Data Augmentation for Graph Neural Networks

Tong Zhao[†*], **Yozen Liu**[‡], **Leonardo Neves**[‡], **Oliver Woodford**[‡], **Meng Jiang**[†], **Neil Shah**[‡]
[†] University of Notre Dame, Notre Dame, IN 46556
[‡] Snap Inc., Santa Monica, CA 90405
{tzhao2, mjiang2}@nd.edu, {yliu2, lneves, oliver.woodford, nshah}@snap.com

Abstract

Data augmentation has been widely used to improve generalizability of machine learning models. However, comparatively little work studies data augmentation for graphs. This is largely due to the complex, non-Euclidean structure of graphs, which limits possible manipulation operations. Augmentation operations commonly used in vision and language have no analogs for graphs. Our work studies graph data augmentation for graph neural networks (GNNs) in the context of improving semi-supervised node-classification. We discuss practical and theoretical motivations, considerations and strategies for graph data augmentation. Our work shows that neural edge predictors can effectively encode class-homophilic structure to promote intra-class edges and demote inter-class edges in given graph structure, and our main contribution introduces the GAUG graph data augmentation framework, which leverages these insights to improve performance in GNN-based node classification via edge prediction. Extensive experiments on multiple benchmarks show that augmentation via GAUG improves performance across GNN architectures and datasets.

1 Introduction

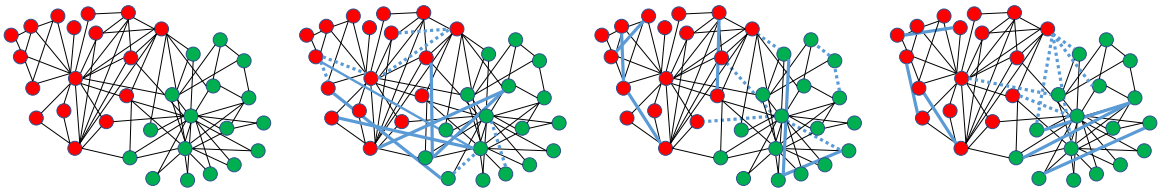
Data driven inference has received a significant boost in generalization capability and performance improvement in recent years from data augmentation techniques. These methods increase the amount of training data available by creating plausible variations of existing data without additional ground-truth labels, and have seen widespread adoption in fields such as computer vision (CV) (DeVries and Taylor 2017; Cubuk et al. 2019; Zhao et al. 2019; Ho et al. 2019), and natural language processing (NLP) (Fadaee, Bisazza, and Monz 2017; Şahin and Steedman 2019). Such augmentations allow inference engines to learn to generalize better across those variations and attend to signal over noise. At the same time, graph neural networks (GNNs) (Hamilton, Ying, and Leskovec 2017; Kipf and Welling 2016a; Veličković et al. 2017; Xu et al. 2018a; Zhang et al. 2019a; Chen, Ma, and Xiao 2018; Wu et al. 2019; Zhang, Cui, and Zhu 2018; Xu et al. 2018b) have emerged as a rising approach for data-driven inference on graphs, achieving promising results on tasks such as node classification, link prediction and graph representation learning.

Despite the complementary nature of GNNs and data augmentation, few works present strategies for combining the two. One major obstacle is that, in contrast to other data, where structure is encoded by position, the structure of graphs is encoded by node connectivity, which is irregular. The hand-crafted, structured, data augmentation operations used frequently in CV and NLP therefore cannot be applied. Furthermore, this irregularity does not lend itself to easily defining new augmentation strategies. The most obvious approaches involve adding or removing nodes or edges. For node classification tasks, adding nodes poses challenges in labeling and imputing features and connectivity of new nodes, while removing nodes simply reduces the data available. Thus, edge addition and removal appears the best augmentation strategy for graphs. But the question remains, *which* edges to change.

Three relevant approaches have recently been proposed. DROPEDGE (Rong et al. 2019) randomly removes a fraction of graph edges before each training epoch, in an approach reminiscent of dropout (Srivastava et al. 2014). This, in principle, robustifies test-time inference, but cannot benefit from added edges. In approaches more akin to denoising or pre-filtering, ADAEDGE (Chen et al. 2019) iteratively add (remove) edges between nodes predicted to have the same (different) labels with high confidence in the modified graph. This ad-hoc, two-stage approach improves inference in general, but is prone to error propagation and greatly depends on training size. Similarly, BGCN (Zhang et al. 2019b) iteratively trains an assortative mixed membership stochastic block model with predictions of GCN to produce multiple denoised graphs, and ensembles results from multiple GCNs. BGCN also bears the risk of error propagation.

Present work. Our work studies new techniques for graph data augmentation to improve node classification. Section 3 introduces motivations and considerations in augmentation via edge manipulation. Specifically, we discuss how facilitating message passing by removing "noisy" edges and adding "missing" edges that could exist in the original graph can benefit GNN performance, and its relation to intra-class and inter-class edges. Figure 1 demonstrates, on a toy dataset (a), that while randomly modifying edges (b) can lead to lower test-time accuracy, strategically choosing ideal edges

[*]This work was done when the author was on internship at Snap Inc.



(a) Original graph. O : 92.4 F1 (b) Random mod. M : 90.4, O : 91.0 F1 (c) Proposed GAUG mod. M : 95.7, O : 94.3 F1 (d) Omniscient mod. M : 98.6, O : 95.6 F1

Figure 1: GCN performance (test micro-F1) on the original Zachary's Karate Club graph in (a), and three augmented graph variants in (b-d), evaluated on both original (O) and modified (M) graph settings. Black, solid-blue, dashed-blue edges denote original graph connectivity, newly added, and removed edges respectively. While random graph modification (b) hurts performance, our proposed GAUG augmentation approaches (c) demonstrate significant relative performance improvements, narrowing the gap to omniscient, class-aware modifications (d).

to add or remove given (unrealistic) omniscience of node class labels (d) can substantially improve it.

Armed with this insight, Section 4 presents our major contribution: the proposed GAUG framework for graph data augmentation. We show that neural edge predictors like GAE (Kipf and Welling 2016b) are able to latently learn class-homophilic tendencies in existent edges that are improbable, and nonexistent edges that are probable. GAUG leverages this insight in two approaches, GAUG-M and GAUG-O, which tackle augmentation in settings where edge manipulation is and is not feasible at inference time. GAUG-M uses an edge prediction module to fundamentally modify an input graph for future training and inference operations, whereas GAUG-O learns to generate plausible edge augmentations for an input graph, which helps node classification without any modification at inference time. In essence, our work tackles the problem of the inherent indeterminate nature of graph data and provides graph augmentations, which can both denoise structure and also mimic variability. Moreover, its modular design allows augmentation to be flexibly applied to any GNN architecture. Figure 1(c) shows GAUG-M and GAUG-O achieves marked performance improvements over (a-b) on the toy graph.

In Section 5, we present and discuss an evaluation of GAUG-O across multiple GNN architectures and datasets, demonstrating a consistent improvement over the state-of-the-art, and quite large in some scenarios. Our proposed GAUG-M (GAUG-O) shows up to 17% (9%) absolute F1 performance improvements across datasets and GNN architectures without augmentation, and up to 16% (9%) over baseline augmentation strategies.

2 Other Related Work

As discussed above, relevant literature in data augmentation for graph neural networks is limited (Rong et al. 2019; Chen et al. 2019; Zhang et al. 2019b). We discuss other related works in tangent domains below.

Graph Neural Networks. GNNs enjoy widespread use in modern graph-based machine learning due to their flexibility to incorporate node features, custom aggregations and inductive operation, unlike earlier works which were based on embedding lookups (Perozzi, Al-Rfou, and Skiena 2014; Wang, Cui, and Zhu 2016; Tang et al. 2015). Many GNN variants have been developed in recent years, following the initial idea of convolution based on spectral graph theory (Bruna et al. 2013). Many spectral GNNs have since been developed and improved by (Defferrard, Bresson, and Vandergheynst 2016; Kipf and Welling 2016a; Henaff, Bruna, and LeCun 2015; Li et al. 2018; Levie et al. 2018; Ma et al. 2020). As spectral GNNs generally operate (expensively) on the full adjacency, spatial-based methods which perform graph convolution with neighborhood aggregation became prominent (Hamilton, Ying, and Leskovec 2017; Veličković et al. 2017; Monti et al. 2017; Gao, Wang, and Ji 2018; Niepert, Ahmed, and Kutzkov 2016), owing to their scalability and flexibility (Ying et al. 2018). Several works propose more advanced architectures which add residual connections to facilitate deep GNN training (Xu et al. 2018b; Li et al. 2019; Verma et al. 2019). More recently, task-specific GNNs were proposed in different fields such as behavior modeling (Wang et al. 2020; Zhao et al. 2020; Yu et al. 2020).

Data Augmentation. Augmentation strategies for improving generalization have been broadly studied in contexts outside of graph learning. Traditional point-based classification approaches widely leveraged oversampling, undersampling and interpolation methods (Chawla et al. 2002; Barandela et al. 2004). In recent years, variants of such techniques are widely used in natural language processing (NLP) and computer vision (CV). Replacement approaches involving synonym-swapping are common in NLP (Zhang, Zhao, and LeCun 2015), as are text-variation approaches (Kafle, Yousefhussien, and Kanan 2017) (i.e. for visual question-answering). Backtranslation methods (Sennrich, Haddow, and Birch 2016; Xie et al. 2019; Edunov et al. 2018) have also enjoyed success. In CV, historical image transformations in the input space, such as rotation, flipping, color space transformation, translation and noise injection (Shorten and Khoshgoftaar 2019), as well as recent methods such as cutout and random erasure (DeVries and Taylor 2017; Zhong et al. 2017) have proven useful. Recently, augmentation via photorealistic generation through adversarial networks shows promise in several applications, especially in medicine (Antoniou, Storkey, and Edwards 2017; Goodfellow et al. 2014). Most-related to our work is liter-

ature on meta-learning based augmentation in CV (Lemley, Bazrafkan, and Corcoran 2017; Cubuk et al. 2019; Perez and Wang 2017), which aim to learn neural image transformation operations via an augmentation network, using a loss from a target network. While our work is similar in motivation, it fundamentally differs in network structure, and tackles augmentation in the much-less studied graph context.

