LEARNING GRAPH AUGMENTATIONS TO LEARN GRAPH REPRESENTATIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

Devising augmentations for graph contrastive learning is challenging due to their irregular structure, drastic distribution shifts, and nonequivalent feature spaces across datasets. We introduce LG2AR, **L**earning **G**raph **A**ugmentations to **L**earn **G**raph **R**epresentations, which is an end-to-end automatic graph augmentation framework that helps encoders learn generalizable representations on both node and graph levels. LG2AR consists of a probabilistic policy that learns a distribution over augmentations and a set of probabilistic augmentation heads that learn distributions over augmentation parameters. We show that LG2AR achieves state-of-the-art results on 18 out of 20 graph-level and node-level benchmarks compared to previous unsupervised models under both linear and semi-supervised evaluation protocols.

1 INTRODUCTION

Graph Neural Networks (GNNs) are a class of deep models that learn node representations over order-invariant and variable-size data, structured as graphs, through an iterative process of transferring, transforming, and aggregating the representations from topological neighbors. The learned representations are then summarized into a graph-level representation (Li et al., 2015; Gilmer et al., 2017; Kipf & Welling, 2017; Veličković et al., 2018; Xu et al., 2019; Khasahmadi et al., 2020a). GNNs are applied to non-Euclidean data such as point clouds (Hassani & Haley, 2019b), robot designs (Wang et al., 2019b), physical processes (Sanchez-Gonzalez et al., 2020), molecules (Duvenaud et al., 2015), social networks (Kipf & Welling, 2017), and knowledge graphs (Vivona & Hassani, 2019).

GNNs are mostly trained end-to-end with supervision from task-dependent labels. Nevertheless, annotating graphs is more challenging compared to other common modalities because they usually represent concepts in specialized domains such as biology where labeling through wet-lab experiments is resource-intensive (You et al., 2020; Hu et al., 2020) and labeling them procedurally using domain knowledge is costly(Sun et al., 2020). To address this, unsupervised objectives are coupled with GNNs to learn representations without relying on labels which are transferable to a priori unknown down-stream tasks. Reconstruction-based methods, i.e., Graph AutoEncoders (GAE), preserve the topological closeness of the nodes in the representations by forcing the model to recover the neighbors from the latent space (Kipf & Welling, 2016; Garcia Duran & Niepert, 2017; Pan et al., 2018; Park et al., 2019). GAEs over-emphasize proximity information at the cost of structural information (Veličković et al., 2019). Contrastive methods train graph encoders by maximizing the Mutual Information (MI) between node-node, node-graph, or graph-graph representations achieving state-of-the-art results on both node and graph classification benchmarks(Veličković et al., 2019; Sun et al., 2020; Hassani & Khasahmadi, 2020; You et al., 2020; Zhu et al., 2021). For a review on graph contrastive learning see (Wu et al., 2021; Xie et al., 2021; Liu et al., 2021a).

Contrastive learning is essentially learning invariances to data augmentations which are thoroughly explored for Computer Vision (CV) (Shorten & Khoshgoftaar, 2019) and Natural Language Processing (NLP) (Feng et al., 2021). Learning policies to sample dataset-conditioned augmentations is also studied in CV (Hataya et al., 2020; Lim et al., 2019; Cubuk et al., 2019; Li et al., 2020). The irregular structure of graphs complicates both adopting augmentations used on images and also devising new augmentation strategies (Zhao et al., 2021). Unlike image datasets where the distribution is mostly from natural images, graph datasets are abstractions diverse in nature and contain shifts on marginal/conditional distributions and nonequivalent feature spaces across datasets. This implies that

the effect of augmentations is different across the datasets and hence both augmentations and their selection policy should be learned from the data to adapt to new datasets.

Present Work. We introduce LG2AR, **L**earning **G**raph **A**ugmentations to **L**earn **G**raph **R**epresentations, a fully-automated end-to-end contrastive learning framework that helps encoders learn transferable representations. Specifically, our contributions are as follows: (1) We introduce a probabilistic augmentation selection policy that learns a distribution over the augmentation space conditioned on the dataset to automate the combinatorial augmentation selection process. (2) We introduce probabilistic augmentation heads where each head learns a distribution over the parameters of a specific augmentation to learn better augmentations for each dataset. (3) We train the policy and the augmentations end-to-end without requiring an outer-loop optimization and show that unlike other methods, our approach can be used for both node-level and graph-level tasks. Finally, (4) we exhaustively evaluate our approach under linear and semi-supervised evaluation protocols and show that it achieves state-of-the-art results on 18 out of 20 graph and node level classification benchmarks.

2 Related Work

Graph augmentations are explored in supervised settings to alleviate over-smoothing and over-fitting. DropEdge (Rong et al., 2020) randomly drops a fraction of the edges during training. ADAEDGE (Chen et al., 2020a) learns to perturb edges between based on the predicted node classes. BGCN (Zhang et al., 2019) generates an ensemble of deonised graphs by perturbing edges. GAUG (Zhao et al., 2021) trains a GAE to generate edge probabilities and interpolates them with the original connectivity to sample a denoised graph. FLAG (Kong et al., 2020) augments node features with adversarial perturbations and GraphMask (Schlichtkrull et al., 2021) introduces differentiable edge masking to achieve interpretability. These works assume that a specific type of augmentation suffice for all supervised tasks and do not utilize the benefit of mixing the augmentations.

Graph augmentations are also studied in contrastive setting to learn transferable graph representations. DGI (Veličković et al., 2019) and InfoGraph (Sun et al., 2020) adopt DeepInfoMax (Hjelm et al., 2019) and enforce the consistency between local (node) and global (graph) representation. MVGRL (Hassani & Khasahmadi, 2020) augments a graph via graph diffusion and constructs two views by randomly sampling sub-graphs from the adjacency and diffusion matrices. GCC (Qiu et al., 2020) uses sub-graph instance discrimination and contrasts sub-graphs from a multi-hop ego network. GraphCL (You et al., 2020) uses trial-and-error to hand-pick graph augmentations and the corresponding parameters of each augmentation. JOAO (You et al., 2021) extends the GraphCL using a bi-level min-max optimization that learns to select the augmentations. Nevertheless, it does not show much improvement over GraphCL. GRACE (Zhu et al., 2020) uses a similar approach to GraphCL learn node representations. GCA (Zhu et al., 2021) uses a set of heuristics to adaptively pick the augmentation parameters. BGRL (Thakoor et al., 2021) adopts BYOL (Grill et al., 2020) and uses random augmentations to learn node representations. ADGCL (Suresh et al., 2021) introduces adversarial graph augmentation strategies to avoid capturing redundant information. Different from these methods, LG2AR emphasizes the importance of conditional augmentations and learns a distribution over the augmentation space along with a set of distributions over the augmentation parameters end-to-end without requiring a bi-level optimization and outperforms the previous contrastive methods on both node and graph level benchmarks under linear and semi-supervised evaluation protocols.

3 Method

Given a dataset of graphs $\mathcal{G} = \{G_k\}_{k=1}^{N}$ where each graph $G_k = (\mathcal{V}, \mathcal{E}, \mathbf{X})$ consists of $|\mathcal{V}|$ nodes, $|\mathcal{E}|$ edges ($\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$), and initial node features $\mathbf{X} \in \mathbb{R}^{|\mathcal{V}| \times d_x}$, and assuming that the semantic labels are not available during the training, the goal is to learn node-level representations $\mathbf{H}_v \in \mathbb{R}^{|\mathcal{V}| \times d_h}$ and graph-level representation $h_G \in \mathbb{R}^{d_h}$ such that the learned representations are transferable to the down-stream tasks unknown a priori. Assuming a set of possible rational augmentations $\mathcal{T}$over $\mathcal{G}$ where each augmentation $\tau_i \in \mathcal{T}, i \in \{1, ..., |\mathcal{T}|\}$ is defined as a function over graph G_k that generates an identity-preserving view of the graph: $G_k^i = \tau_i(G_k)$, 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

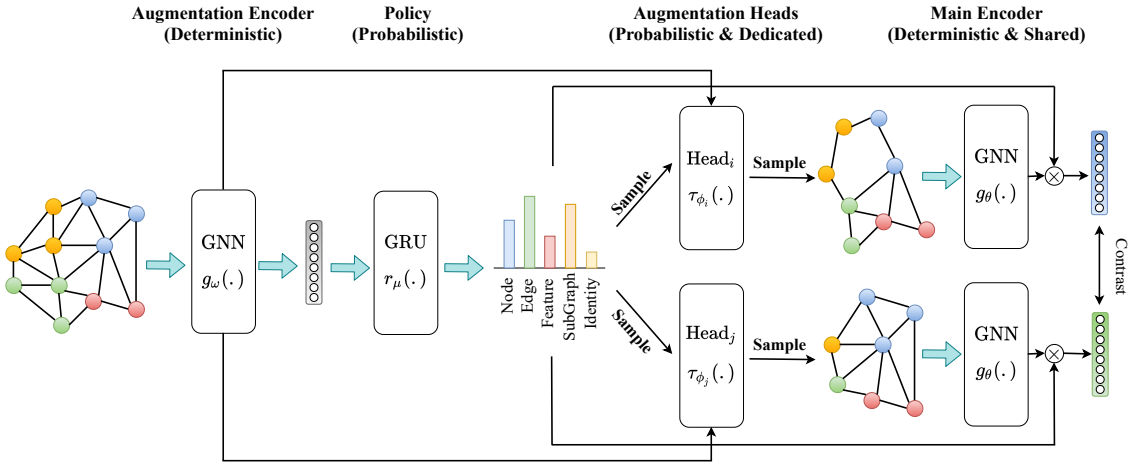

Figure 1: The proposed framework for learning graph augmentation end-to-end.

