, one can integrate GraphSurgeon with existing methods for large-scale graphs. For example, with methods based on neighborhood (layer) sampling [17] and subgraph sampling [13, 37]. For small graphs, we use full-batch GCN [21], while for large-scale graphs we use neighborhood sampling unless stated otherwise.

4 EXPERIMENTS

We validate the practical use of GraphSurgeon using 14 publicly available datasets, ranging from small to large-scale graphs. All of the datasets are collected from PyTorch Geometric (PyG) [1], and grouped as

- Citation Networks (*Cora*, *DBLP*, and *PubMed*): Paper to paper citation networks, and we classify papers into different subjects [17].

---

[1] https://pytorch-geometric.readthedocs.io/en/latest/index.html

Dataset	N	M	F	C	Task
Cora Full	19,793	126,842	8,710	70	*MCC*
DBLP	17,716	105,734	1,639	4	*MCC*
PubMed	19,717	88,648	500	3	*MCC*
Physics	34,493	495,924	8,415	5	*MCC*
CS	18,333	163,788	6,805	15	*MCC*
Computers	13,752	491,722	467	10	*MCC*
Photo	7,650	238,162	745	8	*MCC*
Facebook	22,470	342,004	128	4	*MCC*
Flickr	89,250	899,756	500	7	*MCC*
GitHub	37,700	578,006	128	2	*BC*
WikiCS	11,701	297,110	300	10	*MCC*
Actor	7,600	30,019	932	5	*MCC*
Yelp	716,847	13,954,819	300	100	*MLC*
Reddit	232,965	114,615,892	602	41	*MCC*

Table 1: Summary of the datasets, and $N = |V|$, $M = |E|$, F is the number of features, and C is the number of classes. *BC*, *MCC* and *MLC* represent binary, multi-class and multi-label classification, respectively

- Co-Author Networks (Computer Science (*CS*) an *Physics*): Author collaboration network from Microsoft Academic Graph, and the task is to predict the active field of authors [26].
- Co-Purchased Products Network (*Computers* and *Photo*): Co-purchased products from the respective categories on Amazon, and the task is to predict the refined categories [26].
- Wikipedia (*Actor* and *WikiCS*): WikiCS contains Wikipedia hyperlinks between Computer Science articles, and we classify articles into branches of CS [26], and Actor contains actors co-occurrence on the same Wikipedia article and we classify actors into groups based word of actors' Wikipedia [24].
- Social (*Facebook*, *Flickr*, *GitHub*, *Reddit*, and *Yelp*): Facebook contains a page to page graph of verified Facebook sites, and we want to classify pages into their categories [25]. Flickr contains a network of images based on common properties (e.g., geo-location) along with their description, and the task is to predict a unique tag of an image [37]. GitHub contains the social network of developers, and we want to classify developers as web or machine learning developers [25]. Yelp is also the social network of Yelp users, and we predict business categories each user has reviewed. For Reddit, we predict the subreddits (communities) of user posts [17, 37].

A brief summary of the datasets is provided in Table 1. We also group them into two, as large (Yelp and Reddit) and small (the rest).

We compare GraphSurgeon against 11 state-of-the-art baselines grouped into two

- *Semi-Supervised*: Six of the baselines are methods that use a fraction of the node labels during training, three of which (GCN [21], GAT [29], GraphSage [17]) are used for small and medium-size graphs and the rest (ClusterGCN [13], GraphSaint [37], and PPRGO [4]) for large-scale graphs.
- *Self-supervised*: There are five methods under this group, three of which (DGI [30], MVGRL [18], and GCA [40]) use a contrastive architecture to prevent a trivial solution and

the other two SelfGNN [20] and BGRL [28] use asymmetry. Because these two techniques extend the same method, BYOL [15], for visual representation to graph representation and use the same code base, we present them as one.

For GCN, GAT, GraphSage, ClusterGCN, GraphSaint, and DGI we use the implementation from PyG. For the rest we use the official implementation provided by the authors.