3 Graph Data Augmentation via Edge Manipulation

In this section, we introduce our key idea of graph data augmentation by manipulating $\mathcal{G}$ via adding and removing edges over the fixed node set. We discuss preliminaries, practical and theoretical motivations, and considerations in evaluation under a manipulated-graph context.

3.1 Preliminaries

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be the input graph with node set $\mathcal{V}$ and edge set $\mathcal{E}$. Let $N = |\mathcal{V}|$ be the number of nodes. We denote the adjacency matrix as $\mathbf{A} \in \{0,1\}^{N \times N}$, where $\mathbf{A}_{ij} = 0$ indicates node i and j are not connected. We denote the node feature matrix as $\mathbf{X} \in \mathbb{R}^{N \times F}$, where F is the dimension of the node features and $\mathbf{X}_{i:}$ indicates the feature vector of node i (the ith row of $\mathbf{X}$). We define $\mathbf{D}$ as the diagonal degree matrix such that $\mathbf{D}_{ii} = \sum_j \mathbf{A}_{ij}$.

Graph Neural Networks. In this work, we use the well-known graph convolutional network (GCN) (Kipf and Welling 2016a) as an example when explaining GNNs in the following sections; however, our arguments hold straightforwardly for other GNN architectures. Each GCN layer (GCL) is defined as:

$$\begin{aligned} \mathbf{H}^{(l+1)} &= f_{GCL}(\mathbf{A}, \mathbf{H}^{(l)}; \mathbf{W}^{(l)}) \\ &= \sigma(\tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}} \mathbf{H}^{(l)} \mathbf{W}^{(l)}), \end{aligned} \tag{1}$$

where $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ is the adjacency matrix with added self-loops, $\tilde{\mathbf{D}}$ is the diagonal degree matrix $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$, and $\sigma(\cdot)$ denotes a nonlinear activation such as ReLU.

3.2 Motivation

Practical reasons. Graphs aim to represent an underlying process of interest. In reality, a processed or observed graph may not exactly align with the process it intended to model (e.g. "which users are actually friends?" vs. "which users are observed to be friends?") for several reasons. Many graphs in the real world are susceptible to noise, both adversarial and otherwise (with exceptions, like molecular or biological graphs). Adversarial noise can manifest via spammers who pollute the space of observed interactions. Noise can also be induced by partial observation: e.g. a friend recommendation system which never suggests certain friends to an end-user, thus preventing link formation. Moreover, noise can be created in graph preprocessing, by adding/removing self-loops, removing isolated nodes or edges based on weights. Finally, noise can occur due to human errors: in citation networks, a paper may omit (include) citation to a highly (ir)relevant paper by mistake. All these scenarios can produce a gap between the "observed graph" and the so-called "ideal graph" for a downstream inference task (in our case, node classification).

Enabling an inference engine to bridge this gap suggests the promise of data augmentation via edge manipulation. In the best case, we can produce a graph G_i (ideal connectivity), where supposed (but missing) links are added, and unrelated/insignificant (but existing) links removed. Figure 1 shows this benefit realized in the ZKC graph: strategically adding edges between nodes of the same group (intra-class) and removing edges between those in different groups (inter-class) substantially improves node classification test performance, despite using only a single training example per class. Intuitively, this process encourages smoothness over same-class node embeddings and differentiates other-class node embeddings, improving distinction.

Theoretical reasons. Strategic edge manipulation to promote intra-class edges and demote inter-class edges makes class differentiation in training trivial with a GNN, when done with label omniscience. Consider a scenario of extremity where all possible intra-class edges and no possible inter-class edges exists, the graph can be viewed as k fully connected components, where k is the number of classes and all nodes in each component have the same label. Then by Theorem 1 (proof in Appendix A.1), GNNs can easily generate distinct node representations between distinct classes, with equivalent representations for all same-class nodes. Under this "ideal graph" scenario, learned embeddings can be effortlessly classified.

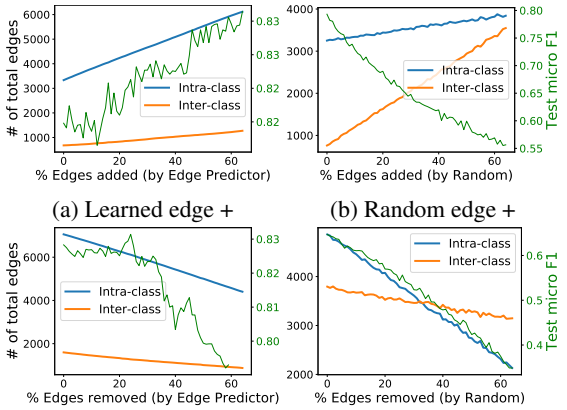
Theorem 1. *Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a undirected graph with adjacency matrix $\mathbf{A}$, and node features $\mathbf{X}$ be any block vector in $\mathbb{R}^{N \times F}$. Let $f : \mathbf{A}, \mathbf{X}; \mathbf{W} \rightarrow \mathbf{H}$ be any GNN layer with a permutation-invariant neighborhood aggregator over the target node and its neighbor nodes $u \cup \mathcal{N}(u)$ (e.g. Eq. 1) with any parameters $\mathbf{W}$, and $\mathbf{H} = f(\mathbf{A}, \mathbf{X}; \mathbf{W})$ be the resulting embedding matrix. Suppose $\mathcal{G}$ contains k fully connected components. Then we have:*

1. *For any two nodes $i, j \in \mathcal{V}$ that are contained in the same connected component, $\mathbf{H}_{i:} = \mathbf{H}_{j:}$.*
2. *For any two nodes $i, j \in \mathcal{V}$ that are contained in different connected components $\mathcal{S}_a, \mathcal{S}_b \subseteq \mathcal{V}$, $\mathbf{H}_{i:} \neq \mathbf{H}_{j:}$ when $\mathbf{W}$ is not all zeros and $\sum_{v \in \mathcal{S}_a} \mathbf{X}_{v:} \neq \varepsilon \sum_{u \in \mathcal{S}_b} \mathbf{X}_{u:}, \forall \varepsilon \in \mathbb{R}$.*

This result suggests that with an ideal, *class-homophilic* graph $\mathcal{G}_i$, class differentiation *in training* becomes trivial. However, it does not imply such results in testing, where node connectivity is likely to reflect $\mathcal{G}$ and not $\mathcal{G}_i$. We would expect that if modifications in training are too contrived, we risk overfitting to $\mathcal{G}_i$ and performing poorly on $\mathcal{G}$ due to a wide train-test gap. We later show techniques (Section 4) for approximating $\mathcal{G}_i$ with a modified graph $\mathcal{G}_m$, and show empirically that these modifications in fact help generalization, both when evaluating on graphs akin to $\mathcal{G}_m$ and $\mathcal{G}$.

3.3 Modified and Original Graph Settings for Graph Data Augmentation

Prior CV literature (Wang, Wang, and Lian 2019) considers image data augmentation a two-step process: (1) applying a transformation $f : \mathcal{S} \rightarrow \mathcal{T}$ to input images $\mathcal{S}$ to generate variants $\mathcal{T}$, and (2) utilizing $\mathcal{S} \cup \mathcal{T}$ for model



(a) Learned edge + (b) Random edge +

(c) Learned edge – (d) Random edge –

Figure 2: GAUG-M uses an edge-predictor module to deterministically modify a graph for future inference. Neural edge-predictors (e.g. GAE) can learn class-homophilic tendencies, promoting intra-class and demoting inter-class edges compared to random edge additions (a-b) and removals (c-d) respectively, leading to node classification performance (test micro-F1) improvements (green).

training. Graph data augmentation is notably different, since typically $|\mathcal{S}| = 1$ for node classification, unlike the image setting where $|\mathcal{S}| \gg 1$. However, we propose two strategies with analogous, but distinct formalisms: we can either (1) apply one or multiple graph transformation operation $f : \mathcal{G} \to \mathcal{G}_m$, such that $\mathcal{G}_m$ replaces $\mathcal{G}$ for both training and inference, or (2) apply many transformations $f_i : \mathcal{G} \to \mathcal{G}_m^i$ for $i = 1 \dots N$, such that $\mathcal{G} \cup \{\mathcal{G}_m^i\}_{i=1}^N$ may be used in training, but only $\mathcal{G}$ is used for inference. We call (1) the *modified-graph* setting, and (2) the *original-graph* setting, based on their inference scenario.

One might ask: when is each strategy preferable? We reason that the answer stems from the feasibility of applying augmentation during inference to avoid a train-test gap. The *modified-graph* setting is thus most suitable in cases where a given graph is unchanging during inference. In such cases, one can produce a single $\mathcal{G}_m$, and simply use this graph for both training and testing. However, when inferences must be made on a dynamic graph (i.e. for large-scale, latency-sensitive applications) where calibrating new graph connectivity (akin to $\mathcal{G}$) with $\mathcal{G}_m$ during inference is infeasible (e.g. due to latency constraints), augmentation in the *original-graph* setting is more appropriate. In such cases, test statistics on $\mathcal{G}_m$ may be overly optimistic as performance indicators. In practice, these loosely align with transductive and inductive contexts in prior GNN literature.

4 Proposed GAUG Framework

In this section, we introduce the GAUG framework, covering two approaches for augmenting graph data in the aforementioned modified-graph and original-graph settings respectively. Our key idea is to leverage information *inherent* in the graph to predict which non-existent edges should likely exist, and which existent edges should likely be removed in $\mathcal{G}$ to produce modified graph(s) $\mathcal{G}_m$ to improve model performance. As we later show in Section 5, by leveraging this label-free information, we can consistently realize improvements in test/generalization performance in semi-supervised node classification tasks across augmentation settings, GNN architectures and datasets.

4.1 GAUG-M for Modified-Graph Setting

We first introduce GAUG-M, an approach for augmentation in the modified-graph setting which includes two steps: (1) we use an edge predictor function to obtain edge probabilities for all possible and existing edges in $\mathcal{G}$. The role of the edge predictor is flexible and can generally be replaced with any suitable method. (2) Using the predicted edge probabilities, we deterministically add (remove) new (existing) edges to create a modified graph $\mathcal{G}_m$, which is used as input to a GNN node-classifier.

