Evaluating Graph Generative Models with Contrastively Learned Features

Anonymous Author(s)
Affiliation
Address
email

Abstract

A wide range of models have been proposed for Graph Generative Models, necessitating effective methods to evaluate their quality. So far, most techniques use either traditional metrics based on subgraph counting, or the representations of randomly initialized Graph Neural Networks (GNNs). We propose using representations from contrastively trained GNNs, rather than random GNNs, and show this gives more reliable evaluation metrics. Neither traditional approaches nor GNN-based approaches dominate the other, however: we give examples of graphs that each approach is unable to distinguish. We demonstrate that Graph Substructure Networks (GSNs), which in a way combine both approaches, are better at distinguishing the distances between graph datasets.

1 Introduction

Quantitative evaluation of generative models is challenging [46]; evaluating purely by visual inspection can introduce biases, particularly towards "precision" at cost of "recall" [38]. The traditional metric, likelihood, is also not only difficult to evaluate for implicit generative models on complex, highly structured datasets, but can also be a poor fit to users' goals with generative modeling [38, 46]. Many quantitative proxy measures for the discrepancy between generator and real distributions have thus been used in recent work on generative modeling of images, some of the most important including the Inception Score (IS) [39], Fréchet Inception Distance (FID) [21], Precision/Recall (PR) [38], and Density/Coverage [33]. FID and PR measures in particular have been shown to correlate with human judgments in some practical settings. All of these measures use representations extracted from CNNs pretrained on ImageNet classification [37]. Nevertheless, it is recently shown that self-supervised representations archives more reasonable ranking in terms of FID/Precision/Recall, while the ranking with ImageNet-pretrained embeddings often can be misleading [31].

These problems are exacerbated for the evaluation of generative models of graph data. Graphs are often used to represent concepts in a broad range of specialized domains, including various fields of engineering, bioinformatics, and so on, so that it is more difficult or perhaps impossible to find "generally good" features like those available for natural images via ImageNet models. Many methods for evaluating graph generative models are thus based on measuring discrepancies between statistics of generic low-level graph features, including local measurements like degree distributions, clustering coefficient distributions, and four-node orbit counts [25, 53], or simple global properties such as spectra [26]. Differences between distributions of these features are generally measured via the maximum mean discrepancy (MMD) [15] or total variation (TV) distance.

More recent work [47] proposes instead extracting features via randomly initialized Graph Isomorphism Networks (GINs) [52]. These features provide a representation for each graph, so that metrics like FID, Precision/Recall, and Density/Coverage can be estimated using these representations. For

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

datasets with natural labels, we can alternatively find representations based on training a classifier; such features, however, are optimized for a very different task.

One significant challenge of graph data is that it is computationally difficult to even tell whether two graphs are the same (are isomorphic to one another). GINs are known to be as powerful at detecting isomorphism as the standard Weisfeiler-Lehman (WL) test [52], but it is not known whether *random* GIN representations have this ability. Experimental evidence [52] shows that trained GIN networks are as powerful as the WL test, but have substantially worse power at random initialization.

Unlike for distributions of natural images, however, the relevant graph features and particularly the semantics of node or edge features varies widely between datasets. On molecular graphs, adding any single edge will almost always violate physical constraints about stable atomic bonds; for house layouts, the feasibility of adding an edge depends significantly on the rest of the layout and the type of rooms (few front doors open into a bathroom, and direct connections between rooms on the first and third floors are unlikely); Traditional metrics and random graph neural networks are both unlikely to be able to capture these complex inter-dependencies functioning very differently across domains, and it similarly seems quite unlikely that pretraining on some "graph ImageNet" would be able to find useful generic graph features.

We therefore propose to train graph encoders on the same data used to train the generative model, using self-supervised contrastive learning to find meaningful feature extractors [19, 20, 35, 42, 45, 49, 54–57], which we then use to compare the generated graphs to a test set. The set of perturbations introduced in contrastive learning teaches the model which kinds of graphs should be considered similar to one another. In this work we use types of perturbations traditionally used for training contrastive learning methods on graphs, but point out that future work focusing on domain-specific modeling can directly incorporate knowledge about which graphs should be considered similar by choosing different perturbation sets.

Inspired by the theoretical results for contrastive learning of [51], we also propose two variations to our representation learning procedures. This upper bound shows that contrastively learned representations work well for downstream tasks as long as the probability of data points yielding overlapping augmentations is relatively large for *within-class* dataset pairs and relatively small for *cross-class* pairs. Edit distances between graphs on typical training sets are large, however; we propose to use subgraphs ("crops") in our set of data augmentations for contrastive learning, which is more likely to yield overlaps. We also suggest enforcing a layer-wise Lipschitz constraint on feature extractors, which encourages similar graphs to have similar learned representations. We show experimentally that both changes improve learning.

With all of these improvements, we further ask: can we theoretically guarantee that our learned GNN representations outperform traditional local metrics? We prove that we cannot: we give examples of graphs easily distinguishable by local metrics that first-order GNNs cannot distinguish. Yet the converse is also true: we show graphs easily distinguishable by GNNs that appear equivalent to local metrics. We thus propose to use models based on Graph Substructure Networks (GSNs) [6], (using node degrees, and node clustering coefficients), and as a result, explicitly incorporating local metrics into our models and surpassing the power of the WL test, and yield further improvements.

2 Related Work

Graph generative models. Our work is not particular to any type of graph generative model, as it focuses on simply evaluating samples; nonetheless, it is worth briefly reviewing some methods. Sequential generation models, such as GraphRNN [53] and GRAN [26], generate nodes and edges in an auto-regressive manner. These models are efficient, but can struggle to capture global properties of the graphs, and require a predefined ordering. One-shot models such as MolGAN [10], GraphDeconv [13], GraphVAE [24], and GraphNVP [28], on the other hand, generate all nodes and edges in one shot; they can model long-range dependencies, but generally are not efficient and cannot scale to large graphs. For a detailed overview of graph generative models, see [17].