4.1 Experimental Protocol

For all the datasets we have three splits, training, validation and test. For some of them, we use the splits provided by PyTorch Geometric, and for the rest, we randomly split them into 5% training, 15% validation and 80% test sets. We tune the hyperparameters of all the algorithms using Bayesian optimization [2], however for a fair comparison, we fix the representation dimension to 128. In addition, we run all the models for 500 epochs and take the epoch with the best validation score.

We train the semi-supervised methods using the training split and tune their hyperparaters using the validation set. Finally, we use the test set to infer the labels and report their performance. For the self-supervised methods, we train them without any label and tune them using the validation set. Following standard practice, they are evaluated under the linear protocol. This means that we freeze the models and add a logistic regression (linear) classifier on top. The linear classifier is trained using only the training split for 100 and 500 epochs, for small and large datasets, respectively. Similar to the semi-supervised setting, we use the test set to simply predict the labels and report prediction quality.

We have three types of node classification tasks, which are binary, multi-class, and multi-label classifications. Similar to existing studies, for the binary and multi-class tasks, we use accuracy, and for the multi-label, the Area Under the Receiver Operating Characteristic Curve (ROC-AUC).

Unless a different setting is stated, we assume the aforementioned protocol.

4.2 Results

The node classification results are reported in Tables 2 and 3. Overall, GraphSurgeon is comparable to the semi-supervised baselines and in-par (sometimes marginally better and at times marginally lower than) the self-supervised baselines.

As expected, the semi-supervised models are consistently better than the self-supervised ones, except for Actor. MVGRL gives the best result, with more than 90% improvement over the best performing method. This comes as a result of using higher-order augmentation that happens to be beneficial for the Actor dataset. A similar performance is not observed for MVGRL on the other datasets. This provides a motivation for automatically learning high-order topology signals that benefit some datasets. As stated earlier, this will be covered in future work.

However, our finding showcases that just using the learned attribute augmentations and without requiring explicit negative samples, one can achieve a performance consistently close to semi-supervised models across a number of datasets and classification tasks. On average, our model is at most 2 percentage points away

Datasets	Algorithms							
	Semi-Supervised			Contrastive			Asymmetric	GRAPHSURGEON
	GCN	GAT	GRAPHSAGE	DGI	MVGRL	GCA	SELFGNN/ BGRL	
Cora	**60.1±.001**	58.27±.003	57.45±.003	50.66±.001	39.42±.193	37.64±.014	54.61±.135	**56.33±.07**
DBLP	82.7±.002	**82.88±.002**	81.39±.005	78.87±.002	69.2±.052	81.16±.007	81.32±.071	**81.48±.09**
PubMed	**85.62±.001**	84.98±.002	84.73±.001	84.28±.001	77.99±.315	82.76±.005	84.6±.076	**84.94±.091**
Physics	**95.4±.001**	95.02±.002	†	94.92±.001	91.18±.024	†	**95.11±.07**	**95.11±0.025**
CS	**91.87±.001**	91.07±.002	91.44±.001	91.72±.001	87.18±.095	88.01±.005	**92.23±.01**	92.03±.0
Computers	**88.54±.003**	88.3±.006	87.93±.004	80.28±.004	78.57±.14	74.04±.005	**86.23±.139**	85.16±.133
Photo	93.02±.003	93.18±002	**93.64±.002**	92.36±.06	86.04±.12	84.93±.009	**92.87±.08**	92.27±.05
Actor	28.38±.008	28.62±.01	**33.88±.007**	29.93±.007	**63.3±.03**	27.39±.01	29.41±1.46	30.19±.34
WikiCS	76.87±.006	77.38±.005	**77.41±.006**	70.01±.007	61.7±.52	75.25±.006	75.34±.528	**75.59±.11**
Facebook	**89.5±.002**	89.3±.01	89.25±.002	82.42±.001	78.88±0.045	86.29±0.004	**86.38±.084**	84.92±0.015
Flickr	51.66±.001	42.35±.001	**52.11±.001**	45.94±.001	†	†	**51.26±.528**	50.91±.054
Github	**86.14±.001**	86.16±.001	85.77±.001	83.84±.001	83.93±0.032	†	85.58±.053	**85.7±.028**

Table 2: The classification accuracy results along with the standard deviation. The bold highlight indicates the best performing algorithms from both the semi-supervised and self-supervised methods. † indicates that the algorithm has crashed because of an out-of-memory error.