The edge predictor can be defined as any model $f_{ep} : \mathbf{A}, \mathbf{X} \to \mathbf{M}$, which takes the graph as input, and outputs an edge probability matrix $\mathbf{M}$ where $\mathbf{M}_{uv}$ indicates the predicted probability of an edge between nodes u and v. In this work, we use the graph auto-encoder (GAE) (Kipf and Welling 2016b) as the edge predictor module due to its simple architecture and competitive performance. GAE consists of a two layer GCN encoder and an inner-product decoder:

$$\mathbf{M} = \sigma\left(\mathbf{Z}\mathbf{Z}^T\right), \text{ where } \mathbf{Z} = f_{GCL}^{(1)}\left(\mathbf{A}, f_{GCL}^{(0)}\left(\mathbf{A}, \mathbf{X}\right)\right). \quad (2)$$

$\mathbf{Z}$ denotes the hidden embeddings learned by the encoder, $\mathbf{M}$ is the predicted (symmetric) edge probability matrix produced by the inner-product decoder, and $\sigma(\cdot)$ is an element-wise sigmoid function. Let $|\mathcal{E}|$ denote the number of edges in $\mathcal{G}$. Then, using the probability matrix $\mathbf{M}$, GAUG-M deterministically adds the top $i|\mathcal{E}|$ non-edges with highest edge probabilities, and removes the $j|\mathcal{E}|$ existing edges with least edge probabilities from $\mathcal{G}$ to produce $\mathcal{G}_m$, where $i, j \in [0, 1]$. This is effectively a denoising step.

Figure 2 shows the change in intra-class and inter-class edges when adding/removing using GAE-learned edge probabilities and their performance implications compared to a random perturbation baseline on CORA: adding (removing) by learned probabilities results in a much steeper growth (slower decrease) of intra-class edges and much slower increase (steeper decrease) in inter-class edges compared to random. Notably, these affect classification performance (micro-F1 scores, in green): random addition/removal hurts performance, while learned addition consistently improves performance throughout the range, and learned removal improves performance over part of the range (until $\sim$20%). Importantly, these results show that while we are generally not able to produce the ideal graph $\mathcal{G}_i$ without omniscience (as discussed in Section 3.2), such capable edge predictors can latently learn to approximate class-homophilic information in graphs and successfully promote intra-class and demote inter-class edges to realize performance gains in practice.

GAUG-M shares the same time and space complexity as its associated GNN architecture during training/inference, while requiring extra disk space to save the dense $O(N^2)$

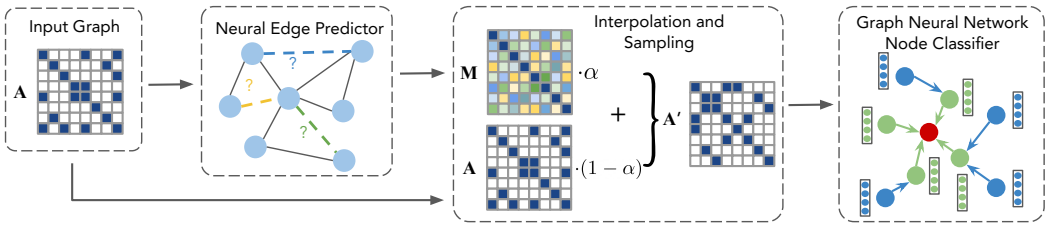


Figure 3: GAUG-O is comprised of three main components: (1) a differentiable edge predictor which produces edge probability estimates, (2) an interpolation and sampling step which produces sparse graph variants, and (3) a GNN which learns embeddings for node classification using these variants. The model is trained end-to-end with both classification and edge prediction losses.

edge probability matrix $\mathbf{M}$ for manipulation. Note that $\mathbf{M}$'s computation can be trivially parallelized.

4.2 GAUG-O for Original-Graph Setting

To complement the above approach, we propose GAUG-O for the original-graph setting, where we cannot benefit from graph manipulation at inference time. GAUG-O is reminiscent of the two-step approach in GAUG in that it also uses an edge prediction module for the benefit of node classification, but also aims to improve model generalization (test performance on $\mathcal{G}$) by generating graph variants $\{\mathcal{G}_m^i\}_{i=1}^N$ via edge prediction and hence improve data diversity. GAUG-O does not require discrete specification of edges to add/remove, is end-to-end trainable, and utilizes both edge prediction and node-classification losses to iteratively improve augmentation capacity of the edge predictor and classification capacity of the node classifier GNN. Figure 3 shows the overall architecture: each training iteration exposes the node-classifier to a new augmented graph variant.

Unlike GAUG-M's deterministic graph modification step, GAUG-O supports a learnable, stochastic augmentation process. As such, we again use the graph auto-encoder (GAE) for edge prediction. To prevent the edge predictor from arbitrarily deviating from original graph adjacency, we interpolate the predicted $\mathbf{M}$ with the original $\mathbf{A}$ to derive an adjacency $\mathbf{P}$. In the edge sampling phase, we sparsify $\mathbf{P}$ with Bernoulli sampling on each edge to get the graph variant adjacency $\mathbf{A}'$. For training purposes, we employ a (soft, differentiable) relaxed Bernoulli sampling procedure as a Bernoulli approximation. This relaxation is a binary special case of the Gumbel-Softmax reparameterization trick (Maddison, Mnih, and Teh 2016; Jang, Gu, and Poole 2016). Using the relaxed sample, we apply a straight-through (ST) gradient estimator (Bengio, Léonard, and Courville 2013), which rounds the relaxed samples in the forward pass, hence sparsifying the adjacency. In the backward pass, gradients are directly passed to the relaxed samples rather than the rounded values, enabling training. Formally,

$$\begin{aligned} \mathbf{A}'_{ij} &= \left\lfloor \frac{1}{1+e^{-(\log \mathbf{P}_{ij}+G)/\tau}} + \frac{1}{2} \right\rfloor, \\ \text{where} \quad \mathbf{P}_{ij} &= \alpha \mathbf{M}_{ij} + (1-\alpha)\mathbf{A}_{ij} \end{aligned} \tag{3}$$

where $\mathbf{A}'$ is the sampled adjacency matrix, τ is the temperature of Gumbel-Softmax distribution, $G \sim Gumbel(0,1)$ is a Gumbel random variate, and α is a hyperparameter mediating the influence of edge predictor on the original graph.

The graph variant adjacency $\mathbf{A}'$ is passed along with node features $\mathbf{X}$ to the GNN node classifier. We then backpropagate using a joint node-classification loss $\mathcal{L}_{nc}$ and edge-prediction loss $\mathcal{L}_{ep}$

$$\begin{aligned} & \mathcal{L} = \mathcal{L}_{nc} + \beta \mathcal{L}_{ep}, \\ \text{where} \quad & \mathcal{L}_{nc} = CE(\hat{\mathbf{y}}, \mathbf{y}) \\ \text{and} \quad & \mathcal{L}_{ep} = BCE(\sigma(f_{ep}(\mathbf{A}, \mathbf{X})), \mathbf{A}) \end{aligned} \tag{4}$$

where β is a hyperparameter to weight the reconstruction loss, $\sigma(\cdot)$ is an elementwise sigmoid, $\mathbf{y}, \hat{\mathbf{y}}$ denote ground-truth node class labels and predicted probabilities, and BCE/CE indicate standard (binary) cross-entropy loss. We train using $\mathcal{L}_{ep}$ in addition to $\mathcal{L}_{nc}$ to control potentially excessive drift in edge prediction performance. The node-classifier GNN is then directly used for inference, on $\mathcal{G}$.

During training, GAUG-O has a space complexity of $O(N^2)$ in full-batch setting due to backpropagation through all entries of the adjacency matrix. Fortunately, we can easily adapt the graph mini-batch training introduced by Hamilton et al. (Hamilton, Ying, and Leskovec 2017) to achieve an acceptable space complexity of $O(M^2)$, where M is the batch size. Appendix C.1 further details (pre)training, mini-batching, and implementation choices.

5 Evaluation

In this section, we evaluate the performance of GAUG-M and GAUG-O across architectures and datasets, and over alternative strategies for graph data augmentation. We also showcase their abilities to approximate class-homophily via edge prediction and sensitivity to supervision.

5.1 Experimental Setup

We evaluate using 6 benchmark datasets across domains: citation networks (CORA, CITESEER (Kipf and Welling 2016a)), protein-protein interactions (PPI (Hamilton, Ying, and Leskovec 2017)), social networks (BLOGCATALOG, FLICKR (Huang, Li, and Hu 2017)), and air traffic (AIR-USA (Wu, He, and Xu 2019)). Statistics for each dataset are shown in Table 1, with more details in Appendix B. We follow the semi-supervised setting in most GNN literature (Kipf and Welling 2016a; Veličković et al. 2017) for train/validation/test splitting on CORA and CITESEER,

Table 1: Summary statistics and experimental setup for the six evaluation datasets.

	CORA	CITESEER	PPI	BLOGCATALOG	FLICKR	AIR-USA
# Nodes	2,708	3,327	10,076	5,196	7,575	1,190
# Edges	5,278	4,552	157,213	171,743	239,738	13,599
# Features	1,433	3,703	50	8,189	12,047	238
# Classes	7	6	121	6	9	4
# Training nodes	140	120	1,007	519	757	119
# Validation nodes	500	500	2,015	1,039	1,515	238
# Test nodes	1,000	1,000	7,054	3,638	5,303	833

Table 2: GAUG performance across GNN architectures and six benchmark datasets.

