Contents lists available at ScienceDirect

Computer Science Review

journal homepage: www.elsevier.com/locate/cosrev

Review article

Towards data augmentation in graph neural network: An overview and evaluation

Michael Adjeisah [a], Xinzhong Zhu [a,b,*], Huiying Xu [a,*], Tewodros Alemu Ayall [a]

[a] College of Mathematics and Computer Science, Zhejiang Normal University, Jinhua, 321004, China
[b] Artificial Intelligence Research Institute of Beijing Geekplus Technology Co. Ltd., Beijing, 100101, China

ARTICLE INFO

Article history:
Received 24 April 2022
Received in revised form 22 August 2022
Accepted 22 November 2022
Available online 9 December 2022

Keywords:
Feature representation
Graph data augmentation
Graph neural network
Topology augmentation

ABSTRACT

Many studies on Graph Data Augmentation (GDA) approaches have emerged. The techniques have rapidly improved performance for various graph neural network (GNN) models, increasing the current state-of-the-art accuracy by absolute values of 4.20%, 5.50%, and 4.40% on Cora, Citeseer, and PubMed, respectively. The success is attributed to two integral properties of relational approaches: topology-level and feature-level augmentation. This work provides an overview of some GDA algorithms which are reasonably categorized based on these integral properties. Next, we engage the three most widely used GNN backbones (GCN, GAT, and GraphSAGE) as plug-and-play methods for conducting experiments. We conclude by evaluating the algorithm's effectiveness to demonstrate significant differences among various GDA techniques based on accuracy and time complexity with additional datasets different from those used in the original works. While discussing practical and theoretical motivations, considerations, and strategies for GDA, this work comprehensively investigates the challenges and future direction by pinpointing several open conceivable issues that may require further study based on far-reaching literature interpretation and empirical outcomes.

Contents

1. Introduction 2
2. Overview 3
  2.1. Graph neural network 3
  2.2. Graph data augmentation 4
    2.2.1. Edges perturbation 4
    2.2.2. Node augmentations 4
    2.2.3. Attributes masking 4
    2.2.4. Graph diffusion 4
    2.2.5. Subgraph extraction 4
3. Categorization of GDA 5
  3.1. Topology-level augmentation 5
    3.1.1. DropEdge 5
    3.1.2. AdaEdge 6
    3.1.3. GAUG-O 6
    3.1.4. GCA 7
  3.2. Feature-level augmentation 7
    3.2.1. NodeAug 7
    3.2.2. FLAG 8
    3.2.3. G-GNN 9
    3.2.4. LA-GNN 9
4. Experiments 10
  4.1. Dataset 10
  4.2. Parameter settings 10

[*] Corresponding authors.
E-mail addresses: madjeisah@zjnu.edu.cn (M. Adjeisah), zxz@zjnu.edu.cn (X. Zhu), xhy@zjnu.edu.cn (H. Xu), ayalltewodros@zjnu.edu.cn (T.A. Ayall).

https://doi.org/10.1016/j.cosrev.2022.100527
1574-0137/© 2022 Elsevier Inc. All rights reserved.

4.3. Experimental results 11
5. Current challenges and future direction 12
6. Conclusion 13
Declaration of competing interest 13
Acknowledgments 13
References 14

1. Introduction

Data augmentation [1,2] applies label-preserving transformations such as rotation and translations [3] in input images in Computer Vision (CV) and back translation [4] and Easy Data Augmentation (EDA) [5] in Natural Language Processing (NLP) tasks. It effectively increases the available training set by constructing apparent data variations without additional ground truth labels, with minimal computational overhead, providing an excellent justification for combating overfitting in deep neural networks [6]. Unfortunately, despite being effective for images and texts, the structured and hand-crafted data augmentation strategies often employed in CV and NLP cannot be extended to graph-level data due to irregular node connectivity encoding in a graph structure compared to positional encoding image and text data structure [7] as far as graph representation learning with GNN [8–11] is concerned.