Algorithms	Datasets	
	Yelp (ROC-AUC)	Reddit (Accuracy)
CLUSTERGCN (semi)	78.21	95.33
GRAPHSAINT (semi)	75.62	95.73
PPRGO (semi)	77.7	91.8
GRAPHSURGEON	77.44	91.22

Table 3: The prediction quality for the large scale datasets. For this experiment, we use semi-supervised and scalable GNN architectures as the full-batch ones do not fit in GPU memory. In addition, all the SSL-GNN baselines throw an out of memory error.

Figure 2: Comparison of the original and improved loss function in terms of accuracy, memory usage and run time (time to finish an epoch).
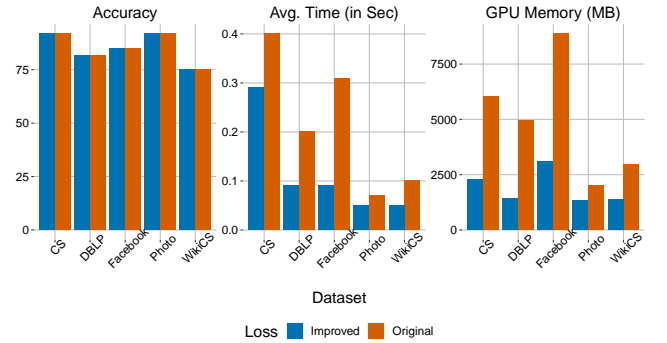
from the best performing semi-supervised method. Moreover, it scales to large networks with hundreds of millions of edges, where the other SSL-GNN methods failed to handle.

4.3 Ablation studies

In the following, we investigate the impact of different aspects of GraphSurgeon.

4.3.1 Loss Function. We have seen the constrained optimization objective in Section 3.2 and also shown a way to improve it. In the following, we analyze the effect of using the original vs. the improved loss function with respect to prediction accuracy and resource usage. For the qualitative experiment, we train both flavors for 100 epochs and just 1 epoch otherwise. The results of this experiment are reported in Fig 2. As expected, both flavors achieve similar qualitative performance. Nonetheless, the improved version is significantly better than the original one in terms of memory usage and run time.

4.3.2 Batch Size. As contrastive signals are indirectly injected because of the orthogonality constraint of Eq. 4, it is important to analyze the impact of batch size to see if a large batch size is needed to effectively avoid trivial solutions. For this reason, we train the model using sampled neighborhood subgraphs [17] instead of full-batch, and both the model and the linear head are trained for 100 epochs. The results are reported in Fig. 3, and in general performance is directly proportional to batch size until a certain point. For small datasets, there is improvement up to 1024. The largest improvements are for the GitHub and Physics datasets, which are 7.57 (from 75.38% to 82.95%) and 7.66 perecentage points of accuracy (from 74.35% to 82.01%), respectively. However, that is not the case for the larger ones (Reddit and Yelp), where both smaller and larger batch sizes give comparable performance. Overall, we have not observed qualitative differences for batch sizes bigger than 1024. In our experiments, batch size greater than 1024 is only related to faster training, not improved quality.
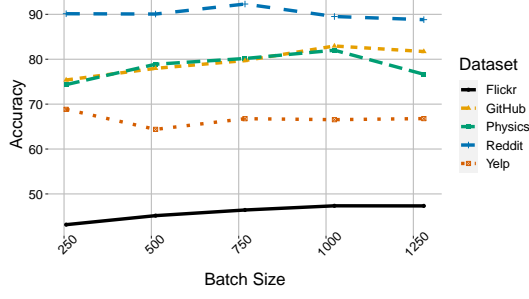
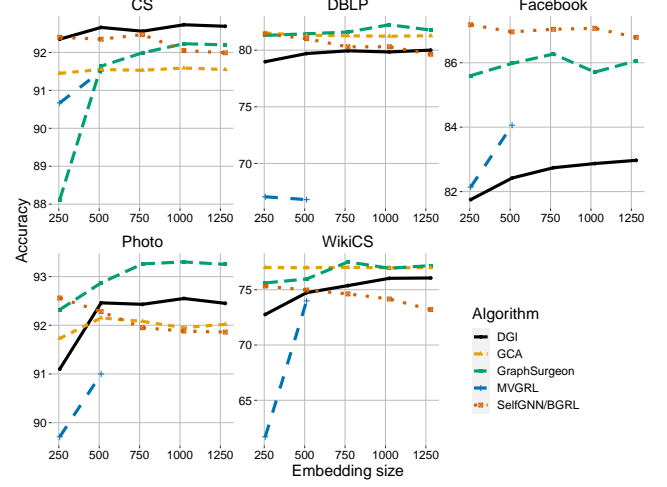**Figure 3: Effect of batch size on GraphSurgeon's performance**