GNN Arch.	Method	CORA	CITESEER	PPI	BLOGC	FLICKR	AIR-USA
GCN	Original	81.6±0.7	71.6±0.4	43.4±0.2	75.0±0.4	61.2±0.4	56.0±0.8
	+BGCN	81.2±0.8	72.4±0.5	–	72.0±2.3	52.7±2.8	56.5±0.9
	+ADAEDGE	81.9±0.7	72.8±0.7	43.6±0.2	75.3±0.3	61.2±0.5	57.2±0.8
	+GAUG-M	83.5±0.4	72.3±0.4	43.5±0.2	**77.6±0.4**	**68.2±0.7**	61.2±0.5
	+DROPEDGE	82.0±0.8	71.8±0.2	43.5±0.2	75.4±0.3	61.4±0.7	56.9±0.6
	+GAUG-O	**83.6±0.5**	**73.3±1.1**	**46.6±0.3**	75.9±0.2	62.2±0.3	**61.4±0.9**
GSAGE	Original	81.3±0.5	70.6±0.5	40.4±0.9	73.4±0.4	57.4±0.5	57.0±0.7
	+BGCN	80.5±0.1	70.8±0.1	–	73.2±0.2	58.1±0.3	53.5±0.3
	+ADAEDGE	81.5±0.6	71.3±0.8	41.6±0.8	73.6±0.4	57.7±0.7	57.1±0.5
	+GAUG-M	**83.2±0.4**	71.2±0.4	41.1±1.0	**77.0±0.4**	**65.2±0.4**	**60.1±0.5**
	+DROPEDGE	81.6±0.5	70.8±0.5	41.1±1.0	73.8±0.4	58.4±0.7	57.1±0.5
	+GAUG-O	82.0±0.5	**72.7±0.7**	**44.4±0.5**	73.9±0.4	56.3±0.6	57.1±0.7
GAT	Original	81.3±1.1	70.5±0.7	41.5±0.7	63.8±5.2	46.9±1.6	52.0±1.3
	+BGCN	80.8±0.8	70.8±0.6	–	61.4±4.0	46.5±1.9	54.1±3.2
	+ADAEDGE	82.0±0.6	71.1±0.8	42.6±0.9	68.2±2.4	48.2±1.0	54.5±1.9
	+GAUG-M	82.1±1.0	71.5±0.5	42.8±0.9	70.8±1.0	**63.7±0.9**	**59.0±0.6**
	+DROPEDGE	81.9±0.6	71.0±0.5	**45.9±0.3**	70.4±2.4	50.0±1.6	52.8±1.7
	+GAUG-O	**82.2±0.8**	**71.6±1.1**	44.9±0.9	**71.0±1.1**	51.9±0.5	54.6±1.1
JK-NET	Original	78.8±1.5	67.6±1.8	44.1±0.7	70.0±0.4	56.7±0.4	58.2±1.5
	+BGCN	80.2±0.7	69.1±0.5	–	65.7±2.2	53.6±1.7	55.9±0.8
	+ADAEDGE	80.4±1.4	68.9±1.2	44.8±0.9	70.7±0.4	57.0±0.3	59.4±1.0
	+GAUG-M	**81.8±0.9**	68.2±1.4	47.4±0.6	**71.9±0.5**	**65.7±0.8**	60.2±0.6
	+DROPEDGE	80.4±0.7	69.4±1.1	46.3±0.2	70.9±0.4	58.5±0.7	59.1±1.1
	+GAUG-O	80.5±0.9	**69.7±1.4**	**53.1±0.3**	71.0±0.6	55.7±0.5	**60.4±1.0**

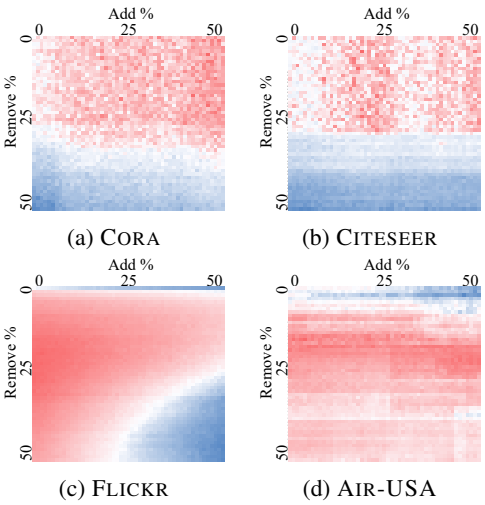
and a 10/20/70% split on other datasets due to varying choices in prior work. We evaluate GAUG-M and GAUG-O using 4 widely used GNN architectures: GCN (Kipf and Welling 2016a), GSAGE (Hamilton, Ying, and Leskovec 2017), GAT (Veličković et al. 2017) and JK-NET (Xu et al. 2018b). We compare our GAUG-M (modified-graph) and GAUG-O (original-graph) performance with that achieved by standard GNN performance, as well as three state-of-the-art baselines: ADAEDGE (Chen et al. 2019) (modified-graph), BGCN (Zhang et al. 2019b) (modified-graph), and DROPEDGE (Rong et al. 2019) (original-graph) evaluating on $\mathcal{G}_m$ and $\mathcal{G}$, respectively. We also show results of proposed GAUG methods on large graphs (Hu et al. 2020) in Appendix D.2 to show their ability of mini-batching. We report test micro-F1 scores over 30 runs, employing Optuna (Akiba et al. 2019) for efficient hyperparameter search. Note that for classification tasks which every object is guaranteed to be assigned to exactly one ground truth class (all datasets except PPI), micro-F1 score is mathematically equivalent to accuracy (proof in Appendix A.2). Our implementation is made publicly available[1].

5.2 Experimental Results

We show comparative results against current baselines in Table 2. Table 2 is organized per architecture (row), per dataset (column), and original-graph and modified-graph settings (within-row). Note that results of BGCN on PPI are missing due to CUDA out of memory error when running the code package from the authors. We bold best-performance per architecture and dataset, but not per augmentation setting for visual clarity. In short, GAUG-O and GAUG-M consistently improve over GNN architectures, datasets and alternatives, with a single exception for GAT on PPI, on which DROPEDGE performs the best.

Improvement across GNN architectures. GAUG achieves improvements over all 4 GNN architectures (averaged

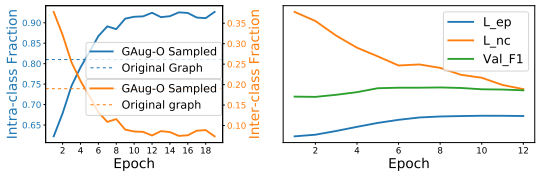
[1] https://github.com/zhao-tong/GAug



(a) CORA (b) CITESEER

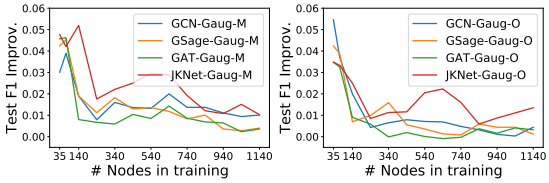
(c) FLICKR (d) AIR-USA

Figure 4: Classification (test) performance heatmaps of GAUG-M on various datasets when adding/dropping edges. Red-white-blue indicate outperformance, at-par, and underperformance w.r.t. GCN on $\mathcal{G}$. Pixel $(0, 0)$ indicates $\mathcal{G}$, and x (y) axes show % edges added (removed).



(a) Edge makeup (b) Learning curve

Figure 5: GAUG-O promotes class-homophily (a), producing classification improvements (b).



(a) GAUG-M (b) GAUG-O

Figure 6: GAUG augmentation especially improves performance under weak supervision.

across datasets): GAUG-M improves 4.6% (GCN), 4.8% (GSAGE), 10.9% (GAT) and 5.7% (JK-NET). GAUG-O improves 4.1%, 2.1%, 6.3% and 4.9%, respectively. We note that augmentation especially improves GAT performance, as self-attention based models are sensitive to connectivity.

Improvements across datasets. GAUG also achieves improvements over all 6 datasets (averaged across architectures): GAUG-M improves 2.4%, 1.0%, 3.1%, 5.5%, 19.2%, 7.9% for each dataset (left to right in Table 2). Figure 4 shows GAUG-M (with GCN) classification performance heatmaps on 4 datasets when adding/removing edges according to various i, j (Section 4.1). Notably, while improvements(red) over original GCN on $\mathcal{G}$ differ over i, j and by dataset, they are feasible in all cases. These improvements are not necessarily monotonic with edge addition(row) or removal(column), and can encounter transitions. Empirically, we notice these boundaries correspond to excessive class mixing (addition) or graph shattering (removal). GAUG-O improves 1.6%, 2.5%, 11.5%, 3.6%, 2.2%, 4.7%. We note that both methods achieves large improvements in social data (BLOGCATALOG and FLICKR) where noisy edges may be prominent due to spam or bots (supporting intuition from Section 3.2): Figure 4(c) shows substantial edge removal significantly helps performance.

Improvements over alternatives. GAUG also outperforms augmentation over BGCN, ADAEDGE, and DROPEDGE (averaged across datasets/architectures): GAUG-M improves 9.3%, 4.8%, and 4.1% respectively, while GAUG-O improves 4.9%, 2.7%, and 2.0% respectively. We reason that GAUG-M outperforms BGCN and ADAEDGE by avoiding iterative error propagation, as well as directly manipulating edges based on the graph, rather than indirectly through classification results. GAUG-O outperforms DROPEDGE via learned denoising via addition and removal, rather than random edge removal. Note that some baselines have worse performance than vanilla GNNs, as careless augmentation/modification on the graph can hurt performance by removing critical edges and adding incorrect ones.

Promoting class-homophily. Figure 5a shows (on CORA) that the edge predictor in GAUG-O learns to promote intra-class edges and demote inter-class ones, echoing results from Figure 2 on GAUG-M, facilitating message passing and improving performance. Figure 5b shows that $\mathcal{L}_{nc}$ decreases and validation F1 improves over the first few epochs, while $\mathcal{L}_{ep}$ increases to reconcile with supervision from $\mathcal{L}_{nc}$. Later on, the $\mathcal{L}_{nc}$ continues to decrease while intra-class ratio increases (overfitting).

Sensitivity to supervision. Figure 6 shows that both GAUG is especially powerful under weak supervision, producing large F1 improvements with few labeled samples. Moreover, augmentation helps achieve equal performance w.r.t standard methods with fewer training samples. Naturally, improvements shrink in the presence of more supervision. GAUG-M tends towards slightly larger outperformance compared to GAUG-O with more training nodes, since inference benefits from persistent graph modifications in the former but not the latter.

6 Conclusion

Data augmentation for facilitating GNN training has unique challenges due to graph irregularity. Our work tackles this problem by utilizing neural edge predictors as a means of exposing GNNs to likely (but nonexistent) edges and limiting exposure to unlikely (but existent) ones. We show that such edge predictors can encode class-homophily to promote intra-class edges and inter-class edges. We propose the GAUG graph data augmentation framework which uses these insights to improve node classification performance in two inference settings. Extensive experiments show our

proposed GAUG-O and GAUG-M achieve up to 17% (9%) absolute F1 performance improvements across architectures and datasets, and 15% (8%) over augmentation baselines.