from two distinct graphs G_k and $G_{k'}, k \neq k'$. Most works, sample the augmentations uniformly and use trail-and-error to determine a single parameter for each augmentation, e.g., the probability of dropping nodes. LG2AR, on the other hand, learns the sampling distribution over $\mathcal{T}$ and also learns parametric augmentation τ_{ϕ_i} end-to-end along with the representations to generate robust representations. The architecture of LG2AR (Figure 1) achieves this using an augmentation encoder, a probabilistic policy, a set of probabilistic augmentation heads, and a shared base encoder. The details are as follows.

3.1 Augmentation Encoder

Augmentation encoder $g_\omega(.) : \mathbb{R}^{|\mathcal{V}|\times d_x} \times \mathbb{R}^{|\mathcal{E}|} \longmapsto \mathbb{R}^{|\mathcal{V}|\times d_h} \times \mathbb{R}^{d_h}$ learns a set of node encoding $\mathbf{H}_v \in \mathbb{R}^{|\mathcal{V}|\times d_h}$ and a graph encoding $h_g \in \mathbb{R}^{d_h}$ over the input graph G_k to provide the subsequent modules with expressive encodings so that they can condition their predictions on the input graphs. The augmentation encoder consists of a GNN producing node representations, a read-out function (summation) aggregating the representations into a graph representation, and two dedicated projection heads (three layer MLPs) applied to the learned representations. To encode graphs, we opted for expressive power and adopted graph isomorphism network (GIN) (Xu et al., 2019).

3.2 Policy

Policy $r_\mu(.) : \mathbb{R}^{|\mathcal{B}|\times d_h} \longmapsto \mathbb{R}^{|\tau|}$ is a probabilistic module that receives a batch of graph-level representations $\mathbf{H}_g \in \mathbb{R}^{|\mathcal{B}|\times d_h}$ from the augmentation encoder, constructs a categorical distribution over the possible augmentations $\mathcal{T}$, and then samples two augmentations, τ_{ϕ_i} and τ_{ϕ_j} from that distribution for each batch with temperature t. It is shown that conditioning the augmentation sampling on the dataset helps achieve better performance (You et al., 2020). However, feeding the whole dataset to the policy module is computationally expensive and hence we approximate it by conditioning the policy on mini-batches. Moreover, the policy must be invariant to the order of representations within the batch. To enforce this, we tried two strategies: (1) a policy instantiated as a deep set (Zaheer et al., 2017) where representations are first projected and then aggregated into a *batch representation*, and (2) a policy instantiated as an RNN where we impose an order on the representations by sorting them based on their L_2-norm and then feeding them into a GRU (Cho et al., 2014). We use the last hidden state as the *batch representation*. We observed that GRU based policy performed better. The policy module automates the ad-hoc trial-and-error augmentation selection process. To let the gradients flow back to the policy module, we use a skip-connection and multiply the final graph representations by their associated augmentation probabilities predicted by the policy.

3.3 AUGMENTATIONS

We use five graph augmentations including three structural augmentations: *node dropping*,*edge perturbation*, and *sub-graph inducing*, one feature augmentation: *feature masking*, and one *identity augmentation*. These augmentations enforce the priors that the semantic meaning of a graph should not change due to perturbations applied to its features or structure. For efficiency, we do not use compute-intensive augmentations such as graph diffusion. Unlike previous works in which the parameters of the augmentations are chosen either randomly or by heuristics, we opt to learn them end-to-end. For example, rather than dropping nodes randomly(You et al., 2020) or computing the probability proportional to a centrality measure (Zhu et al., 2021), we train a model to predict the distribution over all nodes within a graph and then sample from it to decide which nodes to drop. Unlike the policy module, the augmentations are conditioned on the individual graphs. We use a dedicated head for each augmentation modeled as a two layer MLP that learns a distribution over the augmentation parameters. The inputs to the heads are the original graph G and representations $\mathbf{H}_v$ and h_G from the augmentation encoder. We sample the learned distribution using Gumbel-Softmax trick (Jang et al., 2016; Maddison et al., 2016) with temperature t.

Node Dropping Head is conditioned on the node and graph representations to decide which nodes within a graph to drop. It receives the concatenation of the node and graph representations as input and predicts a categorical distribution over the nodes. The distribution is then sampled using Gumbel-Top-K trick (Kool et al., 2019) with a ratio hyper-parameter. We also tried Bernoulli sampling but we observed that it aggressively drops nodes in the few first epochs and the model cannot recover later. To let the gradients flow back from the augmented graph to the head, we introduce edge weights on the augmented graph where an edge weight w_{ij} is computed as $p(v_i) + p(v_j)$ and $p(v_i)$ is the probability assigned to node v_i. See Algorithm 2 in Appendix.

Edge Perturbation Head is conditioned on head and tail nodes to decide which edges to add or remove. To achieve this, $|\mathcal{E}|$ *negative edges* ($\bar{\mathcal{E}}$) are first randomly sampled to form a set of negative and positive edges $\mathcal{E} \cup \bar{\mathcal{E}}$ with size of $2|\mathcal{E}|$. Edges represented as $\big[h_{v_i} + h_{v_j} \parallel \mathbb{1}_{\mathcal{E}}(e_{ij})\big]$($h_{v_i}$ and $h_{v_j}$are the representations of head and tail nodes of edge e_{ij} and $\mathbb{1}_{\mathcal{E}}(e_{ij})$ is an indicator function indicating if the edge belongs to positive or negative edges) are fed into the head to learn Bernoulli distributions over the edged. We use the predicted probabilities $p(e_{ij})$ as the edge weights to let the gradients flow back to the head. See Algorithm 3 in Appendix.

Sub-graph Inducing Head is conditioned on the node and graph representations to decide which node to select as the center node. It receives the concatenation of the node and graph representations (i.e., $[h_v \parallel h_g]$) as input and learns a categorical distribution over the nodes. The distribution is then sampled to select a central node per graph around which a sub-graph is induced using Breadth-First Search (BFS) with K hops. We use a similar trick to node dropping augmentation to overpass the gradients back to the original graph. See Algorithm 4 in Appendix.

Feature Masking Head is conditioned on the node representation to decide which dimensions of the node feature to mask. The head receives the node representation h_v and learns a Bernoulli distribution over each feature dimension of the original node feature. The distribution is then sampled to construct a binary mask m over the initial feature space. Because initial node features can consist of categorical attributes (one-hot or multi-hot), we use a linear layer to project them into a continuous space resulting in x'_v. The augmented graph has the same structure as the original graph with initial node features $x'_v \odot m$ where $\odot$ is the Hadamard product. See Algorithm 5 in Appendix.

3.4 BASE ENCODER

Base encoder $g_\theta(.) : \mathbb{R}^{|\mathcal{V}'|\times d'_x} \times \mathbb{R}^{|\mathcal{V}'|\times|\mathcal{V}'|} \longmapsto \mathbb{R}^{|\mathcal{V}'|\times d_h} \times \mathbb{R}^{d_h}$ is a shared graph encoder among the augmentations which receives an augmented graph $G' = (\mathcal{V}', \mathcal{E}')$ from the corresponding augmentation head and learns a set of node representations $\mathbf{H}'_v \in \mathbb{R}^{|\mathcal{V}'|\times d_h}$ and a graph representation $h'_G \in \mathbb{R}^{d_h}$ over the augmented graph G'. The goal of learning the augmentations is to help the base encoder learn invariances to such augmentations and as a result produce robust representations. The base encoder is trained end-to-end with the policy and augmentation heads. At inference time, the input graphs are directly fed to the base encoder to compute the encodings for down-stream tasks.