Evaluating graph generative models. Graph generative models can be even more challenging to evaluate than visual or textual models, because it is generally more difficult for humans to judge the quality of a generated graph. The classic measure of the quality of a generative model, likelihood, also has significant issues with graphs: in addition to the kinds of issues that appear in generative

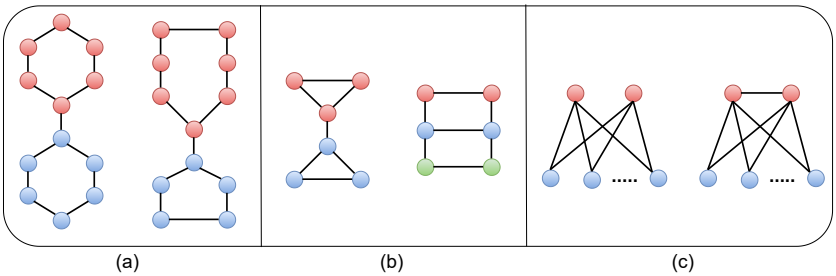

Figure 1: (a) An example of two graphs that can be differentiated by GNNs but not local metrics. Both graphs have same degree distribution, clustering coefficient and 4 node orbits. (b) An example of two graphs that can be differentiated by local metrics (clustering coefficient and smallest cycle) but not GNNs. (c) Adding a single edge can drastically change cluster coefficient and 4-node orbits.

models of images [46], the likelihood is particularly hard to evaluate on graphs where even checking equality is quite difficult [34]. The most common method is to compute the Maximum Mean Discrepancy (MMD) [15] between distributions of local graph statistics, including node degrees, cluster coefficients, and counts of orbits up to a particular size [26, 53]. Global statistics, such as eigenvalues of the normalized graph Laplacian, are also used [53]. These metrics, however, focus only on low-level structure of the graph, and ignore any features that might be present on nodes or edges [47]. Choosing an appropriate kernel is also very important to consistency of these metrics [34]. These types of graph statistics might also encourage models to overfit to the training data, rather than truly learning the target distribution [41].

For unlabeled image-based generative models, most work focuses on metrics including the Inception Score (IS) [39], Fréchet Inception Distance (FID) [21], Precision/Recall (PR) [38], and Density/Coverage [33], all of which compare distributions in a fixed latent space (typically activations of a late layer in an InceptionV3 [44] model trained on ImageNet). These methods are rarely adopted in graph generative models, due to challenges with "general-purpose" graph models discussed in Section 1. Thompson et al. [47] thus used random (untrained) graph encoders in these metrics. We discuss these methods in more detail in the appendix. Our work, inspired by Morozov et al.'s similar proposal in image domains [32], explores the use of self-supervised contrastive training to find representations that work better than random initializations.

Graph Contrastive Learning. A popular method for self-supervised learning, contrastive learning generally aims to find a representation roughly invariant to various operations (e.g. for images, taking random crops, horizontal flipping, shifting colors) but able to identify different source data points. Ideally, such a representation will be useful for downstream tasks not known when learning the representation. In graph settings, learned representations may be at the node, edge, or graph levels.

Early works adopting contrastive learning to encode graphs used DeepInfoMax [22] loss to enforce consistency between local (node) and global (graph) representations (e.g., DGI [49] and InfoGraph [42]). These methods did not use any specific augmentations and simply used distinct graphs as negative examples. Following works begin utilizing graph augmentations to further improve the learned representations. For example, MVGRL [19] uses graph diffusion and sub-graph inducing as two types of augmentations, where as GCC [35] only uses multi-hop ego network inducing. More recent works use four types of graph augmentations including: feature masking, node dropping, edge perturbation, and sub-graph inducing [20, 45, 54–57]. Among them, a few works rely on trial-and-error or heuristics to choose those augmentations for a given dataset (e.g., GraphCL [55], GRACE [56], and GCA [57]), whereas others introduce a policy network to learn to sample augmentations and compute their parameters end-to-end along with graph representations (e.g., JOAO [54] and LG2AR [20]). Other well-known approaches such as BYOL [16] with

3 GNNs Versus Local Metrics

Different methods for understanding graphs can "understand" the difference between graphs in very different ways: a particular change to a graph might barely affect some features, while drastically changing others. One extreme case is when a given metric cannot detect that two distinct (non-

isomorphic) graphs are different. Since graph isomorphism is a computationally difficult problem, we expect that all efficiently computable graph representations "collapse" some pairs of input graphs.[1] It is conceivable, however, that one method could be strictly more powerful than another. For instance, since recent GNN models have overcome traditional models based on local metric representations in a variety of problems [24, 48, 52], is it the case that GNNs are strictly more powerful than local metrics?

We show, constructively, that the answer is no: there are indeed graphs which GNNs can distinguish and local metrics cannot, but there are also graphs which local metrics can distinguish but first-order GNNs cannot. Figure 1(a) shows a pair of graphs with the same degree distribution, clustering coefficient, and four-node orbits, which can nonetheless be distinguished by GNNs (proof in Appendix B). On the other hand, the graphs in Fig. 1(b) have different clustering coefficient and smallest cycle, but first-order GNNs cannot tell them apart. Thus, neither method strictly outperforms the other on all problems, and so there are theoretical generative models which perfectly match in GNN-based representations but differ in local metrics, and vice versa. This motivates our addition of local features to our graph representation models (Section 4).

It is much easier to incorporated such hard-coded structures into GNNs than it would be to add learning to feature metrics; in particular, counting higher-order local patterns quickly becomes prohibitively expensive, with a super-exponential time complexity. GNNs can also easily handle node and/or edge features on the underlying graphs, which is far more difficult to add to local metrics.

Another quality we would like in our graph representations, in addition to the ability to distinguish distinct graphs, is some form of stability: if a distribution of graphs only changes slightly, we would like our evaluation methods to give "partial credit" as opposed to a distribution where all graphs are dramatically different. (This is closely related to issues in training and evaluating image-based generative models [1–3, 46].) As previously discussed, the notion of "similar graphs" is very domain-dependent, but traditional local metrics can be highly sensitive to changes like adding a single edge: Fig. 1(c) shows an example where two graphs differing only by a single edge can have drastically different statistics. By learning GNN representations, we can have some control over these types of smoothness properties; we exploit this explicitly in our methodology in Section 4.