Ethical Impact

We do not foresee ethical concerns posed by our method, but concede that both ethical and unethical applications of graph-based machine learning techniques may benefit from the improvements induced by our work. Care must be taken, in general, to ensure positive ethical and societal consequences of machine learning.

References

Akiba, T.; Sano, S.; Yanase, T.; Ohta, T.; and Koyama, M. 2019. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD*.

Antoniou, A.; Storkey, A.; and Edwards, H. 2017. Data augmentation generative adversarial networks. *arXiv preprint arXiv:1711.04340* .

Barandela, R.; Valdovinos, R. M.; Sánchez, J. S.; and Ferri, F. J. 2004. The imbalanced training sample problem: Under or over sampling? In *Joint IAPR international workshops on SPR and SSPR*, 806–814. Springer.

Bengio, Y.; Léonard, N.; and Courville, A. 2013. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432* .

Bruna, J.; Zaremba, W.; Szlam, A.; and LeCun, Y. 2013. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203* .

Chawla, N. V.; Bowyer, K. W.; Hall, L. O.; and Kegelmeyer, W. P. 2002. SMOTE: synthetic minority over-sampling technique. *Journal of artificial intelligence research* 16.

Chen, D.; Lin, Y.; Li, W.; Li, P.; Zhou, J.; and Sun, X. 2019. Measuring and Relieving the Over-smoothing Problem for Graph Neural Networks from the Topological View. *arXiv preprint arXiv:1909.03211* .

Chen, J.; Ma, T.; and Xiao, C. 2018. Fastgcn: fast learning with graph convolutional networks via importance sampling. *arXiv preprint arXiv:1801.10247* .

Cubuk, E. D.; Zoph, B.; Mane, D.; Vasudevan, V.; and Le, Q. V. 2019. Autoaugment: Learning augmentation strategies from data. In *Proceedings of the IEEE conference on CVPR*.

Defferrard, M.; Bresson, X.; and Vandergheynst, P. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *NeurIPS*, 3844–3852.

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

Edunov, S.; Ott, M.; Auli, M.; and Grangier, D. 2018. Understanding Back-Translation at Scale. In *Proceedings of the 2018 Conference on EMNLP*, 489–500.

Fadaee, M.; Bisazza, A.; and Monz, C. 2017. Data augmentation for low-resource neural machine translation. *arXiv preprint arXiv:1705.00440* .

Gao, H.; Wang, Z.; and Ji, S. 2018. Large-scale learnable graph convolutional networks. In *Proceedings of the 24th ACM SIGKDD*, 1416–1424.

Goodfellow, I.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; and Bengio, Y. 2014. Generative adversarial nets. In *NeurIPS*, 2672–2680.

Goyal, P.; Dollár, P.; Girshick, R.; Noordhuis, P.; Wesolowski, L.; Kyrola, A.; Tulloch, A.; Jia, Y.; and He, K. 2017. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677* .

Hamilton, W.; Ying, Z.; and Leskovec, J. 2017. Inductive representation learning on large graphs. In *NeurIPS*.

Han, J.; Pei, J.; and Kamber, M. 2011. *Data mining: concepts and techniques*. Elsevier.

Henaff, M.; Bruna, J.; and LeCun, Y. 2015. Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163* .

Ho, D.; Liang, E.; Stoica, I.; Abbeel, P.; and Chen, X. 2019. Population based augmentation: Efficient learning of augmentation policy schedules. *arXiv preprint arXiv:1905.05393* .

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

Huang, X.; Li, J.; and Hu, X. 2017. Label informed attributed network embedding. In *Proceedings of the Tenth ACM International Conference on WSDM*, 731–739.

Jang, E.; Gu, S.; and Poole, B. 2016. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144* .

Kafle, K.; Yousefhussien, M.; and Kanan, C. 2017. Data Augmentation for Visual Question Answering. In *Proceedings of the 10th International Conference on Natural Language Generation*, 198–202.

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

Kipf, T. N.; and Welling, M. 2016b. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308* .

Lemley, J.; Bazrafkan, S.; and Corcoran, P. 2017. Smart augmentation learning an optimal data augmentation strategy. *Ieee Access* 5.

Levie, R.; Monti, F.; Bresson, X.; and Bronstein, M. M. 2018. Cayleynets: Graph convolutional neural networks with complex rational spectral filters. *IEEE Transactions on Signal Processing* 67(1): 97–109.

Li, G.; Muller, M.; Thabet, A.; and Ghanem, B. 2019. Deepgcns: Can gcns go as deep as cnns? In *Proceedings of the IEEE ICCV*, 9267–9276.

Li, R.; Wang, S.; Zhu, F.; and Huang, J. 2018. Adaptive graph convolutional neural networks. In *32th AAAI*.

Ma, Y.; Liu, X.; Zhao, T.; Liu, Y.; Tang, J.; and Shah, N. 2020. A Unified View on Graph Neural Networks as Graph Signal Denoising. *arXiv preprint arXiv:2010.01777* .

Maddison, C. J.; Mnih, A.; and Teh, Y. W. 2016. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712* .

Monti, F.; Boscaini, D.; Masci, J.; Rodola, E.; Svoboda, J.; and Bronstein, M. M. 2017. Geometric deep learning on graphs and manifolds using mixture model cnns. In *Proceedings of the IEEE Conference on CVPR*, 5115–5124.

Niepert, M.; Ahmed, M.; and Kutzkov, K. 2016. Learning convolutional neural networks for graphs. In *ICML*.

Perez, L.; and Wang, J. 2017. The effectiveness of data augmentation in image classification using deep learning. *arXiv preprint arXiv:1712.04621* .

Perozzi, B.; Al-Rfou, R.; and Skiena, S. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD*, 701–710.

Rong, Y.; Huang, W.; Xu, T.; and Huang, J. 2019. DropEdge: Towards Deep Graph Convolutional Networks on Node Classification. In *ICLR*.

Şahin, G. G.; and Steedman, M. 2019. Data Augmentation via Dependency Tree Morphing for Low-Resource Languages. *arXiv preprint arXiv:1903.09460* .

Sennrich, R.; Haddow, B.; and Birch, A. 2016. Improving Neural Machine Translation Models with Monolingual Data. In *Proceedings of the 54th ACL*, 86–96.

Shorten, C.; and Khoshgoftaar, T. M. 2019. A survey on image data augmentation for deep learning. *Journal of Big Data* 6(1): 60.

Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; and Salakhutdinov, R. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research* 15(1): 1929–1958.

Tang, J.; Qu, M.; Wang, M.; Zhang, M.; Yan, J.; and Mei, Q. 2015. Line: Large-scale information network embedding. In *Proceedings of the 24th WWW*, 1067–1077.

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

Verma, V.; Qu, M.; Lamb, A.; Bengio, Y.; Kannala, J.; and Tang, J. 2019. Graphmix: Regularized training of graph neural networks for semi-supervised learning. *arXiv preprint arXiv:1909.11715* .

Wang, D.; Cui, P.; and Zhu, W. 2016. Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD*, 1225–1234.

Wang, D.; Jiang, M.; Syed, M.; Conway, O.; Juneja, V.; Subramanian, S.; and Chawla, N. V. 2020. Calendar Graph Neural Networks for Modeling Time Structures in Spatiotemporal User Behaviors. In *Proceedings of the 26th ACM SIGKDD*.

Wang, X.; Wang, K.; and Lian, S. 2019. A survey on face data augmentation. *arXiv preprint arXiv:1904.11685* .

Wu, J.; He, J.; and Xu, J. 2019. DEMO-Net: Degree-specific graph neural networks for node and graph classification. In *Proceedings of the 25th ACM SIGKDD*, 406–415.

Wu, Z.; Pan, S.; Chen, F.; Long, G.; Zhang, C.; and Yu, P. S. 2019. A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596* .

Xie, Q.; Dai, Z.; Hovy, E.; Luong, M.-T.; and Le, Q. V. 2019. Unsupervised Data Augmentation for Consistency Training. *arXiv preprint arXiv:1904.12848* .

Xu, K.; Hu, W.; Leskovec, J.; and Jegelka, S. 2018a. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826* .

Xu, K.; Li, C.; Tian, Y.; Sonobe, T.; Kawarabayashi, K.-i.; and Jegelka, S. 2018b. Representation learning on graphs with jumping knowledge networks. *arXiv preprint arXiv:1806.03536* .

Ying, R.; He, R.; Chen, K.; Eksombatchai, P.; Hamilton, W. L.; and Leskovec, J. 2018. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD*, 974–983.

Yu, W.; Yu, M.; Zhao, T.; and Jiang, M. 2020. Identifying referential intention with heterogeneous contexts. In *Proceedings of The Web Conference*.

Zhang, C.; Song, D.; Huang, C.; Swami, A.; and Chawla, N. V. 2019a. Heterogeneous graph neural network. In *Proceedings of the 25th ACM SIGKDD*, 793–803.

Zhang, X.; Zhao, J.; and LeCun, Y. 2015. Character-level Convolutional Networks for Text Classification .

Zhang, Y.; Pal, S.; Coates, M.; and Ustebay, D. 2019b. Bayesian graph convolutional neural networks for semi-supervised classification. In *AAAI*, volume 33, 5829–5836.

Zhang, Z.; Cui, P.; and Zhu, W. 2018. Deep learning on graphs: A survey. *arXiv preprint arXiv:1812.04202* .

Zhao, A.; Balakrishnan, G.; Durand, F.; Guttag, J. V.; and Dalca, A. V. 2019. Data augmentation using learned transformations for one-shot medical image segmentation. In *Proceedings of the IEEE conference on CVPR*, 8543–8553.

Zhao, T.; Deng, C.; Yu, K.; Jiang, T.; Wang, D.; and Jiang, M. 2020. Error-Bounded Graph Anomaly Loss for GNNs. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*.

Zhong, Z.; Zheng, L.; Kang, G.; Li, S.; and Yang, Y. 2017. Random erasing data augmentation. *arXiv preprint arXiv:1708.04896* .