3.5 TRAINING

We follow (Sun et al., 2020) and train the framework end-to-end using deep InfoMax (Hjelm et al., 2019) and maximize the MI between the node representations of one view with graph representation of the other view and vice versa with the following objective:

$$\max_{\omega,\mu\phi_i,\phi_j,\theta} \frac{1}{|\mathcal{G}|} \sum_{G\in\mathcal{G}} \left[\frac{1}{|\mathcal{V}|} \sum_{v\in\mathcal{V}} \left[\mathrm{I}\left(h_v^i, h_G^j\right) + \mathrm{I}\left(h_v^j, h_G^i\right) \right] \right] \tag{1}$$

where $\omega, \mu, \phi_i, \phi_j, \theta$ are parameters of modules to be learned, h_v^i, h_G^j are representations of node v and graph G encoded by augmentation i and j, and I is the mutual information estimator. We use the Jensen-Shannon MI estimator:

$$\mathrm{I}\left(h_v, h_G\right) = \mathbb{E}_p\left[-\log\left(1+\exp\left(-\mathcal{D}(h_v, h_G)\right)\right)\right] - \mathbb{E}_{p\times\tilde{p}}\left[-\log\left(1+\exp\left(\mathcal{D}(h_v, h_{\tilde{G})})\right)\right)\right] \tag{2}$$

$\mathcal{D}(.,.) : \mathbb{R}^{d_h} \times \mathbb{R}^{d_h} \longmapsto \mathbb{R}$ is a discriminator that takes in a node and a graph representation and scores the agreement between them and is implemented as $\mathcal{D}(h_v, h_g) = h_n.h_g^T$. We provide the positive samples from the joint distribution (p) and the negative samples from the product of marginals $p \times \tilde{p}$, and optimize the model parameters with respect to the objective using mini-batch stochastic gradient descent. We found that regularizing the encoders by randomly alternating between training the base and augmentation encoders helps the base encoder to generalize better. For this purpose, we train the policy and the augmentation heads at each step, but we sample from a Bernoulli to decide whether to update the weights of the base or augmentation encoders . The training process is summarized in Algorithm 1.

Algorithm 1: End-to-end training algorithm.

Input: Augmentation heads $\tau_i, \tau_j \in \mathcal{T}$, policy r, graph encoders g_θ and g_ω, MI estimator $\mathcal{I}$, loss $\mathcal{L}$, and graphs $G \in \mathcal{G}$

for *sampled batch* $\mathcal{B} = \{G_k\}_{k=1}^N \in \mathcal{G}$ **do**

- $\{(\mathbf{H}_{v_k}, h_{G_k})\}_{k=1}^N = g_\omega\left(G_k\right)$ // Compute the augmentation encodings
- $i, j, p_i, p_j = r_\mu(\{(\mathbf{H}_{v_k}, h_{G_k})\}_{k=1}^N)$ // Sample the policy
- **for** $k = 1$ *to* N **do**
  - $G_k^i = \tau_{\phi_i}(G_k, \mathbf{H}_{v_k}, h_{G_k})$ // Sample the first view
  - $\mathbf{H}_{v_k}^i, h_{G_k}^i = g_\theta(G_k^i)$ // Compute the first view encodings
  - $G_k^j = \tau_{\phi_j}(G_k, \mathbf{H}_{v_k}, h_{G_k})$ // Sample the second view
  - $\mathbf{H}_{v_k}^j, h_{G_k}^j = g_\theta(G_k^j)$ // Compute the second view encodings
  - $h_{G_k}^i = h_{G_k}^i \times p_i$ // Scale the encodings by their probabilities
  - $h_{G_k}^j = h_{G_k}^j \times p_j$
- **end**
- **for** $k = 1$ *to* N *and* $k' = 1$ *to* N **do**
  - $s_{k,k'}^i = \mathcal{I}\left(h_{G_k}^i, \mathbf{H}_{v_{k'}}^j\right), s_{k,k'}^j = \mathcal{I}\left(h_{G_k}^j, \mathbf{H}_{v_{k'}}^i\right)$ // Compute pairwise similarity:
- **end**
- $\bigtriangledown_{\omega,\mu,\phi_i,\phi_j,\theta} \frac{1}{N^2} \sum_{k=1}^N \sum_{k'=1}^N \left[\mathcal{L}\left(s_{k,k'}^i\right) + \mathcal{L}\left(s_{k,k'}^j\right)\right]$ // compute gradients and update

end

4 EXPERIMENTAL RESULTS

4.1 UNSUPERVISED REPRESENTATION LEARNING EVALUATION

We evaluate LG2AR under the linear evaluation protocol on both node-level and graph-level classification benchmarks where unsupervised models first encode the graphs, and then the encoding are fed into a down-stream linear classifier without fine-tuning the encoder. For graph classification benchmarks, we follow GraphCL and use eight datasets from TUDataset (Morris et al., 2020) and

Table 1: Mean graph classification accuracy over 10 runs under linear evaluation protocol.

	Method	Mutag	Proteins	IMDB-B	IMDB-M	Reddit-B	Reddit-M
Sup.	GIN	89.4±5.6	76.2±2.8	75.1±5.1	52.3±2.8	92.4±2.5	57.6±1.5
	GAT	89.4±6.1	74.7±4.0	70.5±2.3	47.8±3.1	85.2±3.3	45.9 ±0.1
Kernel	SP	85.2±2.4	–	55.6±0.2	38.0±0.3	64.1±0.1	39.6±0.2
	GK	81.7±2.1	–	65.9±1.0	43.9±0.4	77.3±0.2	41.0±0.2
	WL	80.7±3.0	72.9±0.6	72.3±3.4	47.0±0.5	68.8±0.4	46.1±0.2
	DGK	87.4±2.7	73.3±0.8	67.0±0.6	44.6±0.5	78.0±0.4	41.3±0.2
	MLG	87.9±1.6	–	66.6±0.3	41.2±0.0	–	–
Rnd Walk	RandomWalk	83.7±1.5	–	50.7±0.3	34.7±0.2	–	–
	Node2Vec	72.6±10.2	57.5±3.6	–	–	–	–
	Sub2Vec	61.1±15.8	53.0±5.6	55.3±1.5	36.7±0.8	71.5±0.4	36.7±0.4
	Graph2Vec	83.2±9.6	73.3±2.1	71.1±0.5	50.4±0.9	75.8±1.0	47.9±0.3
Unsupervised	InfoGraph	89.0±1.1	74.4±0.3	73.0±0.9	49.7±0.5	82.5±1.4	53.5±1.0
	GraphCL	86.8±1.4	74.4±0.5	71.1±0.4	48.5 ± 0.6	89.5±0.8	56.0±0.3
	ADGCL	89.7±1.0	73.8±0.5	71.6±1.0	49.9±0.7	85.5±0.8	54.9±0.4
	JOAO	87.7±0.8	74.6±0.4	70.8±0.3	–	86.4±1.5	56.0±0.3
	LG2AR + GRU (Ours)	**90.0±0.6**	**75.0±0.5**	**74.5±0.6**	**51.9±0.3**	91.8±0.4	**56.3±0.2**
	LG2AR + DeepSet (Ours)	88.9±0.6	74.8±0.5	74.1±0.2	51.2±0.3	91.6±0.1	56.0±0.2
	LG2AR + Random (Ours)	88.6±0.5	74.7±0.5	73.8±0.3	51.5±0.3	**92.2±0.1**	56.2±0.2

for the node classification, we follow GCA, and use seven datasets from (Mernyei & Cangea, 2020; Shchur et al., 2018). For fair comparisons, we closely follow the linear evaluation protocol from previous unsupervised works. For evaluating graph classification, we report the mean 10-fold cross validation accuracy with standard deviation after ten runs with a down-stream linear SVM classifier, and for node classification evaluation, we report the mean accuracy of twenty runs over different data splits with a down-stream single layer linear classifier. For details on the evaluation protocol see Appendix A.2 and for implementation details and hyper-parameter settings see Appendix A.3.

For both tasks, we train three variants of our framework denoted as LG2AR-GRU, LG2AR-DeepSet, and LG2AR-Random where each variant indicates its policy instantiation, i.e., GRU, deep set, and random (sampling views from uniform distribution) policies. For graph classification benchmarks, we compare the LG2AR with *two supervised baselines*: GIN and Graph Attention Network (GAT) (Veličković et al., 2018), *five graph kernel methods* including Shortest Path kernel (SP) (Borgwardt & Kriegel, 2005), Graphlet Kernel (GK) (Shervashidze et al., 2009), Weisfeiler-Lehman sub-tree kernel (WL) (Shervashidze et al., 2011), Deep Graph Kernels (DGK) (Yanardag & Vishwana, 2015), and Multi-scale Laplacian Graph kernel (MLG) (Kondor & Pan, 2016), and *four random walk methods* including Random Walk (Gärtner et al., 2003), Node2Vec (Grover & Leskovec, 2016), Sub2Vec (Adhikari et al., 2018), Graph2Vec (Narayanan et al., 2017). We also compare the results with state-of-the-art deep contrastive models including InfoGraph, GraphCL, JOAO, and ADGCL. For node classification benchmarks, we compare our results with random walk methods including DeepWalk with and without concatenating node features, and Node2Vec. We also compare the results with deep learning methods including Graph Autoencoders (GAE, VGAE) (Kipf & Welling, 2016), Graphical Mutual Information Maximization (GMI) (Peng et al., 2020), MVGRL, DGI, and GCA.