4 Self-supervised Training of Graph Representations for Evaluation

Suppose we have two sets of graphs $\mathbb{G}^{train} = \{G_1, \dots G_S\}$ and $\mathbb{G}^{test} = \{G_1, \dots G_M\}$, each sampled from the same data distribution $p(G)$. Also suppose that we have access to an unconditional graph generative model $g_\phi(.)$, which is trained on $\mathbb{G}^{train}$ to learn the distribution of the observed set of graphs. We sample a set of generated graphs $\mathbb{G}^{gen} = \{G_1, \dots G_N\} \sim p_{g_\phi}(G)$ from this model. In order to evaluate the quality of the sampled graphs (i.e., to decide whether the model $g_\phi(.)$ has successfully recovered the underlying distribution $p(G)$), we can define a measure of divergence $\mathcal{D}(\mathbb{G}^{test}, \mathbb{G}^{gen})$ to quantify the discrepancy between distributions of the real and generated graphs. One robust way to achieve this is to define the metric on latent vector representation spaces, and expect representations of graphs rather than the original objects. Thus, to use these metrics, we need to train a shared encoder $f_\theta(.)$ and then compute the discrepancy as $\mathcal{D}(f_\theta(\mathbb{G}^{test}), f_\theta(\mathbb{G}^{gen}))$. There are a few such metrics well-studied in visual domains that can differentiate the fidelity and diversity of the model, and which we can adopt in graph domains.

For evaluating image generative models, due to similar feature space across image datasets and also availability of large-scale data, the trunk of a model trained over ImageNet with explicit supervisory signals is usually chosen as the encoder. However, it is not straightforward to adopt the same trick to graph-structured data: there are no ImageNet-scale graph datasets available, and more importantly, the semantics of graphs and their features vary wildly across commonly used graph datasets, far more than occurs across distributions of natural images. For instance, even datasets of molecular graphs may use different feature sets to represent the molecules [18], in addition to the many cross-domain challenges discussed in Section 1. Thus, it is not feasible to imagine a single "universal" graph representation; we would like a general-purpose method for finding representations useful for a new graph dataset.

[1] Alternatively, an efficient graph representation might be able to distinguish non-isomorphic graphs, if it also sometimes distinguishes isomorphic graphs. We do not consider such representations in this paper.

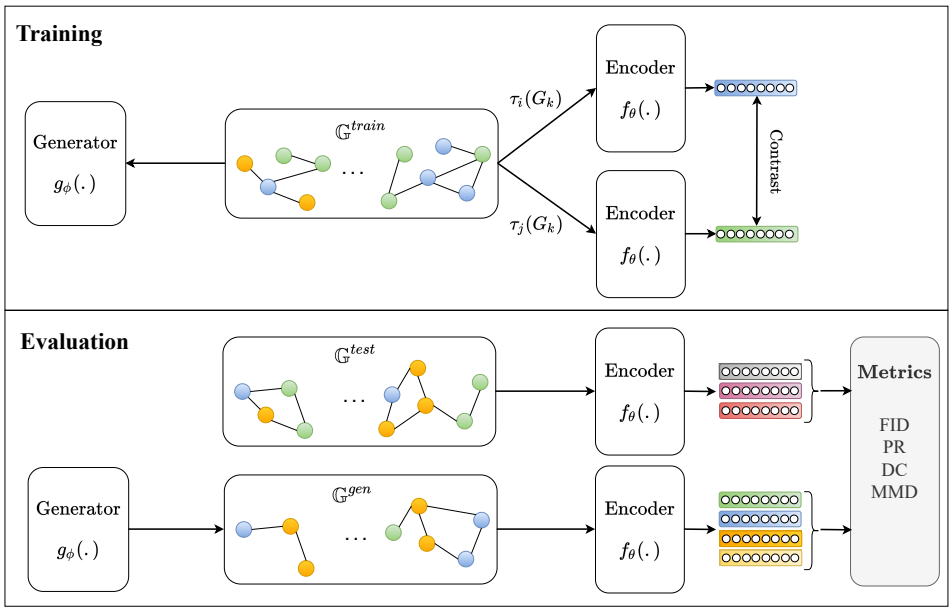

Figure 2: During the **training** phase, we use the same training data ($\mathbb{G}^{train}$) to train both the generator and the encoder networks. The encoder is trained using a contrastive loss where two augmentation τ_i and τ_j are randomly sampled from a set of rational augmentations $\mathcal{T}$ to construct two views of a sampled graph G_k. During **evaluation** phase, we sample the generator to form a generated set of graphs $\mathbb{G}^{gen}$ and feed it along with a held-out set of real graphs $\mathbb{G}^{test}$ to the encoder to compute the graph representations. The representations are then used to compute robust metrics to quantify the discrepency between real and generated graphs.

Training with Graph Contrastive Learning To find expressive representation of real and generated graphs, we train the encoder using a contrastive objective. Assuming a set of rational augmentations $\mathcal{T}$ over $\mathbb{G}$ where each augmentation $\tau_i \in \mathcal{T}$ is defined as a function over graph G_k that generates an identity-preserving view of the graph: $G_k^+ = \tau_i(G_k)$, a contrastive framework with negative sampling strategy uses $\mathcal{T}$ to draw positive samples from the joint distribution $p\left(\tau_i(G_k), \tau_j(G_k)\right)$ in order to maximize the agreement between different views of the same graph G_k and to draw negative samples from the product of marginals $p\left(\tau_i(G_k)\right) \times p\left(\tau_j(G_{k'})\right)$ to minimize it for views from two distinct graphs G_k and $G_{k'}, k \neq k'$.

As mentioned previously, we use a GIN architecture for our feature extractor. Other than the training process, the rest of our evaluation pipeline is similar to that of Thompson et al. [47], who use random GIN weights.We consider two methods for training our GIN's parameters: GraphCL [55] and InfoGraph [43]. GraphCL randomly samples augmentation from four possible augmentaiotns including *node dropping*, *edge perturbation*, *attribute masking*, and *sub-graph inducing* based on a random walk to which we would like the representation to be roughly invariant. GraphCL uses normalized temperature-scaled cross-entropy (NT-Xent) objective [7] to maximize the agreement between positive graph-level representations. InfoGraph works differently: it contrasts the graph-level representation with the representations of individual nodes, which encode neighborhood structure information. InfoGraph uses DeepInfoMax [22] objective to maximize the mutual information between graph-level and node-level representations of each individual graph.