A Proofs

A.1 Proof of Theorem 1

We first reproduce the definition of a permutation-invariant neighborhood aggregator (Xu et al. 2018a) in the context of graph convolution:

Definition 1. *A neighborhood aggregator* $f : \{\mathbf{X}_{v:}, v \in u \cup \mathcal{N}(u)\} \rightarrow \bar{\mathbf{X}}_{u:}$ *is called* ***permutation-invariant*** *when it is invariant to the order of the target node and its neighbor nodes* $u \cup \mathcal{N}(u)$*, i.e. let* $\{x_1, x_2, \ldots, x_M\} = \{\mathbf{X}_{v:}, v \in u \cup \mathcal{N}(u)\}$*, then for any permutation* π : $f(\{x_1, x_2, \ldots, x_M\}) = f(\{x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(M)}\})$.

Next, we prove Theorem 1:

Proof. Let $\tilde{\mathbf{A}}$ be the adjacency matrix with added self loops, i.e., $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$. Here we denote the calculation process of a GNN layer with a permutation-invariant neighborhood aggregator as:

$$\mathbf{H} = f(\mathbf{A}, \mathbf{X}; \mathbf{W}) = \sigma(\bar{\mathbf{A}}\mathbf{X}\mathbf{W}) \tag{5}$$

where σ denotes a nonlinear activation (e.g. ReLU) and $\bar{\mathbf{A}}$ denotes the normalized adjacency matrix according to the design of different GNN architectures. For example, in GCN layer (Kipf and Welling 2016a), $\bar{\mathbf{A}} = \tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}$; in GSAGE layer with GCN aggregator (Hamilton, Ying, and Leskovec 2017), $\bar{\mathbf{A}}$ is the row L1-normalized $\mathbf{A}$.

For any two nodes $i, j \in \mathcal{V}$ that are contained in the same fully connected component $\mathcal{S} \subseteq \mathcal{V}$, i and j are only connected to all other nodes in $\mathcal{S}$ by definition. Hence $\tilde{\mathbf{A}}_{iv} = \tilde{\mathbf{A}}_{jv} = 1, \forall v \in \mathcal{S}$ and $\tilde{\mathbf{A}}_{iu} = \tilde{\mathbf{A}}_{ju} = 0, \forall u \notin \mathcal{S}$, that is, $\tilde{\mathbf{A}}_{i:} = \tilde{\mathbf{A}}_{j:}$. Moreover, as the degrees of all nodes in the same fully connected component are the the same, we have $\bar{\mathbf{A}}_{i:} = \bar{\mathbf{A}}_{j:}$. Thus by Equation 5, $\mathbf{H}_{i:} = \mathbf{H}_{j:}$.

On the other hand, for any two nodes $i, j \in \mathcal{V}$ that are contained in different fully connected components $\mathcal{S}_a, \mathcal{S}_b \subseteq \mathcal{V}$ respectively ($a \neq b$), i and j are not connected and do not share any neighbors by definition. As all nodes in $\mathcal{S}_a$ have the same degree, all nonzero entries in $\tilde{\mathbf{A}}_{i:}$ would have the same positive value $\bar{a}$. Similarly, all the nonzero entries in $\tilde{\mathbf{A}}_{j:}$ also have the same positive value $\bar{b}$. Then, by Equation 5, the embeddings of i and j after the GNN layer will respectively be

$$\mathbf{H}_{i:} = \bar{a} \sum_{v \in \mathcal{S}_a} \mathbf{X}_{v:}\mathbf{W}, \qquad \mathbf{H}_{j:} = \bar{b} \sum_{u \in \mathcal{S}_b} \mathbf{X}_{u:}\mathbf{W}. \tag{6}$$

From the above equation, we can observe that $\mathbf{H}_{i:} \neq \mathbf{H}_{j:}$ when $\mathbf{W}$ is not all zeros and $\sum_{v \in \mathcal{S}_a} \mathbf{X}_{v:} \neq \frac{\bar{b}}{\bar{a}} \sum_{u \in \mathcal{S}_b} \mathbf{X}_{u:}$. □

A.2 Micro-F1 and Accuracy

Definition 2. *Micro-F1 score is mathematically equivalent to accuracy for classification tasks when every data point is guaranteed to be assigned to exactly one class (one ground truth label for each data point.)*

Proof. The $micro\text{-}F1$ score is defined as following (Han, Pei, and Kamber 2011):

$$\begin{aligned} &micro\text{-}precision = TP/(TP + FP) \\ &micro\text{-}recall = TP/(TP + FN) \\ &micro\text{-}F1 = 2 \cdot \frac{micro\text{-}precision \cdot micro\text{-}recall}{micro\text{-}precision + micro\text{-}recall} \end{aligned} \tag{7}$$

where TP, TN, FP, and FN are the number of true positives, true negatives, false positives and false negatives for all classes.

As each data object only has one label, for each misclassified data object, one FP case for the predicted class and one FN case for the ground truth class are created at the same time. Therefore, $micro\text{-}precision$ and $micro\text{-}recall$ will always be the same and we then have:

$$micro\text{-}precision = micro\text{-}recall = micro\text{-}F1 \tag{8}$$

Accuracy is defined as the number of correct predictions divided by the number of total cases (Han, Pei, and Kamber 2011). Since the number of correct predictions is the same as TP and the number of incorrect predictions is FP, accuracy can also be calculated as following:

$$Accuracy = TP/(TP + FP) \tag{9}$$

Thus we have:

$$Accuracy = micro\text{-}precision = micro\text{-}F1 \tag{10}$$

□

B Additional Dataset Details

In this section, we provide some additional, relevant dataset details. The preprocessed files of all datasets used in this work can be found at https://tinyurl.com/gaug-data, including graph adjacency matrix, node features, node labels, train/validation/test node ids and predicted edge probabilities for each dataset.

Citation networks. CORA and CITESEER are citation networks which are used as benchmarks in most GNN-related prior works (Kipf and Welling 2016a; Veličković et al. 2017; Rong et al. 2019; Chen et al. 2019). In these networks, the nodes are papers published in the field of computer science; the features are bag-of-word vectors of the corresponding paper title; the edges represent the citation relation between papers; the labels are the category of each paper.

Protein-protein interaction network. PPI is the combination of multiple protein-protein interaction networks from different human tissue. The node feature contains positional gene sets, motif gene sets and immunological signatures. Gene ontology sets are used as labels (121 in total) (Hamilton, Ying, and Leskovec 2017). The original graph provided by (Hamilton, Ying, and Leskovec 2017) contains total of 295 connected components in various sizes, so in this work we took the top 3 largest connected components, forming a graph with 10,076 nodes.

Social networks. BLOGCATALOG is an online blogging community where bloggers can follow each other, hence forming a social network. The features for each user are

generated by the keywords in each bloggers description and the labels are selected from predefined categories of blogger interests (Huang, Li, and Hu 2017). FLICKR is an image and video sharing platform, where users can also follow each other, hence forming a social network. The user-specified list of interest tags are used as user features and the groups that users joined are used as labels (Huang, Li, and Hu 2017).

Air traffic network. AIR-USA is the airport traffic network in the USA, where each node represents an airport and edge indicates the existence of commercial flights between the airports. The node labels are generated based on the label of activity measured by people and flights passed the airports (Wu, He, and Xu 2019). The original graph does not have any features, so we used one-hot degree vectors as node features.

C Implementation Details and Hyperparameter Tuning

All experiments were conducted on a virtual machine on Google Cloud[2] with 15 vCPUs, 15 Gb of RAM and one NVIDIA Tesla v100 GPU card (16 Gb of RAM at 32Gbps speed).

C.1 Notes for effectively training GAUG-O

Pretraining the Edge Predictor and Node Classifier. Since the graph structure of the GNN node classifier largely depends on the edge predictor, we pretrain both components of GAUG-O to achieve more stable joint training. Otherwise, a randomly initialized edge predictor can generate very unlikely edge probabilities $\mathbf{M}$, which stunt training. Empirically, we find that pretraining the edge predictor is more important in producing good performance compared to the node classifier, and excessive pretraining of the node classifier can lead to overfitting and poor optimizer performance.

Learning Rate Warmup for the Edge Predictor. Since the edge predictor is not only trained using $\mathcal{L}_{ep}$, but also by $\mathcal{L}_{nc}$, we adapt the learning rate warmup schema (Goyal et al. 2017) for the edge predictor to avoid effective undoing of initial pretraining. Specifically, we initialize the edge predictor's learning rate at zero and gradually increase following a sigmoid curve. This empirically helps avoid sudden drift in edge prediction from $\mathcal{L}_{nc}$, and improves results. We also incorporate a parameter that narrows down a section of the sigmoid curve to specify how rapid the learning rate warms up.

Mini-batch Training. For graphs that are too large and exceed GPU memory in full-batch training with GAUG-O, we follow the mini-batch training algorithm proposed by Hamilton, Ying, and Leskovec (2017). For each batch, we first randomly sample a group of seed nodes from the training nodes. We then populate the batch with the union of seed nodes and their k-hop neighbors to form a subgraph, where k is the number of layers in GNNs. Lastly, the subgraph representing the mini-batch, together with their node features is fed as input to the model. $\mathcal{L}_{ep}$ is calculated from the adjacency matrix of this subgraph; $\mathcal{L}_{nc}$ is calculated only with the predictions on seed nodes.

Note that the sampled graph for each mini-batch would be subtly different from the one during full-batch training, as the model cannot sample any new edges between seed nodes and target nodes outside of the extended subgraph. Nevertheless, this slight difference does not affect the training much as most of the sampled new edges are within the subgraph (within a few hops).

C.2 Hyperparameters and Search Space

In this section, we describe the parameters of all methods along with the search space of all hyperparameters. All methods were implemented in `Python 3.7.6` with `PyTorch`. Our implementation can be found at https://github.com/zhao-tong/GAug. We further include code for ADAEDGE (Chen et al. 2019) and DROPEDGE (Rong et al. 2019) for comparisons. The best hyperparameter choices and searching scripts can be found in the supplementary material.