The results for graph classification are reported in Tables 1 and 7 (Appendix) and for the node classification are reported in Tables 2 and 7 (Appendix). The reported performance for other models are from the corresponding papers under the same experiment setting. As shown for graph classification, LG2AR achieves state-of-the-art results with respect to unsupervised models across all eight benchmarks. For example, on IMDB-Multi and COLLAB datasets we achieve 2.2% and 6.4% absolute improvement over previous state-of-the-art. We also observe that LG2AR narrows the gap with the best performing supervised baselines even surpassing them on MUTAG dataset. Moreover, for node classification, we observe that we achieve state-of-the-art results with respect to previous unsupervised models in five out of seven benchmarks. For example, we achieve 4.7%, 1.6%, and 1.7 % absolute improvement on the PubMed, Amazon-Photo, and Amazon-Computer datasets. When compared to supervised baselines, we outperform or perform equally good across the benchmarks.

Table 2: Mean node classification accuracy over 20 runs under linear evaluation protocol.

	Method	WikiCS	Amz-Comp	Amz-Photo	Coauthor-CS	Coauthor-Phy
Sup.	GIN	75.9±0.7	87.5±0.9	90.9±0.5	91.4±0.2	95.2±0.1
	GAT	77.7±0.1	86.9±0.3	92.6±0.4	92.3±0.2	95.5±0.2
Rnd Walk	Raw Features	71.98±0.0	73.8±0.0	78.5±0.0	90.4±0.0	93.6±0.0
	Node2Vec	71.8±0.1	84.4±0.1	89.7±0.1	85.1±0.0	91.2±0.0
	DeepWalk	74.4±0.1	85.7±0.1	89.4±0.1	84.6±0.2	91.8±0.2
	DeepWalk + Feat	77.2±0.0	86.3±0.1	90.1±0.1	87.7±0.0	94.9±0.1
Unsupervised	GAE	70.2±0.0	85.3±0.2	91.6±0.1	90.0±0.7	94.92±0.1
	VGAE	75.6±0.2	86.4±0.2	92.2±0.1	92.1±0.1	94.5±0.0
	DGI	75.4±0.1	84.0±0.5	91.6±0.2	92.2±0.6	94.5±0.5
	GMI	74.9±0.1	82.2±0.3	90.7±0.2	OOM	OOM
	MVGRL	77.5±0.0	87.5±0.1	91.7±0.1	92.1±0.1	95.3±0.0
	GRACE	**80.1±0.5**	89.5±0.4	92.8±0.5	91.1±0.2	OOM
	BGRL	80.0±0.1	**90.3±0.2**	93.2±0.3	93.3±0.1	**95.7±0.1**
	GCA	78.4±0.1	87.9±0.3	92.2±0.2	93.1±0.0	**95.7±0.0**
	LG2AR + GRU (Ours)	77.8 ±0.5	89.3±0.4	**94.1±0.4**	**93.6±0.3**	**95.7±0.2**
	LG2AR + DeepSet (Ours)	76.2±0.9	89.6±0.3	92.6±0.5	92.4±0.3	95.5±0.1
	LG2AR + Random (Ours)	76.2±0.7	88.8±0.4	92.4±0.6	92.3±0.2	95.4±0.1

Table 3: Mean 10-fold accuracy of semi-supervised learning with 10% label rate.

	Proteins	DD	COLLAB	Reddit-B	Reddit-M
GAE	70.5±0.2	74.5±0.7	75.1±0.2	87.7±0.4	53.6±0.1
Infomax	72.3±0.4	75.8±0.3	73.8±0.3	88.7±0.9	53.6±0.3
GraphCL	74.2±0.3	76.2±1.4	74.2±0.2	89.1±0.2	52.6±0.5
JOAO	73.3±0.5	75.8±0.7	75.5±0.2	88.8±0.7	52.7±0.3
ADGCL	74.0±0.5	77.9±0.7	75.8±0.3	90.1±0.2	53.5±0.3
LG2AR (Ours)	**76.1±0.4**	**79.8±0.3**	**78.4±0.4**	**92.3±0.5**	**57.2±0.6**

4.2 Semi-Supervised Learning Evaluation

Furthermore, we evaluated LG2AR in a semi-supervised learning setting on graph classification benchmarks. We follow the experimental protocol introduced in GraphCL and pre-train the encoder in an unsupervised fashion and fine-tune it only on 10% of the labeled data. The results reported in Table 3 show that LG2AR achieves state-of-the-art results compared to previous unsupervised models across all five benchmarks. Most notably, LG2AR achieves an absolute accuracy gain of 3.6% and 2.6% over Collab and Reddit-Multi benchmarks.

4.3 Analysis of the optimization framework

In this section, we discuss how LG2AR cannot fall into trivial solutions and compare its optimization with a few notable works. SimSiam (Chen & He, 2021) states that collapsing, i.e. minimum possible loss with constant outputs, cannot be prevented by solely relying on architecture designs such as batch normalization. By designing multiple experiments they concluded that the non-collapsing behaviour of SimSiam still remains an empirical observation. Inspired by MoCo (He et al., 2020) and BYOL (Grill et al., 2020), AutoMix (Liu et al., 2021b) avoids a nested optimization by decoupling its momentum pipeline. LG2AR does not require a momentum pipeline because it is based on contrasting local-global information. Even if the augmentation distribution ends being a Dirac peaking on any of the five augmentations, which we did not observe, a collapse cannot happen and a single level optimization is sufficient. If solely any of the node, sub-graph or edge augmentations are sampled, the inductive biases that we injected and discussed, force the model to at least select a sub part of the graph as an augmented view, hence avoiding the trivial solution of a graph with only one node and no edges. If only identity or feature augmentation is sampled, LG2AR would reduce to a single level optimization method such as InfoGraph where there are no augmentations. JOAO,

re-frames the auto-augmentation on graphs to a min-max optimization. Inspired from robustness and adversarial learning literature, they employ Alternating Gradient Descent (Wang et al., 2019a) to design an approach for learning the augmentation distribution and the encoder parameters in a bi-level optimization setting. We are using the gumbel-softmax trick to sample from the augmentation distribution and let the gradient flow through discrete parameters. Our algorithm alternates between updating the encoder and augmentation parameters by tossing a fair coin in each iteration. Alternating gradients between modules in unsupervised learning is shown to be efficient in avoiding the trivial solutions (Caron et al., 2018; Hassani & Haley, 2019a; Khasahmadi et al., 2020b). With these bag of tricks, LG2AR avoid collapsing to trivial solutions and solving a min-max problem that needs an inefficient bi-level optimization. Moreover, Figure 3 in the Appendix is showing a stable training trajectory for multiple datasets.

4.4 Ablation Study

Effect of the Policy. To investigate the effect of policy, we trained the models with three variants of the policy including GRU, deep set, and random policies. As shown in Tables 1-2, the GRU-based policy outperforms or performs equally well on 12 out of 15 benchmarks whereas the random policy outperforms the other variants in only 1 out of 15 datasets, indicating the importance of learning the view sampling policy. Also, in order to probe what the policy module is learning, we computed the normalized frequency of the sampled augmentations by the GRU-based policy during the training. The frequencies for two graph classification benchmarks (MUTAG and Reddit-binary) and two node classification benchmarks (Coauthor-CS and PubMed) are shown in Figure 2. We observe the following: (1) The policy is learning different distributions over augmentations for each benchmark suggesting that it is adapting to the given datasets. (2) In node classification benchmarks, because we already induce sub-graphs to transform them to inductive tasks, we observe that the policy inclines towards sampling the identity augmentation more frequently which is essentially a sub-graph of the original graph. (3) We observe that regardless of the task, edge perturbation and sub-graph inducing are the two most commonly sampled augmentations. This confirms the observation that sub-graphs are generally beneficial across datasets (You et al., 2020). (4) We observe that for node classification benchmarks, the probability of sampling feature masking augmentation is positively correlated with the initial node feature dimension.

Effect of the Augmentations. To investigate the effect of the augmentations, we run the experiments with single augmentation, and structural vs feature space augmentations. The results shown in Table 4 indicate that: (1) using our single augmentations performs on par or better than baselines, (2) generally structural augmentations contribute more than feature space augmentations, and (3) all augmentations are contributing to the final performance which suggests that the model should use all augmentations while learning the sampling frequencies.

Effect of the Augmentation Heads. To investigate the effect of the augmentations heads without benefiting from the policy, we compare our framework when trained with a random policy with GraphCL for graph classification benchmarks. Both LG2AR-Random and GraphCL sample the augmentations from a uniform distribution, where the former learns distributions over the augmentation parameters and the latter randomly samples those. As shown in Tables 1 and 7, in 8 out of 8 benchmarks LG2AR-Random outperforms GraphCL. For instance, we see an absolute improvement of 2.7% accuracy on Reddit-Binary. This implies that learning the augmentations contributes to the performance boost on the graph classification benchmarks, and when combined with the policy learning, results are further improved. We see less of this effect in transductive tasks suggesting that the policy learning is playing a more important rule in node classification benchmarks. One reason for this may be the fact that GCA unlike GraphCL uses strong topological inductive biases to select the augmentation parameters.