Using Local Subgraph Counts as Input Features for GINs To build on the insight of Section 3, we also consider various methods for adding information about the local graph structure as node features, similarly to Graph Substructure Networks [6]. The simplest such method is to add the degree of a node as an explicit feature for the GIN to consider. We do this, but also add, by concatenation on node features, higher-order local information as well: three-node and four-node clustering features for each node. Four node clustering coefficient is calculated as:

$$C_4(v) = \frac{\sum_{(u,w)\in\mathcal{N}(v)} q_v(u,w)}{\sum_{(u,w)\in\mathcal{N}(v)} \left[deg(u) + deg(w) - q_v(u,w) - 2\mathbb{I}(u \in \mathcal{N}(w))\right]} \tag{1}$$

where $\mathcal{N}(v)$ denotes immediate neighbors of node v, $deg(v)$ denotes the degree of node v, and $q_v(u, w)$ is the number of common neighbors of u and w, not counting v. Aggregating these features across the whole graph would give information on distribution of 4 node orbits of the graph, but this provides more localized information across the graph that is nonetheless difficult or impossible for a GIN to examine otherwise [8].

Choice of Augmentations Wang et al. [51], building off work of Wang and Isola [50], study contrastive learning in general settings (with an eye towards vision applications), and provide an intriguing bound for the performance of downstream classifiers. To explain it, consider the "augmentation graph" of the training samples: if G_i is the ith training example and G_i^+ is a random augmentation of that example, we connect the nodes G_i and G_j in the augmentation graph if there are feasible augmentations G_i^+ and G_j^+ such that $G_i^+ = G_j^+$.[2] Wang et al. argue that downstream linear classifiers are likely to succeed if this augmentation graph has stronger intra-class connectivity than it does cross-class connectivity, proving a tight connection between the two under a particular setting with strong assumptions about this and related aspects of the setup.

Given the connection between the distribution metrics discussed in Appendix A.2 and classifier performance [e.g. 27, Section 4], if we accept the argument of Wang et al. [51], evaluation metrics based on a contrastively trained graph representation will give poor values (good classifier performance) when generated samples are not well-connected to real samples in the augmentation graph, and vice versa. If we choose augmentations appropriately, this is sensible behavior.

Enforcing Lipschitz Layers in Representation Networks The prior line of reasoning also suggests that we should choose augmentations that are capable of making real graphs look like one another. Edit distances between graphs, however, are typically large on the datasets we consider, and so augmentations based on adding or deleting individual nodes and/or edges will struggle to do this. The same is true for many of the augmentations used on images, except – as Wang et al. [51] note – for crop-type augmentations, where e.g. two different car pictures might become quite similar if we crop to just a wheel. On graphs, an analogous operation is subgraph sampling, which we include in our GraphCL setup; InfoGraph already naturally looks at subgraph features as a core component.

Taking this line of reasoning as well as the general motivations of contrastive learning further, it is also natural to think that if we can inherently enforce "similar graphs" to have similar representations, this could improve the process of contrastive learning: we would save on needing to train the model to learn these similarities, and it could help decrease the classifier performance for good generative models whose output graphs are legitimately near the distribution of target graphs.

A line of work on GANs in visual settings [1–3, 29, 30, 36] has made clear the importance of this type of reasoning in losses for training generative models: the loss should smoothly improve as generator samples approach the target distribution, even if the supports differ. Viewing model evaluation metrics as a kind of "out-of-the-loop" loss function for training generative models – hyperparameter selection and model development focusing on variants with better evaluation metrics – suggests that these kinds of properties may be important for the problem of evaluation as well.

We thus explicitly enforce the layers of our GIN to have a controlled Lipschitz constant, similarly to e.g. spectral normalization in GAN discriminators [30]. To this end, we fix the λ Lipschitz factor to 1.0 in the experiments. For each linear layer with weights $\mathbf{W}_\ell$, we use projected gradient descent; after each update on the weights, if $\|\mathbf{W}_\ell\| > 1.0$, we update the value of the weights to $\mathbf{W}_\ell = \frac{\mathbf{W}_\ell}{\|\mathbf{W}_\ell\|}$. This guarantees small changes in the graphs, such as adding/removing edges, or change in the input features, will not drastically change the final representation.

5 Experimental Results

In all of the experiments we train the model on the full dataset in an self-supervised manner. Following [47], we take the dataset and make perturbations on the dataset and see what is the trend in the

[2] Technically, the augmentations we use (described in Section 4) would result in a densely-connected augmentation graph: for instance, each graph has some vanishingly small probability of being reduced to an empty graph. We can instead think about the augmentation graph based on the augmentations from a high-probability set for each training example.

measurements as the perturbation degree increases. We denote perturbation degree with r, and define it for each type of perturbation. We use these type of perturbations:

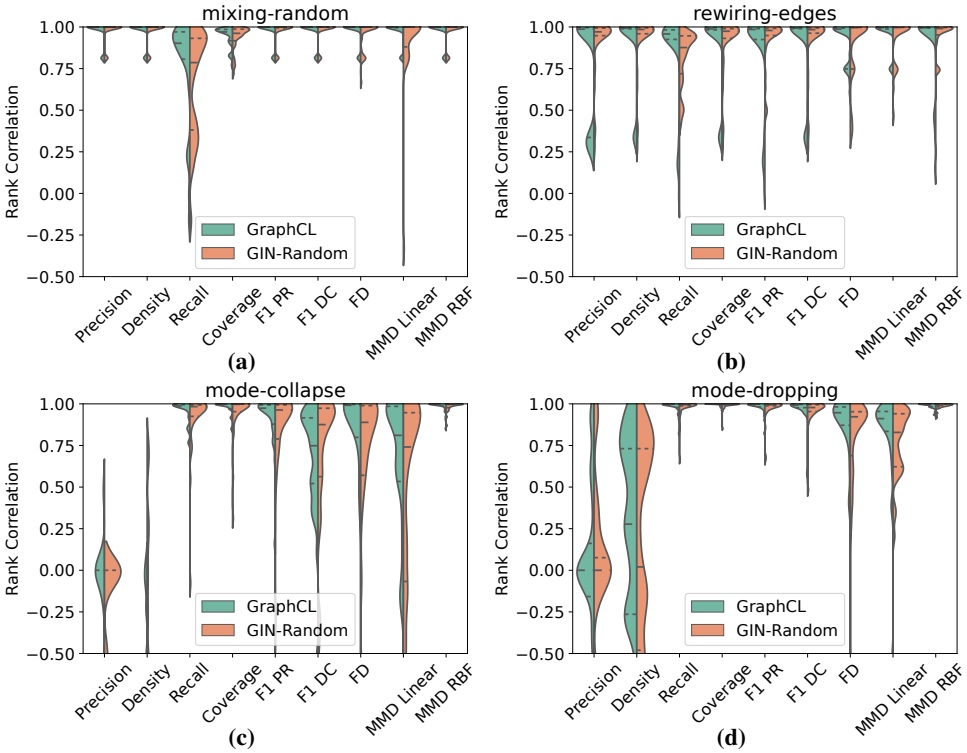

Figure 3: Violin results, comparing the distribution of the rank correlations between the pretrained GraphCL model and randomly initialized GIN network. Lines in the plots are showing the quartiles. Overall distribution shows better performance for self-supervised training. Results are gathered over all datasets and for all random seeds.

- **Mixing-Random:** In perturbation with ratio r, we remove r chunk of the reference samples, and replace them with Erdős–Rényi (ER) graphs with the same ratio of edges.
- **Rewiring Edges:** This perturbation, rewires each edge of the graph with probability r. Each rewired edge, will change one of the sides of the edge with equal probability to another node that is not already connected to the stable node.
- **Mode Collapse and Mode Dropping:** For these perturbation, first we cluster the graphs using the WL-Kernel. First, we choose r ratio of clusters, then, for mode collapse, we replace each graph on that dataset with the center of the cluster. For, mode dropping, we remove the selected clusters and then for making size of the dataset fixed, we randomly repeat samples from other clusters.

For each experiment, we measure the Spearman rand correlation between the perturbation ratio, r, and the value of the measurement. For measurements that supposed to decrease by the increase of ratio, we flip the values. As a result, in all experiments higher is better. We gather the results among different datasets and several random seeds and plot them for distribution of the correlations. For detailed experiments on the individual datasets see Appendix D.

Datasets: Following [47], we use six diverse datasets (three synthetic and three real-world) that are frequently used in literature including: (1) **Lobster** [9] consisting of stochastic graphs with each node at most 2-hops away from a backbone path, (2) 2D **Grid** graphs [9, 26, 53], (3) **Proteins** [11] where represents connectivity among amino acids if their physical distance is less than a threshold, (4) 3-hop **Ego** networks [53] extracted from the CiteSeer network [40] representing citation-based connectivity among documents, (5) **Community** [53] graphs generated by Erdős–Rényi model [12], and (6) **Zinc** [23] is a dataset of attributed molecular graphs which allows to study sensitivity of metrics to changes

in node/edge feature distributions. We follow the exact protocols used in [9, 26, 47, 53] as follows. We randomly sample 1000 graphs from Zinc dataset and use one-hot encoding for edge/node features. For community dataset, we set $n = |\mathcal{V}|/2$, $p = 0.3$, and add $0.05|\mathcal{V}|$ inter-community edges with uniform probability. For all datasets, we use the node size range indicated in Table 4 in Appendix.

Contrastive Training Versus Random GIN without Structural Features: In the first experiment, we compare the methods without adding structural features. Results can be seen in Figure 3. In general we can see that in most measurements pretraining shows superior performance compared to the random network. In this figure we have used GraphCL method. Overall InfoGraph shows similar results. For InfoGraph results check Appendix D. In this experiments we use 10 random seeds and gather data on all datasets. In Appendix D, same results separated for each dataset can be seen. In our experience, pretraining shows near perfect results on larger datasets, but for Lobster and Grid datasets correlations are not near to 1. Our observation is these measurements are moving with the correct trend up to some perturbation ratio; but for example precision/recall become zero after some perturbation and keeps still. Our intuition here is because of highly regular structures in these datasets, model learns instantly to discrimate the real graphs from the fake ones very easily. And after small amount of perturbation the perturbed graphs are all very far from the reference graphs.

Experiment	Model Name	Precision	Density	Recall	Coverage	F1PR	F1DC	FD	MMD Lin	MMD RBF
Mixing Random	GIN-Random	0.97/1.0	0.97/1.0	0.64/0.79	0.93/0.96	0.97/1.0	0.97/1.0	0.94/1.0	0.91/1.0	0.97/1.0
	GraphCL	0.97/1.0	0.97/1.0	0.77/0.9	0.94/0.97	0.97/1.0	0.97/1.0	0.97/1.0	0.97/1.0	0.97/1.0
	GraphCL Full	1.0/1.0	1.0/1.0	0.77/0.89	0.95/0.97	1.0/1.0	1.0/1.0	0.97/1.0	0.95/1.0	1.0/1.0
Rewiring Edges	GIN-Random	0.89/0.97	0.9/0.99	0.81/0.88	0.88/0.97	0.91/0.98	0.9/0.98	0.87/1.0	0.91/1.0	0.94/0.99
	GraphCL	0.76/0.99	0.88/0.99	0.83/0.96	0.88/0.99	0.84/0.99	0.88/0.99	0.9/0.99	0.95/1.0	0.89/1.0
	GraphCL Full	0.87/0.98	0.88/0.99	0.79/0.89	0.87/0.98	0.87/0.97	0.88/0.99	0.95/0.99	0.97/1.0	0.97/1.0
Mode Collapse	GIN-Random	-0.51/-0.68	-0.64/-0.88	0.9/0.98	0.94/1.0	0.82/0.96	0.68/0.88	0.6/0.89	0.44/0.74	0.98/1.0
	GraphCL	-0.6/-0.88	-0.76/-0.88	0.95/0.99	0.98/1.0	0.91/0.97	0.67/0.75	0.74/0.99	0.59/0.81	0.98/1.0
	GraphCL Full	-0.62/-0.95	-0.56/-0.8	0.93/0.99	0.96/1.0	0.9/0.97	0.66/0.81	0.77/0.99	0.72/0.9	0.98/1.0
Mode Dropping	GIN-Random	-0.15/0.0	0.09/0.02	0.98/1.0	0.99/1.0	0.97/0.99	0.96/0.98	0.77/0.92	0.76/0.83	0.98/0.99
	GraphCL	-0.0/0.0	0.2/0.28	0.98/0.99	0.99/1.0	0.98/0.99	0.97/0.99	0.84/0.95	0.81/0.91	0.99/0.99
	GraphCL Full	0.07/0.0	0.34/0.37	0.97/1.0	0.99/1.0	0.97/0.99	0.97/0.99	0.88/0.97	0.86/0.92	0.99/1.0