Original graph neural network architectures. All original GNN architectures are implemented in `DGL`[3] with Adam optimizer. We search through the basic parameters such as learning rate and the choice of aggregators (and number of layers only for JK-NET) to determine the default settings of each GNN. By default, GCN, GSAGE and GAT have 2 layers, and JK-NET has 3 layers due to its unique design. GCN, GSAGE and JK-NET have hidden size 128, and GAT has a hidden size of 16 for each head (have use 8 heads). GCN, GSAGE and JK-NET have learning rates of $1\mathrm{e}{-2}$ and GAT has best performance with learning rate of $5\mathrm{e}{-3}$. All methods have weight decay of $5\mathrm{e}{-4}$. GCN, GSAGE and JK-NET use feature dropout of 0.5, while GAT uses both feature dropout and attention dropout of 0.6. For GSAGE, we use the GCN-style aggregator. For JK-NET, we use GSAGE layer with GCN-style aggregator as neighborhood aggregation layers and concatenation for the final aggregation layer. To make fair comparisons, these parameters are fixed for all experiments and our hyperparameter searches only search over the new parameters introduced by baselines and our proposed methods.

ADAEDGE. We implement ADAEDGE (Chen et al. 2019) based on the above mentioned GNNs and the provided pseudo-code in their paper in `PyTorch`, since the author-implemented code was unavailable. We tune the following hyperparameters over ranges: $order \in \{add\_first, remove\_first\}$, $num_+ \in \{0, 1, \ldots, |\mathcal{E}| - 1\}$, $num_- \in \{0, 1, \ldots, |\mathcal{E}| - 1\}$, $conf_+ \in [0.5, 1]$, $conf_- \in [0.5, 1]$.

DROPEDGE. We also implement DROPEDGE (Rong et al. 2019) (adapting the authors' code for easier comparison) based on the above mentioned GNNs, where the GNNs randomly remove $p|\mathcal{E}|$ of the edges and redo the normalization on the adjacency matrix before each training epoch, where p is searched in the range of $[0, 0.99]$.

[2] https://cloud.google.com/

[3] https://www.dgl.ai/

Table 3: Ablation study of GAUG-O on CORA

Setting	GCN	GSAGE	GAT	JK-NET
Original (GCN)	81.6±0.7	81.3±0.5	81.3±1.1	78.0±1.5
+GAUG-O	**83.6±0.5**	**82.0±0.5**	**82.2±0.8**	**80.5±0.9**
+GAUG-O No Sampling	82.8±0.9	81.2±0.8	77.8±2.2	76.9±1.4
+GAUG-O Rounding	82.5±0.5	81.4±0.5	81.3±1.1	79.5±1.3
+GAUG-O No $\mathcal{L}_{ep}$	82.8±0.8	81.5±1.1	81.9±0.8	79.5±1.0

BGCN. The BGCN model consists of two parts: an assortative mixed membership stochastic block model (MMSBM) and a GNN. For MMSBM, we use the code package[4] provided by the authors (Zhang et al. 2019b). For GNNs, we use the above mentioned implementations. We follow the training process provided in the authors' code package.

GAUG-M. As described in Section 4.1, GAUG-M has two hyperparameters i and j, which are both searched within the range of $\{0, 0.01, 0.02, \ldots, 0.8\}$.

GAUG-O. We tune the following hyperparameters over search ranges for GAUG-O. The influence of the edge predictor on the original graph: $\alpha \in \{0, 0.01, 0.02, \ldots, 1\}$; the weight for $\mathcal{L}_{ep}$ when training the model: $\beta \in \{0, 0.1, 0.2, \ldots, 4\}$; the temperature for the relaxed Bernoulli sampling: $temp \in \{0.1, 0.2, \ldots, 2\}$; number of pretrain epochs for both edge predictor and node classifier: $n\_pretrain_{ep}, n\_pretrain_{nc} \in \{5, 10, 15, \ldots, 300\}$; the parameter for warmup: $warmup \in \{0, 1, \ldots, 10\}$.

Mini-batch training. For vanilla GNNs, we follow the standard mini-batch training proposed by Hamilton, Ying, and Leskovec (2017) and Ying et al. (2018). For GNNs with GAUG-M, both the VGAE edge predictor and GNN also followed the same mini-batch training. For GNNs with GAUG-O, we used mini-batch training as described in Appx. C.1.

D Discussion and Additional Experimental Results

D.1 Ablation Study for GAUG-O

We extensively study the effect of various design choices in GAUG-O to support our decisions. Here, we compare the results of the 4 GNN architectures in combination with baseline and GAUG-O applied with different graph sampling and training choices on CORA. The results are shown in Table 3.

No Sampling: Instead of interpolating $\mathbf{M}$ and $\mathbf{A}$ and subsequently sampling, we avoid the sampling step and feed the fully dense adjacency into the GNN classifier (every edge is used for convolution, with different weights). This shows decreased performance compared to sampling-based solution, likely because the sampling removes noise from many, very weak edges.

Rounding: Instead of sampling the graph, we deterministically round the edge probabilities to 0 or 1, using the same ST gradient estimator in the backward pass. Rounding creates the same decision boundary for edge and no-edge at each epoch. We observe that it hurts performance compared to sampling, likely due to reduced diversity during model training.

[4]https://github.com/huawei-noah/BGCN

Table 4: Summary statistics for the large datasets.

	PUBMED	OGBN-ARXIV
# Nodes	19,717	169,343
# Edges	44,338	1,166,243
# Features	500	128
# Classes	3	40
# Training nodes	60	90,941
# Validation nodes	500	29,799
# Test nodes	1000	48,603

Table 5: GAUG performance with mini-batch training.

Methods	PUBMED	OGBN-ARXIV
GCN	78.5±0.5	68.1±0.3
GCN + GAUG-M	**80.2±0.3**	68.2±0.3
GCN + GAUG-O	79.3±0.4	**71.4±0.5**

No Edge-Prediction Loss: Instead of training GAUG-O with a positive β (coefficient for $\mathcal{L}_{ep}$), we set $\beta = 0$. Without controlling drift of pre-trained edge predictor by combining its loss($\mathcal{L}_{ep}$) with node classification loss($\mathcal{L}_{nc}$) in training, we risk generating unrealistic graphs which arbitrarily deviate from the original graph, and producing instability in training. We find that removing this loss term leads to empirical performance decrease.

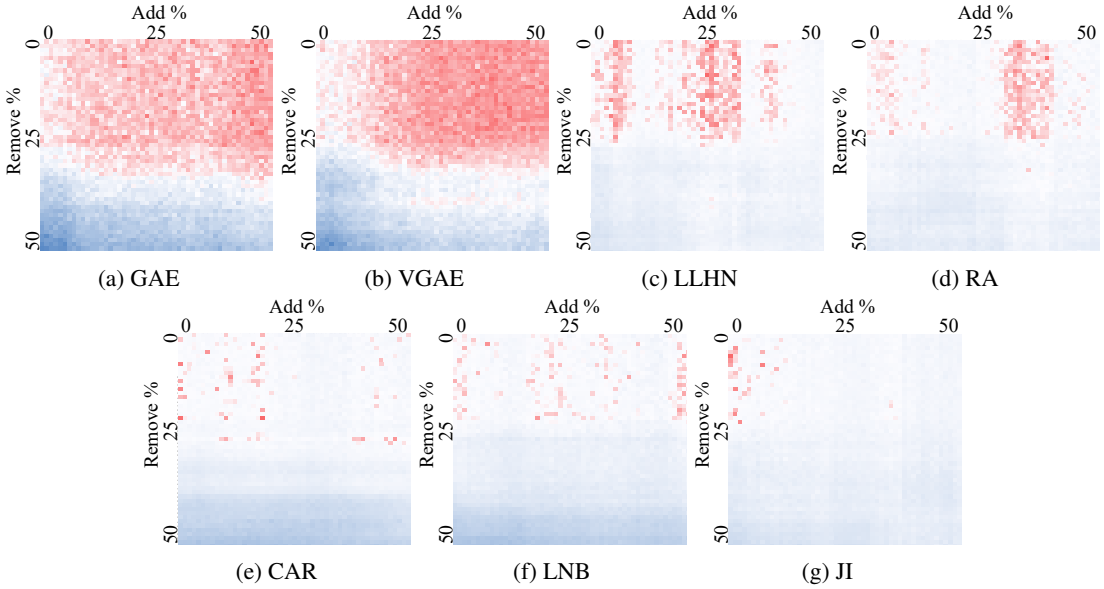
D.2 GAUG with Mini-batch Training

In order to better show the scalability of the proposed methods with mini-batch training, Table 5 shows that the proposed GAUG methods are able to perform well when using mini-batch training. More specifically, on PUBMED, augmentation (via GAUG-M) achieves 2.2% improvement, while on OGBN-ARXIV, augmentation (via GAUG-O) achieves 4.8% improvement. All three methods are trained in the mini-batch setting with the same batch size. Note that the performance using mini-batch training is generally not as good as full-batch training (even for vanilla GNNs), hence mini-batch training is only recommended when graphs are too large to fit in GPU as a whole.

Similar to CORA and CITESEER, both of the two large graph datasets are citation networks: PUBMED is a commonly used GNN benchmark (Kipf and Welling 2016a), and OGBN-ARXIV is a standard benchmark provided by the

Table 6: GAUG-M performance on original and modified graphs.

Backbone	Method	CORA	CITESEER	PPI	BLOGC	FLICKR	AIR-USA
GCN	original	81.6±0.7	71.6±0.4	43.4±0.2	75.0±0.4	61.2±0.4	56.0±0.8
	+GAUG-M	**83.5±0.4**	72.3±0.4	**43.5±0.2**	**77.6±0.4**	**68.2±0.7**	**61.2±0.5**
	+GAUG-M-*O*	83.1±0.5	**72.8±0.5**	**43.5±0.2**	75.6±0.4	61.6±0.6	58.1±0.6
GSAGE	Original	81.3±0.5	70.6±0.5	40.5±0.9	73.4±0.4	57.4±0.5	57.0±0.7
	+GAUG-M	**83.2±0.4**	71.2±0.4	**41.1±1.0**	**77.0±0.4**	**65.2±0.4**	**60.1±0.5**
	+GAUG-M-*O*	82.4±0.5	**71.6±0.3**	41.1±1.4	74.3±0.3	58.1±0.5	58.9±0.5
GAT	Original	81.3±1.1	70.5±0.7	41.5±0.7	63.8±5.2	49.6±1.6	52.0±1.3
	+GAUG-M	**82.1±1.0**	**71.5±0.5**	42.8±0.9	70.8±1.0	**63.7±0.9**	**59.0±0.6**
	+GAUG-M-*O*	82.0±0.9	71.3±0.7	**46.3±0.2**	**71.0±1.3**	48.5±1.9	53.4±1.1
JK-NET	Original	78.8±1.5	67.6±1.8	44.1±0.7	70.0±0.4	56.7±0.4	58.2±1.5
	+GAUG-M	**81.8±0.9**	68.2±1.4	47.4±0.6	**71.9±0.5**	**65.7±0.8**	**60.2±0.6**
	+GAUG-M-*O*	80.6±1.0	**68.3±1.4**	**48.6±0.5**	71.0±0.4	57.0±0.4	60.2±0.8



(a) GAE (b) VGAE (c) LLHN (d) RA

(e) CAR (f) LNB (g) JI

Figure 7: GAUG-M with GCN on CORA with different edge prediction heuristics

Open Graph Benchmark[5] (Hu et al. 2020). Table 4 summarizes their statistics.