Effect of the Mutual Information Estimator and Discriminator. We investigated four MI estimators including: noise-contrastive estimation (NCE) Gutmann & Hyvärinen (2010); Oord et al. (2018), Jensen-Shannon (JSD) estimator following formulation in Nowozin et al. (2016), normalized temperature-scaled cross-entropy (NT-Xent) Chen et al. (2020b), and Donsker-Varadhan (DV) representation of the KL-divergence Donsker & Varadhan (1975). The results shown in Table 4 suggests that JSD and NT-Xent perform better compared to the other estimators. We also investigated the effect of discriminator by training the model using four variants including dot product, cosine

Table 4: Effects of Mutual Information Estimator, Discriminator, and augmentations.

		Proteins	DD	COLLAB	IMDB-B	IMDB-M	Reddit-B	Reddit-M
Loss	**JSD**	75.0±0.5	**79.1±0.3**	**77.8±0.2**	74.5±0.6	**51.9±0.3**	**91.8±0.4**	56.3±0.2
	NCE	74.4±0.6	78.4±0.5	77.1±0.3	73.9±0.4	51.2±0.6	90.8±0.4	56.4±0.2
	NT-Xent	**75.1±0.6**	78.7±0.5	77.5±0.4	**74.6±0.7**	51.5±0.4	91.3±0.5	**56.7±0.4**
	DV	74.3±0.4	78.2±0.4	77.1±0.3	73.5±0.6	50.7±0.5	91.1±0.5	55.2±0.3
Discrimi.	**Dot Product**	75.0±0.5	79.1±0.3	**77.8±0.2**	74.5±0.6	51.9±0.3	**91.8±0.4**	56.3±0.2
	Cosine	75.4±0.4	79.2±0.3	77.4±0.3	74.3±0.7	51.6±0.4	92.1±0.3	56.4±0.2
	Bilinear	74.6±0.4	78.7±0.4	77.5±0.4	73.8±0.6	51.0±0.5	90.4±0.5	55.4±0.4
	MLP	**75.3±0.6**	**79.6±0.5**	77.5±0.5	**74.7±0.3**	**60.4±0.6**	91.7±0.5	**56.8±0.5**
Augmentations	**All**	**75.0±0.5**	**79.1±0.3**	**77.8±0.2**	**74.5±0.6**	**51.9±0.3**	**91.8±0.4**	**56.3±0.2**
	Structure	74.6±0.5	79.1±0.2	77.3±0.3	74.1±0.5	51.8±0.3	91.1±0.3	56.2±0.2
	Feature	74.2±0.4	77.9±0.3	76.7±0.3	73.7±0.2	51.3±0.3	89.1±0.4	55.3±0.3
	Node	74.1±0.5	78.1±0.2	76.6±0.2	73.6±0.3	51.2±0.3	89.4±0.3	55.1±0.2
	Edge	74.3±0.5	77.7±0.2	76.9±0.2	73.7±0.3	50.9±0.3	89.5±0.3	55.2±0.2
	SubGraph	73.9±0.5	78.4±0.3	76.4±0.2	73.1±0.3	51.2±0.4	89.4±0.2	55.3±0.3

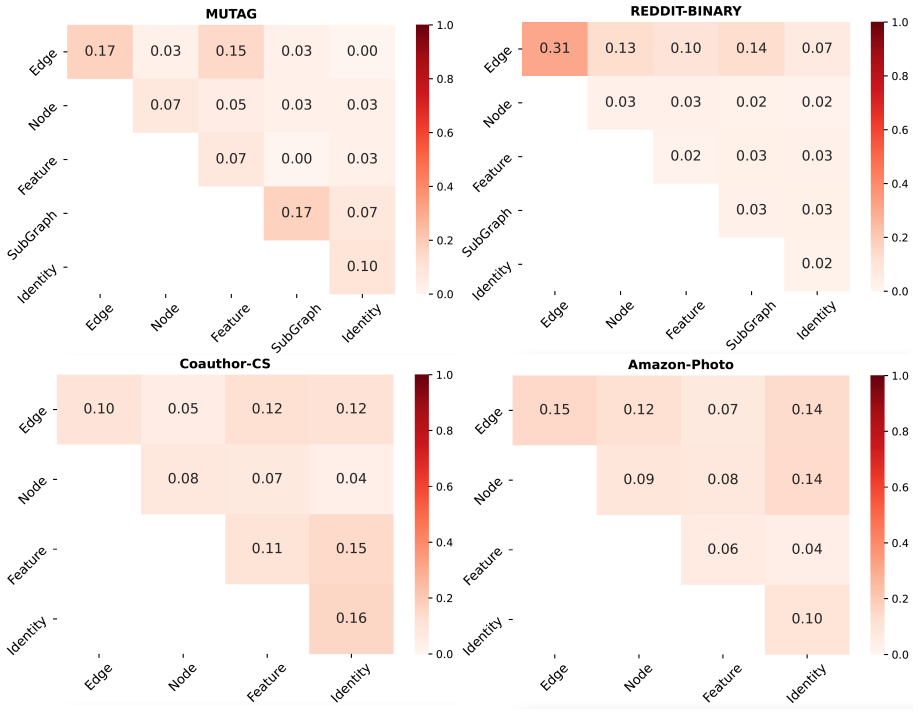

Figure 2: The normalized frequency of augmentation selection by the GRU-based policy for two graph benchmarks (top row) and two node benchmarks (bottom row).

distance, Bilinear, and MLP discriminators. The results shown in Table 4 suggests that discriminator instantiated as an MLP performs better across the datasets.

5 Conclusion

We introduced LG2AR, and end-to-end framework to automate graph contrastive learning. The proposed framework learns the augmentations, view selection policy, and the encoders end-to-end without requiring ad-hoc trial-and-error processes for devising the augmentations for each and every dataset. Experimental results showed that LG2AR achieves state-of-the-art results on 8 out of 8

graph classification benchmarks, and 6 out of 7 node classification benchmarks compared to the previous unsupervised methods. The results also suggest that LG2AR narrows the gap with its supervised counterparts. Furthermore, the results suggest that both learning the policy and learning the augmentations contributes to the performance. In future work, we are planning to investigate large pre-training and transfer learning capabilities of the proposed method.

REFERENCES

Bijaya Adhikari, Yao Zhang, Naren Ramakrishnan, and B Aditya Prakash. Sub2vec: Feature learning for subgraphs. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pp. 170–182, 2018.

Karsten M Borgwardt and Hans-Peter Kriegel. Shortest-path kernels on graphs. In *International Conference on Data Mining*, 2005.

Mathilde Caron, Piotr Bojanowski, Armand Joulin, and Matthijs Douze. Deep clustering for unsupervised learning of visual features. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 132–149, 2018.

Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, and Xu Sun. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. In *AAAI Conference on Artificial Intelligence*, pp. 3438–3445, 2020a.

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

Xinlei Chen and Kaiming He. Exploring simple siamese representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 15750–15758, 2021.

Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Conference on Empirical Methods in Natural Language Processing*, pp. 1724–1734, 2014.

Ekin D. Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V. Le. Autoaugment: Learning augmentation strategies from data. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2019.

Paul D Dobson and Andrew J Doig. Distinguishing enzyme structures from non-enzymes without alignments. *Journal of molecular biology*, 330(4):771–783, 2003.

Monroe D Donsker and SR Srinivasa Varadhan. Asymptotic evaluation of certain markov process expectations for large time. *Communications on Pure and Applied Mathematics*, 28(1):1–47, 1975.

David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in Neural Information Processing Systems*, pp. 2224–2232, 2015.

Steven Feng, Varun Prashant Gangal, Jason Wei, Soroush Vosoughi, Sarath Chandar, Teruko Mitamura, and Eduard Hovy. A survey on data augmentation approaches for nlp. 2021.

Alberto Garcia Duran and Mathias Niepert. Learning graph representations with embedding propagation. In *Advances in Neural Information Processing Systems*, pp. 5119–5130. 2017.

Thomas Gärtner, Peter Flach, and Stefan Wrobel. On graph kernels: Hardness results and efficient alternatives. In *Learning theory and kernel machines*, pp. 129–143. 2003.

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International Conference on Machine Learning*, pp. 1263–1272, 2017.

Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Guo, Mohammad Gheshlaghi Azar, Bilal Piot, koray kavukcuoglu, Remi Munos, and Michal Valko. Bootstrap your own latent - a new approach to self-supervised learning. In *Advances in Neural Information Processing Systems*, pp. 21271–21284, 2020.

Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *International Conference on Knowledge Discovery and Data Mining*, pp. 855–864, 2016.

Michael Gutmann and Aapo Hyvärinen. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *International Conference on Artificial Intelligence and Statistics*, pp. 297–304, 2010.