Table 1: Mean/Median values from Figures 3 and 4. Table summarizes the distributions by their mean and median values. Results are gathered over all datasets and for all random seeds.

Contrastive Training with Structural Information: In this experiment we concatenate node degrees and node clustering coefficients for each node feature. Here we can see that adding these features improves the models power for some measurements. However, on small datasets Grid and Lobster results become poorer. Again intuition is more powerful model can easier distinguish main graphs from the fake ones. Break down of the results on the datasets in Appendix D proves this point.

Generative Models Benchmark In this experiment, we use samples generated from the GRAN [26] after 25% of epochs, after full training, and also compare to a holdout set of the real data (half of the original dataset). We pretrain a model with full structural features and evaluate the criteria on them. As expected, the early-training results are worse than post-training results, while the real data gives almost perfect results. Results are provided in Table 2.

Table 2: Comparative results for training GRAN model for 25% of epochs, full training, and $50/50$ split of the datasets. GGM stands for Graph Generative Model.

Dataset	GGM	Precision↑	Density↑	Recall ↑	Coverage↑	F1PR↑	F1DC↑	FD↓	MMD Lin↓	MMD RBF↓
Lobster	GRAN-25%	0.24	0.23	0.56	0.43	0.34	0.30	$6.4e5$	$3.1e4$	0.24
	GRAN-100%	0.38	0.27	0.75	0.47	0.50	0.34	$4.7e5$	$2.1e4$	0.14
	50/50 split	1.0	1.0	0.98	1.0	0.99	1.0	3.7	0.53	0.001
Grid	GRAN-25%	0.0	0.0	1.0	0.0	0.0	0.0	$3.6e9$	$2.3e9$	0.94
	GRAN-100%	0.33	0.29	1.0	0.76	0.50	0.42	$1.4e6$	$1.57e5$	0.075
	50/50 split	1.0	1.0	1.0	1.0	1.0	1.0	0.86	0.81	0.009

Ablation Study: We analyze how layer normalization and using subgraph augmentations in GraphCL reflects on the final results. We have used no structural feature setup and conducted the experiment for mixing-random and mode-collapse experiments. Figure 5, shows the results on this task. The results prove that both of these improvements are essential for getting better results.

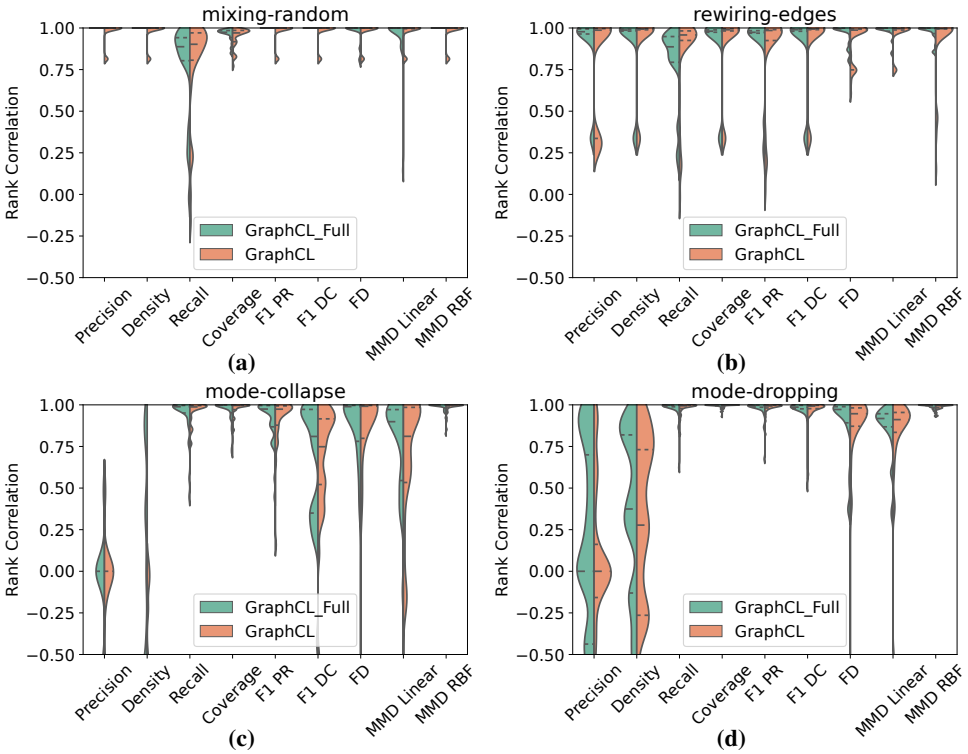

Figure 4: Violin results, comparing the distribution of the rank correlations between the pretrained GraphCL models with/without structural features. GraphCL is without structural features, and GraphCL_Full indicates GraphCL model + full structural features. Lines in the plots are showing the quartiles. Overall distribution shows better performance with augmentation of the structural features. Results are gathered over all datasets and for all random seeds.