Figure 4: Effect of embedding size on GraphSurgeon and the ssl-gnn baselines.

4.3.3 Embedding Size. To provide a perspective, for this analysis we include the baselines. We use the same setting as the first experiment (Tables 2 and 3), and we examine embedding sizes in [256, 512, 768, 1024, 1280]. GraphSurgeon and DGI have the tendency to improve as we increase the embedding size. On the other hand, GCA and SelfGNN/BGRL, stay the same or decrease. For MVGRL, it seems that it has the tendency to improve proportional to the embedding size. However, we were able to observe only for 256 and 512, as it throws an out of memory error for larger values.

4.3.4 Pre vs. Post Augmentation. In terms of performance, both augmentation techniques give equivalent results. The only difference is that the hyperparameters need to be tuned separately for each one. In Fig. 5 we show the final configuration of the hyperparameters that maximize prediction accuracy on the validation set, which are obtained using Bayesian optimization. As can be seen from the figure, they converge to different values.

Since the main motivation for introducing the post-augmentation is efficiency, in Fig. 6 we show the memory usage and run time (to finish an epoch) required by these variants. As anticipated, the post-augmentation is significantly faster and has a lot less memory overhead.

4.3.5 Symmetry vs. Asymmetry. GraphSurgeon's architecture can easily be replaced by an asymmetric one. For this reason, we create asymmetry just by adding a prediction head on the left-network and inserting batch norm in the GNN encoder [20, 28, 36]. We update the parameters of both the left and right networks using stochastic gradient descent. As shown in Fig. 7, the performance of the asymmetric architecture is slightly lower than the symmetric one.

4.3.6 Convergence. Recent studies [36, 40] have shown that ssl-gnn methods usually require a large number of epochs (several thousand) to achieve a performance comparable to semi-supervised methods. In Fig. 8 we show GraphSurgeon's convergence, and usually, 50 epochs are sufficient to achieve comparable performance to semi-supervised methods.
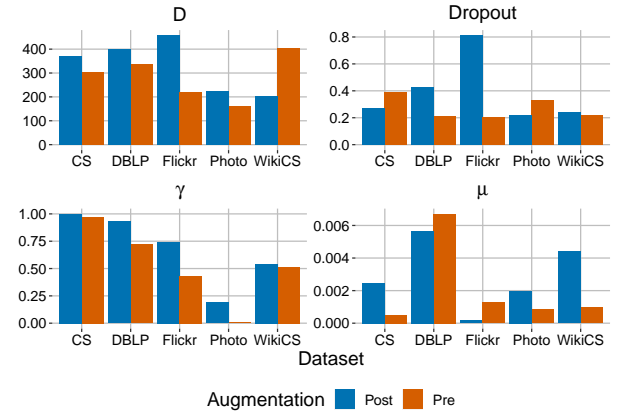
Figure 5: Tuned values of the hyperparameters of Graph-Surgeon for the pre and post augmentation techniques. D is the number of features after augmentation and the GNN encoder for pre and post augmentations, respectively. γ the weight of the constraint in Eq. 4, and μ is the learning rate.

4.4 Implementation Details

GraphSurgeon is implemented using PyTorch and PyTorch Geometric libraries [3].

For each augmentation head, we use a simple one-layer linear head. To avoid overfitting, we use dropout in both heads.