Kaveh Hassani and Mike Haley. Unsupervised multi-task feature learning on point clouds. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 8160–8171, 2019a.

Kaveh Hassani and Mike Haley. Unsupervised multi-task feature learning on point clouds. In *International Conference on Computer Vision*, pp. 8160–8171, 2019b.

Kaveh Hassani and Amir Hosein Khasahmadi. Contrastive multi-view representation learning on graphs. In *International Conference on Machine Learning*, pp. 4116–4126, 2020.

Ryuichiro Hataya, Jan Zdenek, Kazuki Yoshizoe, and Hideki Nakayama. Faster autoaugment: Learning augmentation strategies using backpropagation. In *European Conference on Computer Vision*, pp. 1–16, 2020.

Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9729–9738, 2020.

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. In *International Conference on Learning Representations*, 2019.

Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. Strategies for pre-training graph neural networks. In *International Conference on Learning Representations*, 2020.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In *International Conference on Learning Representations*, 2016.

Amir Hosein Khasahmadi, Kaveh Hassani, Parsa Moradi, Leo Lee, and Quaid Morris. Memory-based graph networks. In *International Conference on Learning Representations*, 2020a.

Amir Hosein Khasahmadi, Kaveh Hassani, Parsa Moradi, Leo Lee, and Quaid Morris. Memory-based graph networks. In *International Conference on Learning Representations*, 2020b. URL `https://openreview.net/forum?id=r1laNeBYPB`.

Thomas N Kipf and Max Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017.

Risi Kondor and Horace Pan. The multiscale laplacian graph kernel. In *Advances in Neural Information Processing Systems*, pp. 2990–2998. 2016.

Kezhi Kong, Guohao Li, Mucong Ding, Zuxuan Wu, Chen Zhu, Bernard Ghanem, Gavin Taylor, and Tom Goldstein. Flag: Adversarial data augmentation for graph neural networks. *arXiv preprint arXiv:2010.09891*, 2020.

Wouter Kool, Herke Van Hoof, and Max Welling. Stochastic beams and where to find them: The gumbel-top-k trick for sampling sequences without replacement. In *International Conference on Machine Learning*, pp. 3499–3508, 2019.

Nils Kriege and Petra Mutzel. Subgraph matching kernels for attributed graphs. In *International Conference on Machine Learning*, pp. 291–298, 2012.

Yonggang Li, Guosheng Hu, Yongtao Wang, Timothy Hospedales, Neil M. Robertson, and Yongxin Yang. Differentiable automatic data augmentation. In *European Conference on Computer Vision*, pp. 580–595, 2020.

Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. In *International Conference on Learning Representations*, 2015.

Sungbin Lim, Ildoo Kim, Taesup Kim, Chiheon Kim, and Sungwoong Kim. Fast autoaugment. In *Advances in Neural Information Processing Systems*, 2019.

Yixin Liu, Shirui Pan, Ming Jin, Chuan Zhou, Feng Xia, and Philip S Yu. Graph self-supervised learning: A survey. *arXiv preprint arXiv:2103.00111*, 2021a.

Zicheng Liu, Siyuan Li, Di Wu, Zhiyuan Chen, Lirong Wu, Jianzhu Guo, and Stan Z Li. Automix: Unveiling the power of mixup. *arXiv preprint arXiv:2103.13027*, 2021b.

Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.

Péter Mernyei and Cătălina Cangea. Wiki-cs: A wikipedia-based benchmark for graph neural networks. *arXiv preprint arXiv:2007.02901*, 2020.

Christopher Morris, Nils M. Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. Tudataset: A collection of benchmark datasets for learning with graphs. In *ICML 2020 Workshop on Graph Representation Learning and Beyond (GRL+ 2020)*, 2020. URL `www.graphlearning.io`.

Annamalai Narayanan, Mahinthan Chandramohan, Rajasekar Venkatesan, Lihui Chen, Yang Liu, and Shantanu Jaiswal. graph2vec: Learning distributed representations of graphs. *arXiv preprint arXiv:1707.05005*, 2017.

Sebastian Nowozin, Botond Cseke, and Ryota Tomioka. f-gan: Training generative neural samplers using variational divergence minimization. In *Advances in Neural Information Processing Systems*, pp. 271–279. 2016.

Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

Shirui Pan, Ruiqi Hu, Guodong Long, Jing Jiang, Lina Yao, and Chengqi Zhang. Adversarially regularized graph autoencoder for graph embedding. In *International Joint Conference on Artificial Intelligence*, pp. 2609–2615, 2018.

Jiwoong Park, Minsik Lee, Hyung Jin Chang, Kyuewang Lee, and Jin Young Choi. Symmetric graph convolutional autoencoder for unsupervised graph representation learning. In *International Conference on Computer Vision*, pp. 6519–6528, 2019.

Zhen Peng, Wenbing Huang, Minnan Luo, Qinghua Zheng, Yu Rong, Tingyang Xu, and Junzhou Huang. Graph representation learning via graphical mutual information maximization. In *Proceedings of The Web Conference*, pp. 259–270, 2020.

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*, pp. 1150–1160. 2020.

Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. Dropedge: Towards deep graph convolutional networks on node classification. In *International Conference on Learning Representations*, 2020.

Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter Battaglia. Learning to simulate complex physics with graph networks. In *Proceedings of the 37th International Conference on Machine Learning*, pp. 8459–8468, 2020.

Michael Sejr Schlichtkrull, Nicola De Cao, and Ivan Titov. Interpreting graph neural networks for {nlp} with differentiable edge masking. In *International Conference on Learning Representations*, 2021.

Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI Magazine*, 29(3):93–93, 2008.

Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*, 2018.

Nino Shervashidze, SVN Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten Borgwardt. Efficient graphlet kernels for large graph comparison. In *Artificial Intelligence and Statistics*, pp. 488–495, 2009.

Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12:2539–2561, 2011.

Connor Shorten and Taghi M Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1):1–48, 2019.

Fan-Yun Sun, Jordan Hoffman, Vikas Verma, and Jian Tang. Infograph: Unsupervised and semi-supervised graph-level representation learning via mutual information maximization. In *International Conference on Learning Representations*, 2020.

Susheel Suresh, Pan Li, Cong Hao, and Jennifer Neville. Adversarial graph augmentation to improve graph contrastive learning. *arXiv preprint arXiv:2106.05819*, 2021.

Shantanu Thakoor, Corentin Tallec, Mohammad Gheshlaghi Azar, Rémi Munos, Petar Veličković, and Michal Valko. Bootstrapped representation learning on graphs. *arXiv preprint arXiv:2102.06514*, 2021.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018.

Petar Veličković, William Fedus, William L. Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. Deep graph infomax. In *International Conference on Learning Representations*, 2019.

Salvatore Vivona and Kaveh Hassani. Relational graph representation learning for open-domain question answering. *Advances in Neural Information Processing Systems, Graph Representation Learning Workshop*, 2019.

Jingkang Wang, Tianyun Zhang, Sijia Liu, Pin-Yu Chen, Jiacen Xu, Makan Fardad, and Bo Li. Towards a unified min-max framework for adversarial exploration and robustness. *arXiv preprint arXiv:1906.03563*, 2019a.

Tingwu Wang, Yuhao Zhou, Sanja Fidler, and Jimmy Ba. Neural graph evolution: Automatic robot design. In *International Conference on Learning Representations*, 2019b.

Lirong Wu, Haitao Lin, Zhangyang Gao, Cheng Tan, Stan Li, et al. Self-supervised on graphs: Contrastive, generative, or predictive. *arXiv preprint arXiv:2105.07342*, 2021.

Yaochen Xie, Zhao Xu, Jingtun Zhang, Zhengyang Wang, and Shuiwang Ji. Self-supervised learning of graph neural networks: A unified review. *arXiv preprint arXiv:2102.10757*, 2021.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019.

Pinar Yanardag and S.V.N. Vishwana. Deep graph kernels. In *International Conference on Knowledge Discovery and Data Mining*, pp. 1365–1374, 2015.

Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. Graph contrastive learning with augmentations. *Advances in Neural Information Processing Systems*, 33: 5812–5823, 2020.

Yuning You, Tianlong Chen, Yang Shen, and Zhangyang Wang. Graph contrastive learning automated. *arXiv preprint arXiv:2106.07594*, 2021.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, 2017.

Yingxue Zhang, Soumyasundar Pal, Mark Coates, and Deniz Ustebay. Bayesian graph convolutional neural networks for semi-supervised classification. In *AAAI Conference on Artificial Intelligence*, pp. 5829–5836, 2019.

Tong Zhao, Yozen Liu, Leonardo Neves, Oliver Woodford, Meng Jiang, and Neil Shah. Data augmentation for graph neural networks. In *AAAI Conference on Artificial Intelligence*, pp. 11015–11023, 2021.

Yanqiao Zhu, Yichen Xu, Feng Yu, Qiang Liu, Shu Wu, and Liang Wang. Deep graph contrastive representation learning. *arXiv preprint arXiv:2006.04131*, 2020.