Graph representation learning has emerged as a practical approach to exploring graph-level data [12] over the past few years. There has been substantial interest in transforming nodes into low-dimensional dense embeddings [13] that maintain the structural features and attributes of the graph. Therefore, it is becoming more and more widespread for graph-level data [14]. Moreover, multiple works in graph-based studies involve variants of GNNs for link prediction [15], node classification [8,9,16,17], and graph classification [18] and have achieved remarkable state-of-the-art performances.

Apparently, after a few years of development, the research in GNN suffered from impediments due to the highly limited data sizes of the traditional graph datasets. Such impediments include over-fitting and over-smoothing [19–21], unrealistic and arbitrary data splits, non-rigorous evaluation metrics, and common neglect of validation set [22–24]. However, the Open Graph Benchmark (OGB) datasets [25] have tackled some of these significant issues and unraveled more practical challenges in the GNN research society. Furthermore, a review of the standard GCN for node classification reveals that they are usually shallow, limiting the number of layers to 2 [7]. Therefore, an attempt to go deeper is still entrenched with these impediments. Most recently, several works have proposed research on developing deep GNN models [17,19,26,27] for node classification, ranging from node sampling [28–30], layer sampling [16,31,32], and subgraph sampling [33,34]. Nevertheless, such research still falls in the highly limited data sizes of graph datasets which have led to GDA such as adversarial perturbation [35–38], node and structural perturbation [13,14,39] and all other methods. Despite the progress of numerous GDA techniques developed for various GNN architectures, we observe that there are only a few surveys [40] to recap and examine recent progress.

Furthermore, all these works on graph data augmentation, ranging from global [41] to local [42] perturbations, can be grouped into two categories. Therefore, we provide an overview of GDA by discussing current state-of-the-art algorithms and their progress to bridge the gap in other works over time. More importantly, various studies propose various mathematical formulations, making comparing different methods challenging. However, this work unifies the formulation surrounding most GDA studies by grouping the algorithms into two categories: topology-level $\mathbf{A}' = \mathcal{H}_{\varphi}(\mathbf{X}, \mathbf{A})$ (Section 3.1) and feature-level $\mathbf{X}' = \mathcal{H}^{\varphi}(\mathbf{X}, \mathbf{A})$ (Section 3.2), augmentation. Then, we tune the formulation of the proposed algorithms for each category based on each group. What is more, all the proposed GDA methods can be applied with any GNN model in a plug-and-play manner to extensively perform experimental analysis across a diverse set of benchmarks without using a sophisticated architecture. Hence, we perform such experiments on eleven public graph datasets to evaluate their performance. Note that the evaluation is not to prove which algorithms perform best since they are uniquely designed to experiment on a different datasets. Instead, our main objective is to project how data augmentation has improved accuracy to current state-of-the-art and demonstrated significant differences among various GDA techniques. Furthermore, the various algorithms discussed (See Table 1 for perturbed parts) involve different types of data augmentation, such as sampling and reconstruction, in the adjacency matrix **A**, the feature matrix **X**, or the combination of both. Table 2 describes the notations used in this work.

Table 1
Algorithms and augmentation types used relative to the part considered in the graph data.

Method	Type	Considered part	Perturbed part
AdaEdge	Sampling	**A**	**A**
DROPEDGE	Sampling	**A**	**A**
GAUG-O	Reconstruction	**A&X**	**A**
G-GNN	Reconstruction	**A&X**	**X**
LA-GNN	Generation	**A&X**	**X**
NodeAug	Sampling	**A&X**	**X**
FLAG	Generation	**X**	**X**
GCA	Generation	**A&X**	**A**

In summary, the contributions of this work are as follows:

1. We provide an overview of GDA in GNN based on an in-depth exploration of the state-of-the-art techniques.
2. We reasonably categorize the state-of-the-art GDA algorithms into topology-level and feature-level augmentation to comprehend the similarity and the difference between various works. In addition to the outlined category, we interpret the algorithms proposed for each in a unified mathematical formulation.
3. We apply the GDA methods to the three most widely used GNN variants to conduct extensive experiments with additional datasets different from the datasets used in the original work to test the algorithm's effectiveness for a fair and reliable comparison based on accuracy and time complexity.
4. Finally, we comprehensively investigate the challenges and future directions of GDA based on the interpretation of the wide-reaching literature and empirical results. We provide a GitHub repository[1] with a reading checklist of various GDA works and codes that will be constantly updated.