For the GNN encoder we use two types of architectures, which are GCN [21] and GraphSage [17]. A full-batch GCN is used for the small datasets, and a mini-batch GNN based on GraphSage with neighborhood sampling [17] is used for the larger ones. As stated earlier, one can substitute these with any other architecture as

---

[3]To respect the double blind review policy of the conference we shall release the source code after the double blind period has ended
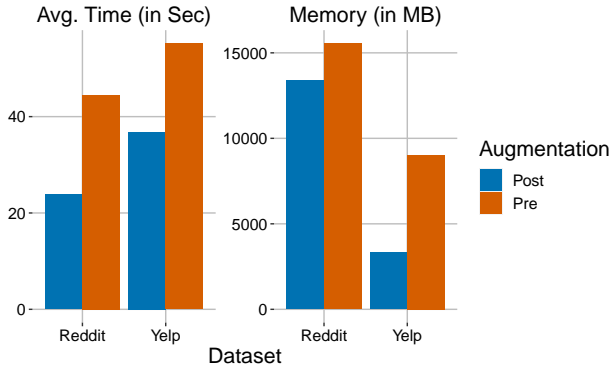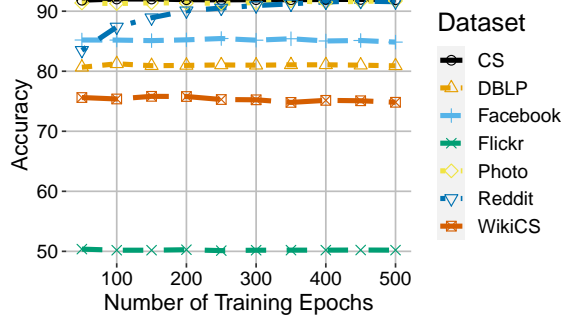
Figure 6: Analysis of Pre and Post Augmentation techniques in terms of memory usage and run time complexity (time to finish an epoch)
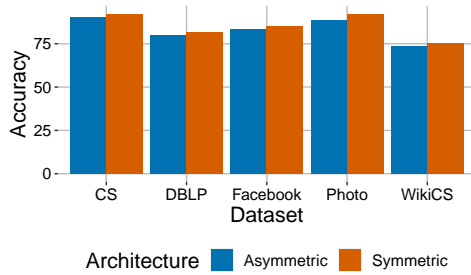
Figure 7: Comparison between GraphSurgeon's architecture (Symmetric) and an Asymmetric architecture

Figure 8: Analysis of the convergence of GraphSurgeon

necessary. A dropout is also added, and finally, we use a residual connection for the GNN encoder.

As indicated using the shape of the matrices in Listing 1, we use the output of the GNN encoder as the embedding of nodes for the pre-augmentation architecture and the output of the augmentation head for the post-augmentation case.

Although existing ssl-gnn methods [20, 28, 28, 36] require different normalization strategies, such as Batch Norm and Layer

Norm that is not necessary for GraphSurgeon; as a result, no such normalization is used.

5 DISCUSSION AND CONCLUSION

In this paper, we propose a self-supervised graph representation learning method called GraphSurgeon based on the Siamese network. Unlike prior methods that rely on heuristics for data augmentation, our method jointly learns the data augmentation with the representation (embedding) guided by the signal encoded in the graph. By capitalizing on the flexibility of the learnable augmentations, we propose an alternative new strategy for augmentation, called post-augmentation, which happens after an encoding. This is in contrast to the standard pre-augmentation strategy that happens before the encoding. We also show that the alternative strategy significantly improves the scalability and efficiency of our model.

Furthermore, the method does not require explicit contrastive terms or negative sampling. However, contrary to related studies with no contrastive terms, we employ a scalable principled constrained optimization inspired by Laplacian Eigenmaps, as opposed to engineering tricks to prevent trivial solutions.

We perform an extensive empirical evaluation of the proposed method using 14 publicly available datasets on three types of node classification tasks. Besides, we compare the method with strong SOTA baselines, six semi-supervised GNNs, and five self-supervised GNNs. Our finding shows that GraphSurgeon is comparable to the semi-supervised GNNs and on-par with the self-supervised ones.