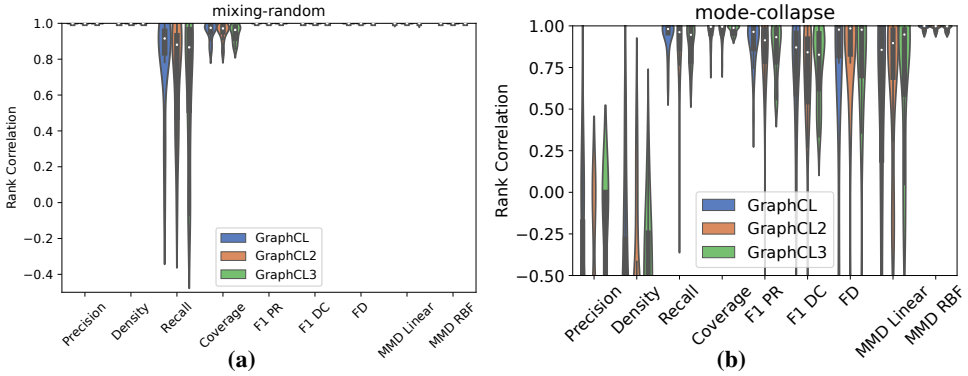

Figure 5: Comparison of the full method on no feature data versus removing layer normalization versus removing subgraph data augmentations on pretraining the network. GraphCL is normal training. In GraphCL2 we do not enforce lipschitzness. In GraphCL3 we remove the subgraph constraints and reduce the probability of node and edge dropping in the augmentations.

6 Conclusion

We have demonstrated that self-supervised pre-training of representations can yield significantly better metrics for graph evaluation than random ones, particularly when incorporating local graph features with Lipschitz control, as inspired by theory. We suggest graph generative modeling papers should consider evaluating with these metrics in addition to or instead of their existing ones.

References

[1] Michael Arbel, Danica J. Sutherland, Mikołaj Bińkowski, and Arthur Gretton. "On Gradient Regularizers for MMD GANs." *NeurIPS*. 2018. arXiv: 1805.11565.

[2] Martin Arjovsky and Léon Bottou. "Towards Principled Methods for Training Generative Adversarial Networks." *ICLR*. 2017. arXiv: 1701.04862.

[3] Martin Arjovsky, Soumith Chintala, and Léon Bottou. "Wasserstein GAN." *ICML*. 2017. arXiv: 1701.07875.

[4] Mikołaj Bińkowski, Danica J. Sutherland, Michael Arbel, and Arthur Gretton. "Demystifying MMD GANs." *ICLR*. 2018. arXiv: 1801.01401.

[5] Ali Borji. "Pros and cons of GAN evaluation measures." *Computer Vision and Image Understanding* 179 (2019), pp. 41–65.

[6] Giorgos Bouritsas, Fabrizio Frasca, Stefanos P Zafeiriou, and Michael Bronstein. "Improving graph neural network expressivity via subgraph isomorphism counting." *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2022).

[7] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. *A simple framework for contrastive learning of visual representations*. 2020. arXiv: 2002.05709.

[8] Zhengdao Chen, Lei Chen, Soledad Villar, and Joan Bruna. "Can graph neural networks count substructures?" *NeurIPS* 33 (2020), pp. 10383–10395.

[9] Hanjun Dai, Azade Nazi, Yujia Li, Bo Dai, and Dale Schuurmans. "Scalable deep generative modeling for sparse graphs." *ICML*. PMLR. 2020, pp. 2302–2312.

[10] Nicola De Cao and Thomas Kipf. "MolGAN: An implicit generative model for small molecular graphs" (2018). arXiv: 1805.11973.

[11] Paul D Dobson and Andrew J Doig. "Distinguishing enzyme structures from non-enzymes without alignments." *Journal of Molecular Biology* 330.4 (2003), pp. 771–783.

[12] Paul Erdos, Alfréd Rényi, et al. "On the evolution of random graphs." *Publ. Math. Inst. Hung. Acad. Sci* 5.1 (1960), pp. 17–60.

[13] Daniel Flam-Shepherd, Tony Wu, and Alan Aspuru-Guzik. "Graph deconvolutional generation" (2020). arXiv: 2002.07087.

[14] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. "Neural message passing for quantum chemistry." *ICML*. 2017, pp. 1263–1272.

[15] Arthur Gretton, Karsten M. Borgwardt, Malte J. Rasch, Bernhard Schölkopf, and Alexander Smola. "A Kernel Two-Sample Test." *Journal of Machine Learning Research* 13.25 (2012), pp. 723–773.

[16] Jean-Bastien Grill et al. "Bootstrap Your Own Latent - A New Approach to Self-Supervised Learning." *NeurIPS*. 2020, pp. 21271–21284.

[17] Xiaojie Guo and Liang Zhao. "A systematic survey on deep generative models for graph generation" (2020). arXiv: 2007.06686.

[18] Kaveh Hassani. "Cross-Domain Few-Shot Graph Classification." *Proceedings of the AAAI Conference on Artificial Intelligence* 36.6 (2022), pp. 6856–6864.

[19] Kaveh Hassani and Amir Hosein Khasahmadi. "Contrastive Multi-View Representation Learning on Graphs." *ICML*. 2020, pp. 4116–4126.

[20] Kaveh Hassani and Amir Hosein Khasahmadi. "Learning Graph Augmentations to Learn Graph Representations" (2022). arXiv: 2201.09830.

[21] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. "GANs trained by a two time-scale update rule converge to a local Nash equilibrium." *NeurIPS* (2017).

[22] R Devon Hjelm, Alex Fedorov, Samuel Lavoie-Marchildon, Karan Grewal, Phil Bachman, Adam Trischler, and Yoshua Bengio. "Learning deep representations by mutual information estimation and maximization." *ICLR*. 2019.

[23] John J Irwin, Teague Sterling, Michael M Mysinger, Erin S Bolstad, and Ryan G Coleman. "ZINC: a free tool to discover chemistry for biology." *Journal of chemical information and modeling* 52.7 (2012), pp. 1757–1768.

[24] Thomas N Kipf and Max Welling. "Semi-supervised classification with graph convolutional networks." *ICLR*. 2016. arXiv: 1609.02907.

[25] Yujia Li, Oriol Vinyals, Chris Dyer, Razvan Pascanu, and Peter Battaglia. "Learning Deep Generative Models of Graphs." *ICLR*. 2018.