The rest of the paper is organized as follows. Section 2 introduces an overview of recent Graph-level data augmentation

[1] https://github.com/Madjeisah/gda-overview.

and some key augmentation strategies. A representative of GDA algorithms and newly proposed ones based on the two categories explaining their core methods are thoroughly discussed in Section 3. Section 4 presents various graph datasets, experimental analysis, and visualization of results. Section 5 discusses the challenges and future direction associated with GDA. Finally, we conclude this work in Section 6.

2. Overview

2.1. Graph neural network

Given that $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, is an undirected graph, with vertices or node set $\mathcal{V} = \{v_1, \ldots, v_n\}$ and edge set $\mathcal{E}$. The adjacency matrix of $\mathcal{G}$ is the $\mathbf{A} \in \mathbb{R}^{n\times n}$ matrix such that:

$$\mathbf{A}_{ij} = \begin{cases} 1, & \text{if}(v_i, v_j) \in \mathcal{E}; \\ 0, & \text{if}(v_i, v_j) \notin \mathcal{E}. \end{cases} \tag{1}$$

Thus binary matrix $\mathbf{A}$ in a graph with edges of n nodes is such that, if there is an edge between node i and j, then $\mathbf{A}_{ij} = 1$, else $\mathbf{A}_{ij} = 0$. Hence $\mathbf{A}$ is also written as $\mathbf{A} = [\mathbf{A}_{ij}]$. The attribute matrix $\mathbf{X}$ is represented as $\mathbf{X} \in \mathbb{R}^{n\times q}$, where the ith row denotes v_i's attributes and q denotes the total number of the attributes. During training the entire adjacency matrix $\mathbf{A}$ and node attributes $\mathbf{X}$ serves as an input data:

$$\mathcal{X} = \mathcal{H}\Big(\mathbf{A}, \mathbf{X}\Big), \tag{2}$$

where, $\mathcal{H}\big(\cdot\big)$ is a perturbation function. GNN, such as Graph Convolutional Networks (GCN) [8] and Graph Attention Networks (GAT) [9], are neural network types that operate on graph structures to capture the dependence of graphs through message handling between the nodes. The feed forward propagation in graph convolutional operations is mathematically expressed as:

$$H^{(l)} = \lambda\Big(\hat{A}H^{(l-1)}W^{(l)}\Big). \tag{3}$$

Here, $H^{(l)} \in \mathbb{R}^{n\times d^{(l)}}$ and $H^{(l-1)} \in \mathbb{R}^{n\times d^{(l-1)}}$ are the output and input node representation matrices of the lth layer, $W^{(l)} \in \mathbb{R}^{d^{(l-1)}\times d^{(l)}}$ is a trainable weight matrix for the lth layer. $\hat{A} = D^{-\frac{1}{2}}\hat{\hat{A}}D^{-\frac{1}{2}}$, where, $\hat{\hat{A}} = \mathbf{A} + \boldsymbol{I}$ is the adjacency matrix with self-connections. The diagonal node degree matrix $D = \begin{pmatrix} d_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & d_n \end{pmatrix}$, if d_i be the degree of v_i in Eq. (1), and $\lambda\big(\cdot\big)$ is a non-linear activation function. Following Hu et al. [43], the GNN architecture operates on two main layer parameters–**AGGREGATE**$\big(\cdot\big)$ and **COMBINE**$\big(\cdot\big)$. The former is passing messages from all neighboring nodes and edges [44], and COMBINE with the previous embedding of the given node in the later.