In this study, we focus on learning augmentations for attribute signals. However, one can also consider topological augmentations and we shall explore this in future work. In addition, it is also important to learn to identify if a particular type (topological/attribute) or a combination thereof is relevant to a down stream task. Finally, this study does not inspect and interpret the learned augmentations, and further study is necessary to shed more insight regarding what is learned.

REFERENCES

[1] Adrien Bardes, Jean Ponce, and Yann LeCun. 2021. VICReg: Variance-Invariance-Covariance Regularization for Self-Supervised Learning. arXiv:2105.04906 [cs.CV]

[2] Mikhail Belkin and Partha Niyogi. 2003. Laplacian Eigenmaps for Dimensionality Reduction and Data Representation. *Neural Computation* 15, 6 (2003), 1373–1396. https://doi.org/10.1162/089976603321780317

[3] Piotr Bielak, Tomasz Kajdanowicz, and Nitesh V. Chawla. 2021. Graph Barlow Twins: A self-supervised representation learning framework for graphs. arXiv:2106.02466 [cs.LG]

[4] Aleksandar Bojchevski, Johannes Klicpera, Bryan Perozzi, Amol Kapoor, Martin Blais, Benedek Rózemberczki, Michal Lukasik, and Stephan Günnemann. 2020. Scaling Graph Neural Networks with Approximate PageRank. *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Jul 2020). https://doi.org/10.1145/3394486.3403296

[5] 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.

[6] Michael M. Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. 2021. Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges. arXiv:2104.13478 [cs.LG]

[7] Mathilde Caron, Ishan Misra, Julien Mairal, Priya Goyal, Piotr Bojanowski, and Armand Joulin. 2020. Unsupervised Learning of Visual Features by Contrasting Cluster Assignments. *CoRR* abs/2006.09882 (2020). arXiv:2006.09882 https://arxiv.org/abs/2006.09882

[8] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. 2021. Emerging Properties in Self-Supervised Vision Transformers. *CoRR* abs/2104.14294 (2021). arXiv:2104.14294 https://arxiv.org/abs/2104.14294

[9] Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, and Xu Sun. 2019. Measuring and Relieving the Over-smoothing Problem for Graph Neural Networks from the Topological View. arXiv:1909.03211 [cs.LG]

[10] Jie Chen, Tengfei Ma, and Cao Xiao. 2018. FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling. arXiv:1801.10247 [cs.LG]

[11] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey E. Hinton. 2020. A Simple Framework for Contrastive Learning of Visual Representations. *CoRR* abs/2002.05709 (2020). arXiv:2002.05709 https://arxiv.org/abs/2002.05709

[12] Xinlei Chen and Kaiming He. 2020. Exploring Simple Siamese Representation Learning. *CoRR* abs/2011.10566 (2020). arXiv:2011.10566 https://arxiv.org/abs/2011.10566

[13] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. 2019. Cluster-GCN. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Jul 2019). https://doi.org/10.1145/3292500.3330925

[14] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. 2017. Neural Message Passing for Quantum Chemistry. arXiv:1704.01212 [cs.LG]

[15] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre H. Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Ávila 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. *CoRR* abs/2006.07733 (2020). arXiv:2006.07733 https://arxiv.org/abs/2006.07733

[16] William L. Hamilton. 2020. *Graph Representation Learning*.

[17] William L. Hamilton, Rex Ying, and Jure Leskovec. 2018. Inductive Representation Learning on Large Graphs. arXiv:1706.02216 [cs.SI]

[18] Kaveh Hassani and Amir Hosein Khas Ahmadi. 2020. Contrastive Multi-View Representation Learning on Graphs. *CoRR* abs/2006.05582 (2020). arXiv:2006.05582 https://arxiv.org/abs/2006.05582

[19] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross B. Girshick. 2019. Momentum Contrast for Unsupervised Visual Representation Learning. *CoRR* abs/1911.05722 (2019). arXiv:1911.05722 http://arxiv.org/abs/1911.05722

[20] Zekarias T. Kefato and Sarunas Girdzijauskas. 2021. Self-supervised Graph Neural Networks without explicit negative sampling. *CoRR* abs/2103.14958 (2021). arXiv:2103.14958 https://arxiv.org/abs/2103.14958

[21] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. arXiv:1609.02907 [cs.LG]