Yanqiao Zhu, Yichen Xu, Feng Yu, Qiang Liu, Shu Wu, and Liang Wang. Graph contrastive learning with adaptive augmentation. In *Proceedings of the Web Conference*, pp. 2069–2080, 2021.

A APPENDIX

A.1 BENCHMARKS

We use seven node classification and eight graph classification benchmarks reported by previous state-of-the-art methods. For node classification benchmarks, we follow GCA (Zhu et al., 2021) and use Wiki-CS (Mernyei & Cangea, 2020) which is a computer science subset of the Wikipedia, Amazon-Computers and Amazon-Photo (Shchur et al., 2018) which are networks of co-purchase relationships constructed from Amazon, Coauthor-CS and Coauthor-Physics (Shchur et al., 2018) which are two academic networks containing co-authorship graphs, and two other citation networks, Cora and Pubmed Sen et al. (2008). For graph classification benchmarks, we follow GraphCL (You et al., 2020) and use benchmarks from TUDatasets (Morris et al., 2020). We use Proteins and DD (Dobson & Doig, 2003) modeling neighborhoods in the amino-acid sequences and protein structures, respectively, MUTAG Kriege & Mutzel (2012) modeling compounds tested for carcinogenicity, COLLAB (Yanardag & Vishwana, 2015) derived from 3 public physics collaboration, Reddit-Binary and Reddit-Multi-5K (Yanardag & Vishwana, 2015) connecting users through responses in Reddit online discussions, and IMDB-Binary and IMDB-Multi Yanardag & Vishwana (2015) connecting actors/actresses based on movie appearances. The statistics of the graph and graph classification benchmarks are summarized in Tables 5 and 6, respectively.

Table 5: Statistics of graph classification benchmarks.

	Biology			*Social Networks*				
	MUTAG	**PROTEINS**	**DD**	**COLLAB**	**IMDB-B**	**IMDB-M**	**REDDIT-B**	**REDDIT-M**
|GRAPHS|	188	1113	1178	5000	1000	1500	2000	4999
|NODES|	17.93	39.06	284.32	74.49	19.77	13.00	429.63	508.52
|EDGES|	19.79	72.82	715.66	2457.78	96.53	65.94	497.75	594.87
|FEATURES|	7	4	89	0	0	0	0	0
|CLASSES|	2	2	2	3	2	3	2	5

Table 6: Statistics of node classification benchmarks.

	CORA	**PUBMED**	**WIKICS**	**AMZ-COMP**	**AMZ-PHOTO**	**COAU-CS**	**COAU-PHY**
|NODES|	3,327	19,717	11,701	13,752	7,650	18,333	34,493
|EDGES|	4,732	44,338	216,123	245,861	119,081	81,894	247,962
|FEATURES|	1,433	500	300	767	745	6,805	8,415
|CLASSES|	6	3	10	10	8	15	5

A.2 EVALUATION PROTOCOL DETAILS

For node classification, we follow (Veličković et al., 2019; Zhu et al., 2021) where we train the model with the contrastive method, and then use the resulting embeddings to train and test a simple logistic regression classifier. We train the model for twenty runs over different data splits and report the mean accuracy with standard deviation. For fair evaluation and following GCA, we use a two-layer Graph Convolution Network (GCN) (Kipf & Welling, 2017) for the base encoder across all node classification benchmarks. In order to make the transductive node classification benchmarks compatible with our inductive framework, we sample $|\mathcal{B}|$ sub-graphs from the input graph around randomly selected nodes to emulate a batch of graphs, and then feed the batch to our framework. Because sub-graph augmentation occurs before our framework, we remove this augmentation from the policy and also remove its corresponding head. For graph classification, we follow (You et al., 2020; Sun et al., 2020) where we first use the contrastive loss to train the model and then report the best mean 10-fold cross validation accuracy with standard deviation after five runs. The classifier is a linear SVM trained using cross-validation on the training folds of the learned embeddings. Following (Sun et al., 2020), we use GIN layers for the base encoder and treat the number of layers as a hyper-parameter. We observed that contrasting graph-level representation achieves better results in graph classification benchmarks. Finally, we report more node and graph level evaluation results under linear evaluation protocol in Table 7.

Table 7: Mean graph and node classification accuracy under linear evaluation protocol.

Node			*Graph*		
Method	**Cora**	**PubMed**	**Method**	**Collab**	**DD**
DeepWalk	70.7±0.6	74.3±0.9	InfoGraph	70.7±1.1	72.9±1.8
GAE	71.5±0.4	72.1±0.5	GraphCL	71.4±1.2	78.6±0.4
VERSE	72.5±0.3	–	AD-GCL	73.3±0.6	75.1±0.4
DGI	82.3±0.6	76.8±0.6	JOAO	69.5±0.4	77.4±1.2
LG2AR + GRU (Ours)	**82.7±0.7**	81.0±0.6	LG2AR + GRU (Ours)	**77.8±0.2**	**79.1±0.3**
LG2AR + DeepSet (Ours)	80.8±1.0	**81.5±0.7**	LG2AR + DeepSet (Ours)	**77.8±0.2**	78.6±0.5
LG2AR + Random (Ours)	81.6±0.9	81.3±0.8	LG2AR + Random (Ours)	77.6±0.2	78.8±0.4

A.3 Implementation & Hyper-Parameter Selection

We implemented the experiments using PyTorch and used Pytorch Geometric library to implement the graph encoders. Each experiments was run on a single RTX 6000 GPU. We initialize the parameters using Xavier initialization and train the model using Adam optimizer. All our graph implementations are sparse and in Pytorch Geometric format. Therefore, in order to let the gradients back-propagate, we use edge weights computed from augmentation heads as a way to pass the gradients. Also, in order to let the gradients back-propagate to the policy module, we multiply the final graph encodings from the two views with the associated probability of each view computed by the policy.

For node graph classification benchmarks, following GCA, we fix the base encoder to a tow-layer GCN model with mean-pooling as the read-out function. We select the number of augmentation encoder layers from [1, 2, 3, 4, 5, 6], number of sub-graphs per batch from [4, 8, 12, 16, 32], hidden dimension from [128, 256, 512], learning rate from [1e-4, 1e-1], number of hops from [1, 2, 3, 4, 5, 6], temperature from [0.7, 1.4], node dropping ration from [0.6, 0.9], and the dropout from [0.0, 0.2]. The augmentation consists of GIN layers with three layer projection heads and a summation read-out function. Following DGI, we use a early-stopping with patience of 50 steps. Following GCA, we train the linear model for 300 epochs with the learning rate of 1e-2.

For graph classification benchmarks, following InfoGraph, we design the both base and augmentation encoders with GIN layers, dedicated three-layer projection heads for node and graph encodings, and a summation read-out function. We share the learning rate and the number of layers between the two encoders and select them from [1e-4, 1e-1] and [1, 2, 3, 4, 5, 6], respectively. We select the batch size from [32, 64, 128], hidden dimension from [128, 256, 512], number of epochs from [10, 20, 40, 60, 100, 200], learning rate from [1e-4, 1e-1], number of hops from [1, 2, 3, 4, 5, 6], temperature from [0.7, 1.4], node dropping ration from [0.6, 0.9], and the dropout from [0.0, 0.2]. We also follow InfoGraph for graph classification and choose the C parameter of the SVM from $[10^{-3}, 10^{-2}, ..., 10^{2}, 10^{3}]$. The selected hyper-parameters are shown in Table 9.

Algorithm 2: Node dropping head.

Input: Node and graph encodings $\mathbf{H}_v$ and h_g, graph $G = (\mathcal{V}, \mathcal{E}, \mathbf{X})$, Ratio μ
$p(\mathcal{V}) = \text{MLP}\left([\mathbf{H}_v \parallel h_g]\right)$
$\mathcal{V}' \leftarrow$ Sample-Top-K$(p(\mathcal{V}), \mu)$
$\mathcal{E}' \leftarrow \mathcal{E} \subseteq \mathcal{V}' \times \mathcal{V}'$
$\mathbf{X}' \leftarrow \mathbf{X}[\mathcal{V}']$
$\mathbf{W}_{\mathcal{E}} \leftarrow [p(v_i) + p(v_j) \quad \forall e_{ij} \in \mathcal{E}']$
$G' \leftarrow (\mathcal{V}', \mathcal{E}', \mathbf{X}', \mathbf{W}_{\mathcal{E}})$
return G'

Table 8: Selected hyper-parameters.