$$\begin{aligned} \mathrm{msg}_v^{(k)} &= \mathrm{AGGREGATE}^{(k)}\Big(\Big\{h_u^{(k-1)} : u \in \mathcal{N}(v)\Big\}\Big) \\ h_v^{(k)} &= \mathrm{COMBINE}^{(k)}\Big(h_v^{(k-1)}, \mathrm{msg}_v^{(k)}\Big), \end{aligned} \tag{4}$$

where, $h_v^{(k)}$ denotes the embedding of node v at the kth layer and $\mathcal{N}(v)$ represents the neighbor set of node v which produces the propagation perspective through the graph by its edges. The formulation allows the GNN to function on arbitrary sizes and shapes of graphs, ensuring that the update stage is consistent with the sequence of nodes (generally known as permutation invariance). GCN integrates the aggregate and combined functions into a single update equation [8,45] as:

$$h_v^{(k)} = \lambda\left(W^{(k)} \cdot \sum_{u\in\mathcal{N}(v)\cup\{v\}} \frac{1}{d_u d_v} h_u^{(k-1)}\right), \tag{5}$$

where, λ is a ReLU activation function. Given a node, GAT updates the embedding by assigning attention to each edge with a non-isotropic aggregation to specify the contribution of neighboring nodes. Mathematically,

$$\begin{aligned} e_{uv}^{(k)} &= \lambda\Big(\mathit{\Lambda} \cdot \mathrm{CONCAT}^{(k)}\Big[W^{(k)}h_u^{(k-1)}, W^{(k)}h_u^{(k-1)}\Big]\Big) \\ \alpha_{uv}^{(k)} &= \bar{\lambda}_{u\in\mathcal{N}(v)\cup\{v\}}\Big(e_{uv}^{(k)}\Big) \\ h_v^{(k)} &= \sigma\left(\sum_{u\in\mathcal{N}(v)\cup\{v\}} \alpha_{uv}^{(k)}W^{(k)}h_u^{(k-1)}\right). \end{aligned} \tag{6}$$

Here, $\mathit{\Lambda}$ denotes the attention, λ and $\bar{\lambda}$ are the activation functions of LeakyReLU and SOFTMAX, respectively. α is the final non-linearity (usually a softmax or logistic sigmoid for classification problems) [9]. Such learned attentions are helpful to analyze and interpret the graph. Similarly to GCN, **Graph SA**mple and aggre**G**at**E** (GraphSAGE) [29], on the other hand, uses max pooling instead of symmetric mean pooling or normalization as:

$$\begin{aligned} a_v^{(k)} &= \mathrm{MAX}\Big(\Big\{\lambda\Big(W_{pool}^{(k)} \cdot h_u^{(k-1)}\Big) : u \in \mathcal{N}(v)\Big\}\Big) \\ h_v^{(k)} &= \lambda\Big(W^{(k)} \cdot \mathrm{CONCAT}\Big[h_u^{(k-1)}, a_v^{(k)}\Big]\Big). \end{aligned} \tag{7}$$

All node embeddings are transformed using a fully connected layer in the aggregate function, performing an element-wise max-pooling function to aggregate information from all neighbors. The aggregate and previous embedding output are first concatenated with the combine function for a fully connected layer with the ReLU activation function. Other variants of GNN include the Graph Isomorphism Network (GIN) [10] and Simple Graph Convolution (SGC) [46]. However, we limit this work to how various GDAs engage the three widely used variants (GCN, GAT, and GraphSAGE, also called GSAGE).

The primary motivation is that GCN is the most popular GNN architecture due to its simplicity and effectiveness in various tasks and applications. To be precise, the node representations in each layer are updated according to a propagation rule, as shown in Eq. (3). In GCNs, the importance of a neighbor j is specified by the weight of its edge $\mathbf{A}_{ij}$ for a target node i. Nevertheless, in practice, the edge weights may not be able to reflect the true strength between two nodes, hence the inability to handle noisy input graphs.