[22] Ishan Misra and Laurens van der Maaten. 2019. Self-Supervised Learning of Pretext-Invariant Representations. *CoRR* abs/1912.01991 (2019). arXiv:1912.01991 http://arxiv.org/abs/1912.01991

[23] Kenta Oono and Taiji Suzuki. 2021. Graph Neural Networks Exponentially Lose Expressive Power for Node Classification. arXiv:1905.10947 [cs.LG]

[24] Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. 2020. Geom-GCN: Geometric Graph Convolutional Networks. arXiv:2002.05287 [cs.LG]

[25] Benedek Rozemberczki, Carl Allen, and Rik Sarkar. 2021. Multi-scale Attributed Node Embedding. arXiv:1909.13021 [cs.LG]

[26] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. 2019. Pitfalls of Graph Neural Network Evaluation. arXiv:1811.05868 [cs.LG]

[27] Mengying Sun, Jing Xing, Huijun Wang, Bin Chen, and Jiayu Zhou. 2021. MoCL: Contrastive Learning on Molecular Graphs with Multi-level Domain Knowledge. arXiv:2106.04509 [physics.bio-ph]

[28] Shantanu Thakoor, Corentin Tallec, Mohammad Gheshlaghi Azar, Rémi Munos, Petar Veličković, and Michal Valko. 2021. Bootstrapped Representation Learning on Graphs. arXiv:2102.06514 [cs.LG]

[29] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. arXiv:1710.10903 [stat.ML]

[30] Petar Veličković, William Fedus, William L. Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. 2018. Deep Graph Infomax. arXiv:1809.10341 [stat.ML]

[31] Xiao Wang, Nian Liu, Hui Han, and Chuan Shi. 2021. Self-supervised Heterogeneous Graph Neural Network with Co-contrastive Learning. arXiv:2105.09111 [cs.LG]

[32] Felix Wu, Tianyi Zhang, Amauri Holanda de Souza Jr. au2, Christopher Fifty, Tao Yu, and Kilian Q. Weinberger. 2019. Simplifying Graph Convolutional Networks. arXiv:1902.07153 [cs.LG]

[33] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken ichi Kawarabayashi, and Stefanie Jegelka. 2018. Representation Learning on Graphs with Jumping Knowledge Networks. arXiv:1806.03536 [cs.LG]

[34] Yuning You, Tianlong Chen, Yang Shen, and Zhangyang Wang. 2021. Graph Contrastive Learning Automated. *CoRR* abs/2106.07594 (2021). arXiv:2106.07594 https://arxiv.org/abs/2106.07594

[35] Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. 2020. Graph Contrastive Learning with Augmentations. *CoRR* abs/2010.13902 (2020). arXiv:2010.13902 https://arxiv.org/abs/2010.13902

[36] Jure Zbontar, Li Jing, Ishan Misra, Yann LeCun, and Stéphane Deny. 2021. Barlow Twins: Self-Supervised Learning via Redundancy Reduction. *CoRR* abs/2103.03230 (2021). arXiv:2103.03230 https://arxiv.org/abs/2103.03230

[37] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. 2020. GraphSAINT: Graph Sampling Based Inductive Learning Method. arXiv:1907.04931 [cs.LG]

[38] Kaixiong Zhou, Xiao Huang, Yuening Li, Daochen Zha, Rui Chen, and Xia Hu. 2020. Towards Deeper Graph Neural Networks with Differentiable Group Normalization. arXiv:2006.06972 [cs.LG]

[39] Yanqiao Zhu, Yichen Xu, Feng Yu, Qiang Liu, Shu Wu, and Liang Wang. 2020. Deep Graph Contrastive Representation Learning. arXiv:2006.04131 [cs.LG]

[40] Yanqiao Zhu, Yichen Xu, Feng Yu, Qiang Liu, Shu Wu, and Liang Wang. 2021. Graph Contrastive Learning with Adaptive Augmentation. In *Proceedings of the Web Conference 2021* (Ljubljana, Slovenia) (*WWW '21*). Association for Computing Machinery, New York, NY, USA, 2069–2080. https://doi.org/10.1145/3442381.3449802