D.3 Evaluating GAUG-M on original vs. modified graph

Although GAUG-M is designed for the modified-graph setting, it is still possible to do inference on $\mathcal{G}$ while training the model on $\mathcal{G}_m$. As previously mentioned in Section 3.3, inference with GAUG-M on G would result in a train-test gap, which would affect the test performance. Table 6 presents the inference results of GAUG-M with $\mathcal{G}_m$ and $\mathcal{G}$ (GAUG-M-*O* makes inference on $\mathcal{G}$). We can observe that both variants of GAUG-M show performance improvements over the original GNNs across different architectures and datasets. Moreover, inference with GAUG-M on $\mathcal{G}_m$ has equal or better performance in almost cases, which aligns with our intuition in Section 3.3. This suggests that GAUG-M actually improves training in a way that helps generalization even on the original graph by better parameter inference, despite modifying the graph during training to achieve this.

D.4 GAUG-M performance under different edge predictors

Although we use GAE as the edge prediction model of choice due to its strong performance in link prediction, GAUG-M can be generally equipped with any edge prediction module. In Figure 7 we show classification performance heatmaps of GAUG-M (with GCN) on CORA, when adding/removing edges according to different heuristics. Specifically, graph auto-encoder (GAE) (Kipf and Welling 2016b), variational graph auto-encoder (VGAE) (Kipf and Welling 2016b), the Local Leicht-Holme-Newman Index (LLHN), the Resource Allocation Index (RA), CAR-based Indices (CAR), Local Naive Bayes (LNB) and the Jaccard

[5]https://ogb.stanford.edu/

Table 7: GAUG performance for deeper GNNs.

# of layers	Method	GCN	GSAGE	GAT	JK-NET
default	Original	81.6±0.7	81.3±0.5	81.3±1.1	78.0±1.5
	+GAUG-M	83.5±0.4	**83.2±0.4**	82.1±1.0	**81.8±0.9**
	+GAUG-O	**83.6±0.5**	82.0±0.5	**82.2±0.8**	80.5±0.9
4 layers	Original	74.7±2.7	78.9±1.4	79.8±1.0	79.6±1.6
	+GAUG-M	78.9±1.0	**81.5±0.8**	**81.4±0.8**	**81.9±1.2**
	+GAUG-O	**80.6±0.9**	80.1±1.0	75.3±2.1	80.9±0.9
6 layers	Original	57.2±9.6	77.7±1.3	77.8±1.5	79.7±1.0
	+GAUG-M	74.2±2.4	**80.7±1.1**	**79.2±0.8**	**82.0±0.8**
	+GAUG-O	**79.8±1.0**	79.8±0.7	13.6±2.8	80.9±0.6
8 layers	Original	25.0±4.7	61.7±9.9	65.0±6.4	79.2±1.5
	+GAUG-M	56.4±5.7	**78.1±2.0**	**77.9±1.5**	**82.1±0.8**
	+GAUG-O	**77.4±1.9**	76.7±1.5	13.0±0.0	81.1±1.1

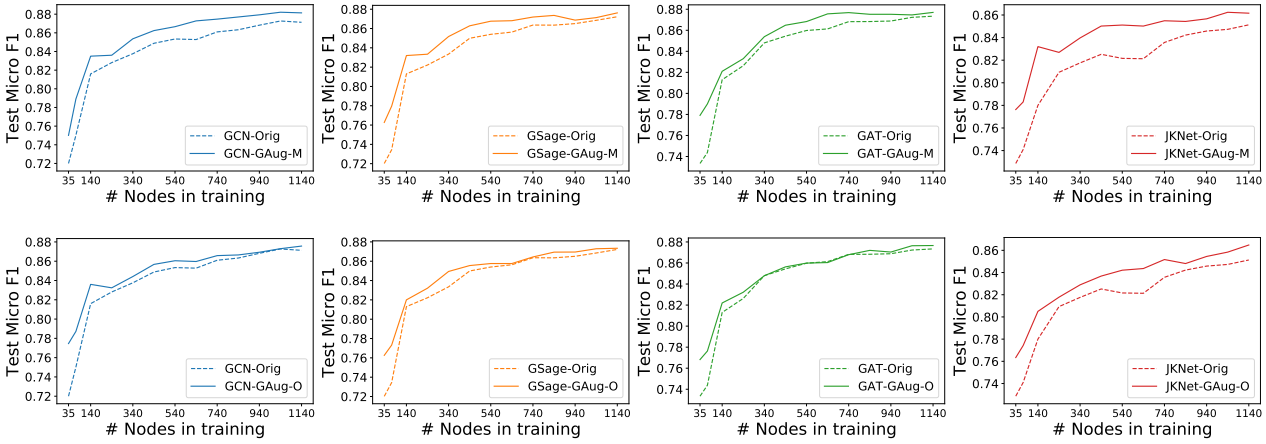
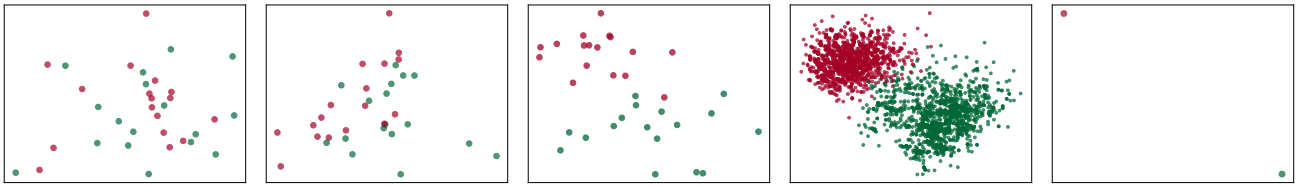


Figure 8: GAUG improves performance under weak supervision with each GNN (GCN, GSAGE, GAT and JK-NET, left to right) and across augmentation settings (GAUG-M on top, GAUG-O on bottom). Relative improvement is clear even with many training nodes, but is larger with few training nodes.

Index (JI). The first two are GCN based neural auto-encoder models and the later two are edge prediction methods based on local neighborhoods, which are commonly used and celebrated in network science literature. It is noticeable that even with the same dataset, the performance heatmaps are characteristically different when using various edge prediction methods, demonstrating the relative importance of edge predictor to GAUG-M's augmentation performance. Moreover, it supports our findings on the importance strategic edge addition and removal to improve performance of graph augmentation/regularization based methods – as Table 2 shows, careless edge addition/removal can actually hurt performance. We do not show results when equipping GAUG-O with these different edge predictors, since GAUG-O requires the edge predictor to be differentiable for training (hence our choice of GAE).

D.5 Classification Performance with Deeper GNNs

In Table 7 we show the performance of our proposed GAUG framework with different number of layers on the CORA dataset. As mentioned in Appendix C.2, GCN, GSAGE and GAT have 2 layers by default, while JK-NET has 3 layers due to it's unique design. From Table 7 we can observe that when increasing the number of layers, most GNNs perform worse except for JK-NET, which is specifically designed for deep GNNs. GAUG-M shows stable performance improvements over all GNN architectures with different depth; GAUG-O shows performance improvements on GCN, GSAGE and JK-NET with different depth. Our results suggest that augmentation can be a tool which facilitates deeper GNN training, as performance improvements for the common practical GNN implementations (GCN and GSAGE) demonstrate quite large performance improvements when compared to standard implementations (e.g. 52.4 point absolute F1 improvement for GCN, 16.4 point absolute F1 improvement for GSAGE at 8 layers).



(a) Raw features (b) Original graph (c) Modified graph (d) 100 Mod. graph (e) Ideal graph

Figure 9: Embeddings after one GCN layer. (c) and (e) show that augmentation can produce more clear decision boundaries between red and green nodes, compared to raw features (a), and naive GCN on raw features (b). (e) shows the effortless classification possible in the ideal graph scenario, where all same-class nodes have the same embedding.

D.6 GAUG's sensitivity to supervision

As previously mentioned in Section 5, which we showed that GAUG is especially powerful under weak supervision, in Figure 8 we detail the sensitivity to supervision for each GNN combined with GAUG-M and GAUG-O. We can observe a larger separation between original (dotted) and GAUG (solid) in each plot when training samples decrease, indicating larger performance gain under weak supervision. In most settings, test F1 score of GAUG-M and GAUG training with 35 training nodes is at par or better than baseline training with 2 times as many (70) nodes. This suggest that GAUG is an especially appealing option in cases of weak supervision, like in anomaly detection and other imbalanced learning settings.

D.7 Embedding Visualization

To further illustrate the case for graph data augmentation in addition to Fig. 1 and Section 3.2, we plot the features and embeddings to understand how edge manipulation can contribute to lower-effort (and in the fully class-homophilic/ideal scenario, effortless) classification. In Figure 9, we randomly initialize 2-D node features for the Zachary's Karate Club graph from normal distribution $\mathcal{N}(0, 1)$ (Figure 9a), and show the results of applying a single graph convolution layer (Eq. 1) with randomly initialized weights. When the input graph is "ideal" (all intra-class edges exist, and no inter-class edges exist) all same-class points project to the same embedding, making discriminating the classes trivial (Figure 9e). It is obvious that the original feature and embeddings (Figure 9b) are harder for a classifier to separate than Figure 9c, which the graph was modified by adding (removing) intra(inter)-class edges (simulating the scenario which GAUG-M achieves). Figure 9d shows superimposed points resulting from 100 different modifications of the graph (simulating GAUG-O), illustrating the much clearer decision boundary between red-class and green-class points.