GAT is built on the principle that the approach automatically learns the importance of each neighbor based on the Attention mechanism [47]. In GAT, the attention layer defines how to transfer the hidden node representations in layer $k-1$ to the new node representations k. A shared linear transformation is then applied to every node to sufficiently transform the lower-level node to higher-level node representations. Self-attention is finally defined on the nodes to estimate the attention coefficients for any pair of nodes. Both GCN and GAT are suitable for node classification in the transductive setting; however, they are not scalable in the inductive setting.

As the graph data is rapidly growing, the graph size is increasing exponentially. The medium-size ogbn-arxiv dataset of the OGB datasets consists of relatively large nodes and edges. Therefore, the original implementation of GNN is not suitable due to the large memory requirement and weak gradient update. Also,

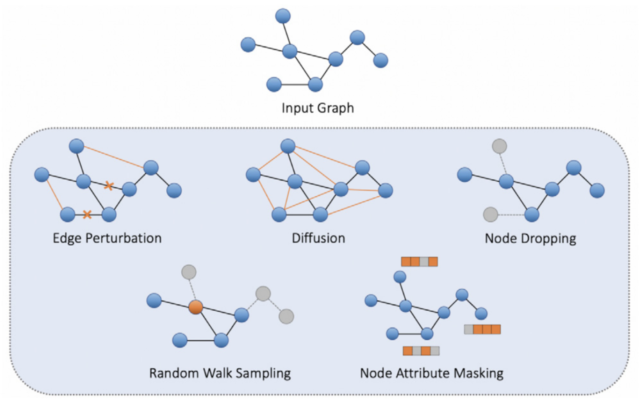

Fig. 1. The various types of data augmentation strategies for graphs [45].

as mentioned, most of the work focuses on transductive learning on a fixed-size graph [8] at the early stage of GNN development. In many cases, the inductive setting is more applicable. Although Yang et al. [48] developed inductive learning on graph embeddings, inductive representation learning was first considered in GraphSAGE on large-scale graphs. It presents a generalized aggregator, mini-batch training, and fixed-size neighbor sampling algorithm to accelerate the training process, hence more scalable.

Ultimately, GCN is a spectral-based graph convolutional network, GAT is based on the Attention mechanism, and GraphSAGE uses mean, LSTM and max-pooling aggregator, respectively, to aggregate the information from neighbors [47].

2.2. Graph data augmentation

Graph data augmentation initiates data objects through structure modification or feature generation, a technique similar to data augmentation in CV and NLP. However, graphs are connected data with complex non-Euclidean structures. The advantage of graph data is the ability to model a complex data relationship that is intimately structured to accommodate inference of indirect facts and tangentially related information. The edges are just as important and detailed as the vertices/nodes. Therefore, the GDA methods modify the entire graph for tasks at the node and edge level instead of some data objects [40].

Researchers have recently focused on uncovering suitable augmentation strategies for learning better representations for different GNN tasks. While some augmentation approaches change the original graph's structure, others maintain the graph structure. In addition, they perturb the node features, making the representations consistent with initial node attributes. It has also been established that the involvement of various pairs of augmentations relative to the graph and its augmentation further improves performance [14,49,50]. For instance, GCA composed edge perturbation and applied attribute masking to achieve adequate performance in denser graphs. Subgraph and node dropping are generally beneficial in various domains of datasets. On the other hand, edge perturbation benefits social networks, however, and damages some biochemical molecules [14]. The following is a summary of the popular GDA approaches for GNN and an illustration in Fig. 1.

2.2.1. Edges perturbation

Given the graph $\mathcal{G}$ or batched graphs $\mathcal{G}_b$ it perturbs the connectivity by randomly adding or dropping a specific ratio of edges in an existing graph to create new graphs. The implication is that the semantic aspects of $\mathcal{G}$ or $\mathcal{G}_b$ have a certain robustness to the variances of the edge connectivity patterns [14,26,27,51]. In particular, edge perturbation has been established to benefit social networks, but is conceptually incompatible with data from biomedical molecules [36,52].