[26] Renjie Liao, Yujia Li, Yang Song, Shenlong Wang, Charlie Nash, William L Hamilton, David Duvenaud, Raquel Urtasun, and Richard S Zemel. "Efficient graph generation with graph recurrent attention networks" (2019). arXiv: 1910.00760.

[27] Feng Liu, Wenkai Xu, Jie Lu, Guangquan Zhang, Arthur Gretton, and Danica J. Sutherland. "Learning Deep Kernels for Non-Parametric Two-Sample Tests." *ICML*. 2020. arXiv: 2002.09116.

[28] Kaushalya Madhawa, Katushiko Ishiguro, Kosuke Nakago, and Motoki Abe. "GraphNVP: An invertible flow model for generating molecular graphs" (2019). arXiv: 1905.11600.

[29] Lars M. Mescheder, Andreas Geiger, and Sebastian Nowozin. "Which Training Methods for GANs do actually Converge?" *ICML*. 2018. arXiv: 1801.04406.

[30] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. "Spectral Normalization for Generative Adversarial Networks." *ICLR*. 2018. arXiv: 1802.05957.

[31] Stanislav Morozov, Andrey Voynov, and Artem Babenko. "On Self-Supervised Image Representations for GAN Evaluation." *ICLR*. 2021.

[32] Stanislav Morozov, Andrey Voynov, and Artem Babenko. "On self-supervised image representations for GAN evaluation." *ICLR*. 2020.

[33] Muhammad Ferjad Naeem, Seong Joon Oh, Youngjung Uh, Yunjey Choi, and Jaejun Yoo. "Reliable fidelity and diversity metrics for generative models." *ICML*. PMLR. 2020, pp. 7176–7185.

[34] Leslie O'Bray, Max Horn, Bastian Rieck, and Karsten Borgwardt. "Evaluation Metrics for Graph Generative Models: Problems, Pitfalls, and Practical Solutions" (2021). arXiv: 2106.01098.

[35] Jiezhong Qiu, Qibin Chen, Yuxiao Dong, Jing Zhang, Hongxia Yang, Ming Ding, Kuansan Wang, and Jie Tang. "GCC: Graph Contrastive Coding for Graph Neural Network Pre-Training." *International Conference on Knowledge Discovery and Data Mining*. 2020.

[36] Kevin Roth, Aurélien Lucchi, Sebastian Nowozin, and Thomas Hofmann. "Stabilizing Training of Generative Adversarial Networks through Regularization." *NeurIPS*. 2017. arXiv: 1705.09367.

[37] Olga Russakovsky et al. "ImageNet Large Scale Visual Recognition Challenge." *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252.

[38] Mehdi SM Sajjadi, Olivier Bachem, Mario Lucic, Olivier Bousquet, and Sylvain Gelly. "Assessing generative models via precision and recall." *NeurIPS*. 2018, pp. 5234–5243.

[39] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. "Improved techniques for training GANs." *NeurIPS* (2016), pp. 2234–2242.

[40] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. "Collective classification in network data." *AI magazine* 29.3 (2008), pp. 93–93.

[41] Hamed Shirzad, Hossein Hajimirsadeghi, Amir H Abdi, and Greg Mori. "TD-GEN: Graph Generation With Tree Decomposition" (2021). arXiv: 2106.10656.

[42] Fan-Yun Sun, Jordan Hoffman, Vikas Verma, and Jian Tang. "InfoGraph: Unsupervised and Semi-supervised Graph-Level Representation Learning via Mutual Information Maximization." *ICLR*. 2020.

[43] Fan-Yun Sun, Jordan Hoffmann, Vikas Verma, and Jian Tang. "Infograph: Unsupervised and semi-supervised graph-level representation learning via mutual information maximization" (2019). arXiv: 1908.01000.

[44] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. "Rethinking the Inception Architecture for Computer Vision." *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[45] Shantanu Thakoor, Corentin Tallec, Mohammad Gheshlaghi Azar, Rémi Munos, Petar Veličković, and Michal Valko. "Bootstrapped representation learning on graphs" (2021). arXiv: 2102.06514.

[46] Lucas Theis, Aäron van den Oord, and Matthias Bethge. "A note on the evaluation of generative models." *ICLR* (2016).

[47] Rylee Thompson, Boris Knyazev, Elahe Ghalebi, Jungtaek Kim, and Graham W Taylor. "On Evaluation Metrics for Graph Generative Models" (2022). arXiv: 2201.09871.

[48] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. "Graph Attention Networks." *ICLR*. 2018. arXiv: 1710.10903.

[49] Petar Veličković, William Fedus, William L. Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. "Deep Graph Infomax." *ICLR*. 2019.

[50] Tongzhou Wang and Phillip Isola. "Understanding contrastive representation learning through alignment and uniformity on the hypersphere." *ICML*. PMLR. 2020, pp. 9929–9939.

[51] Yifei Wang, Qi Zhang, Yisen Wang, Jiansheng Yang, and Zhouchen Lin. "Chaos is a Ladder: A New Theoretical Understanding of Contrastive Learning via Augmentation Overlap" (2022). arXiv: 2203.13457.

[52] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. "How powerful are graph neural networks?" (2018). arXiv: 1810.00826.

[53] Jiaxuan You, Rex Ying, Xiang Ren, William Hamilton, and Jure Leskovec. "Graphrnn: Generating realistic graphs with deep auto-regressive models." *ICML*. PMLR. 2018, pp. 5708–5717.

[54] Yuning You, Tianlong Chen, Yang Shen, and Zhangyang Wang. "Graph Contrastive Learning Automated" (2021). arXiv: 2106.07594.

[55] Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. "Graph contrastive learning with augmentations." *NeurIPS* (2020).

[56] Yanqiao Zhu, Yichen Xu, Feng Yu, Qiang Liu, Shu Wu, and Liang Wang. "Deep graph contrastive representation learning" (2020). arXiv: 2006.04131.

[57] Yanqiao Zhu, Yichen Xu, Feng Yu, Qiang Liu, Shu Wu, and Liang Wang. "Graph contrastive learning with adaptive augmentation." *Web Conference*. 2021, pp. 2069–2080.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]