	Benchmark	|Hidden|	|Batch|	Epoch	|Layers|	Learning Rate	Temperature	|Hops|	Ratio	Dropout
Graph Benchmarks	**MUTAG**	256	128	20	6	0.001	1.27	5	0.75	0.10
	Proteins	256	64	40	3	0.0003	0.73	4	0.77	0.05
	DD	256	128	100	3	0.0008	1.05	1	0.77	0.15
	COLLAB	128	128	10	4	0.0003	1.05	2	0.73	0.05
	IMDB-B	256	128	200	2	0.0003	1.04	4	0.84	0.00
	IMDB-M	512	128	100	3	0.0002	1.00	3	0.83	0.20
	Reddit-B	128	64	20	6	0.001	1.17	4	0.89	0.05
	Reddit-M	128	128	10	6	0.0004	1.24	2	0.72	0.05
Node Benchmarks	**CORA**	512	12	NA	2	0.03	1.19	6	0.86	0.20
	PubMed	512	16	NA	3	0.001	1.37	4	0.85	0.00
	WikiCS	512	8	NA	2	0.0001	0.83	1	0.86	0.05
	Amz-Comp	512	16	NA	2	0.0001	1.17	1	0.84	0.05
	Amz-Photo	512	16	NA	3	0.0005	1.23	1	0.76	0.20
	Coau-CS	256	4	NA	2	0.003	1.38	2	0.78	0.10
	Coau-Phy	512	8	NA	5	0.001	1.07	3	0.80	0.05

Algorithm 3: Edge perturbation head.

Input: Node and graph encodings $\mathbf{H}_v$ and h_g, graph $G = (\mathcal{V}, \mathcal{E}, \mathbf{X})$, temperature t
$\mathcal{V}', \mathcal{E}', \mathbf{X}', \mathbf{W}_{\mathcal{E}} \leftarrow \emptyset$
$\bar{\mathcal{E}} =$ Sample-Negative-Edges($\mathcal{E}$)
for e_{ij} *to* $\bar{\mathcal{E}} \cup \mathcal{E}$ **do**
  $h_{e_{ij}} = \left[h_{v_i} + h_{v_j} \parallel \mathbb{1}_{\mathcal{E}}(e_{ij})\right]$
  $p(e_{ij}) =$ MLP($h_{e_{ij}}$)
  if *Bernoulli-Sample*($t, p(e_{ij})$) **then**
    $\mathcal{V}' \leftarrow \mathcal{V}' \cup \{v_i, v_j\}$
    $\mathcal{E}' \leftarrow \mathcal{E}' \cup \{e_{ij}\}$
    $\mathbf{X}' \leftarrow \mathbf{X}' \cup \{h_{v_i}, h_{v_j}\}$
    $\mathbf{W}_{\mathcal{E}} \leftarrow \mathbf{W}_{\mathcal{E}} \cup \{p(e_{ij})\}$
  end
  $G' \leftarrow (\mathcal{V}', \mathcal{E}', \mathbf{X}', \mathbf{W}_{\mathcal{E}})$
end
return G'

Algorithm 4: Sub-graph inducing head.

Input: Node and graph encodings $\mathbf{H}_v$ and h_g, graph $G = (\mathcal{V}, \mathcal{E}, \mathbf{X})$, Number of hops K
$p(\mathcal{V}) =$ MLP $([\mathbf{H}_v \parallel h_g])$
$v_{center} \leftarrow$ Sample-Categorical($p(\mathcal{V})$)
$\mathcal{V}', \mathcal{E}', \mathbf{X}' \leftarrow$ k-Hop-BFS(v_{center}, K)
$\mathcal{E}' \leftarrow \mathcal{E} \subseteq \mathcal{V}' \times \mathcal{V}'$
$\mathbf{W}_{\mathcal{E}} \leftarrow [p(v_i) + p(v_j) \quad \forall e_{ij} \in \mathcal{E}']$
$G' \leftarrow (\mathcal{V}', \mathcal{E}', \mathbf{X}', \mathbf{W}_{\mathcal{E}})$
return G'

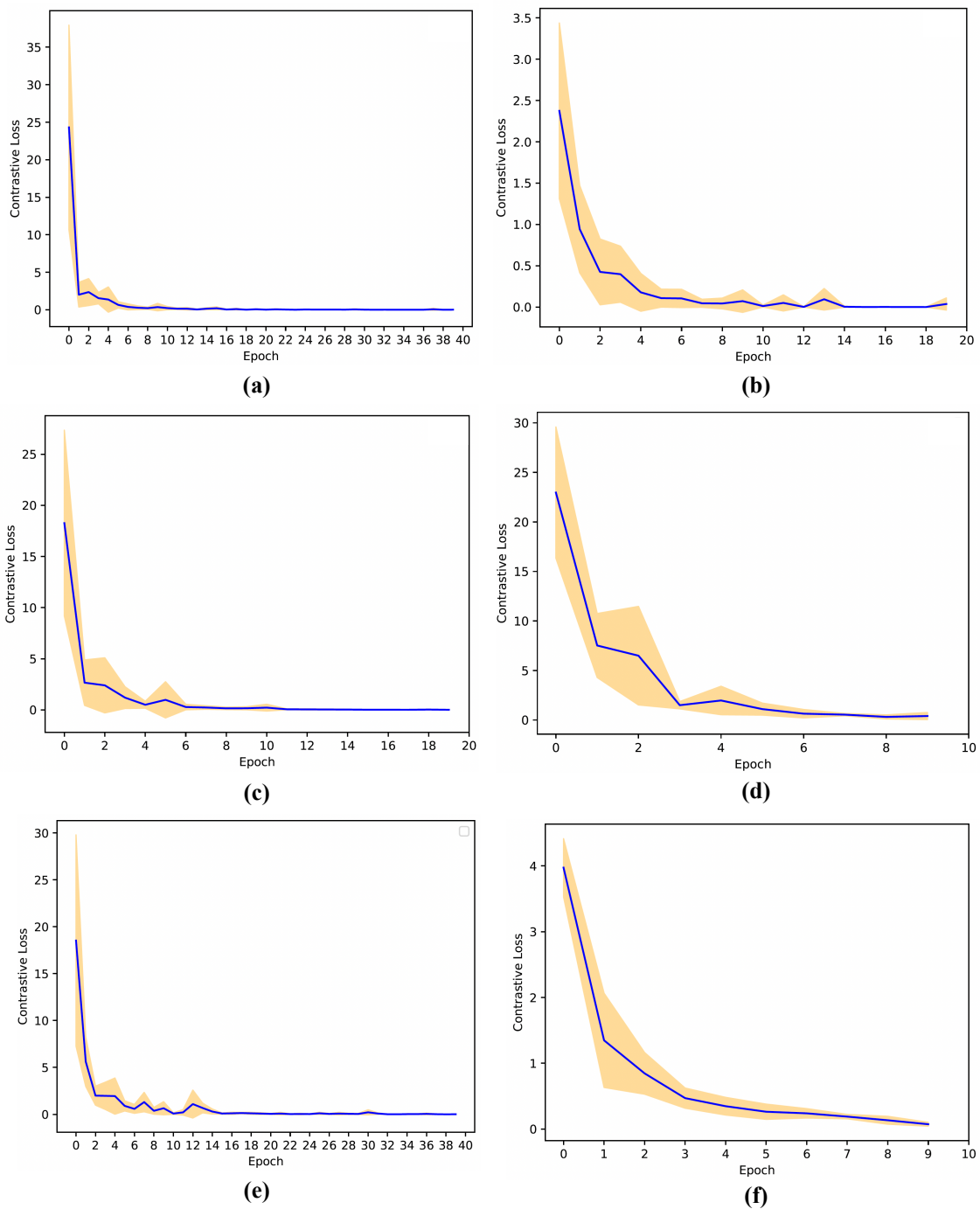

Figure 3: The evolution of the contrastive loss during the training averaged over ten runs for: (a) Proteins, (b) IMDB-Binary, (c) Reddit-Binary, (d) Reddit-Multi, (e) DD, and (f) Collab benchmarks.

Algorithm 5: Feature masking head.

Input: Node and graph encodings $\mathbf{H}_v$ and h_g, graph $G = (\mathcal{V}, \mathcal{E}, \mathbf{X})$, temperature t
$\mathbf{X}' \leftarrow \text{Linear}(\mathbf{X})$
$\mathbf{M} \leftarrow \text{Bernoulli-Sample}(\text{MLP}(\mathbf{H}_v), t)$
$\mathbf{X}' \leftarrow \mathbf{X}' \odot \mathbf{M}$
$G' \leftarrow (\mathcal{V}, \mathcal{E}, \mathbf{X}', \mathbf{1})$
return G'

Table 9: Mean time (seconds per epoch) and space (Gigabytes of GPU memory) for Infograph (single encoder) and LG2AR.

		MUTAG	Proteins	DD	COLLAB	IMDB-B	IMDB-M	Reddit-B	Reddit-M
Info	**Time** (Sec/epoch)	0.15	0.82	1.05	2.55	0.37	0.75	6.06	11.51
	Space (Gigabytes)	1.249	1.343	4.423	3.153	1.355	1.545	4.341	10.321
Ours	**Time** (Sec/epoch)	0.59	2.06	3.81	12.64	1.67	1.74	13.38	24.33
	Space (Gigabytes)	1.525	2.113	14.051	16.331	2.535	2.905	9.495	22.229