After DropEdge [27], which randomly dropped a fixed fraction of edges, TADropEdge [53] dropped the edges utilizing the weights of the edges as probabilities. Zheng et al. [54] employed an MLP-based graph sparsification model to remove the potential tasks-irrelevant edges; the proposed Neural Sparse model is supervised and jointly trained with GNN for node classification tasks. However, set a high constraint on the modified graph. Luo et al. [55] further proposed using a nuclear norm regularization loss to impose a low-rank constraint by the graph sparsification model [40].

2.2.2. Node augmentations

Given the graph, the primary objective of node dropping is to randomly discard a certain fraction of vertices along with their connections to create new graphs. The underlying prior enforced by it is that all edges linked to that particular node are deleted, while missing parts of the vertices does not affect the semantic aspects of the graph [14,35,36]. For example, DropNode [56] removes nodes by masking selected nodes' features. Some works [7] modified the graph structure for training, and Mixup [57] combined two existing nodes to generate new nodes.

2.2.3. Attributes masking

The underlying objective of attribute masking is to learn representations consistent with node features based on the graph structure. Some masking approaches are fashioned by sampling each mask entry from a Gaussian of pre-specified mean and variance [11,43]. In contrast, others propose masking patterns and more high-degree hub nodes to benefit from denser graphs [14]. For example, FLAG [6] engaged gradient-based adversarial perturbations to increase the characteristics of the nodes, and You et al. [14] randomly masked off-node features.

2.2.4. Graph diffusion

This approach transforms the adjacency matrix in graph convolution into a diffusion matrix using a heat kernel that provides a global view instead of the local view of the graph [58,59]. GNNs can aggregate information only in one hop in each layer during message passing. The diffusion-based induced graph by Klicpera et al. [60] is used for training and inference. The method allows GNNs to learn from multi-hop information without redesigning the model. However, the approach is limited to only the created views. The MV-GCN method [61] utilizes the information given by the different graph diffusions (both the created views and the original graph).

2.2.5. Subgraph extraction

Random walk [22,62] has been established as one of the approaches to continue adding nodes until a fixed predecided number of nodes is achieved to form a subgraph given the graph $\mathcal{G}$ or batched graphs $\mathcal{G}_b$. The works [63,64] related to the subgraph show that implementing local (subgraphs) and global information consistency is advantageous for representation learning. Although operations are established to affect numerous nodes, they are generally adopted for graph-level tasks. In particular, the recent use of subgraph cropping [49,65] and the creation of new graphs by mixing up two subgraphs [66–68] in subgraph augmentation.

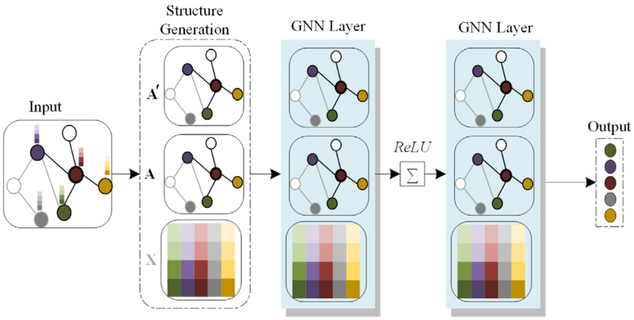

Fig. 2. Schematic diagram of a two-layer GNN illustrating structure representation learning.

Table 2
List of key notations.

Notations	Description
$\mathcal{G} = \left(\mathcal{V}, \mathcal{E}\right)$	Graph representation
$\mathcal{G}_i = \left(\mathcal{V}_i, \mathcal{E}_i\right)$	Subgraph representation
$\mathbf{A}$	Adjacency matrix
$\mathbf{X}$	Attribute matrix
$\mathcal{H}\left(\cdot\right)$	Perturbation function
$\Theta\&\tilde{\Theta}$	Model parameter
W	Trainable weight matrix
χ/η	Regularization coefficients
$d_\omega\left(\cdot\right)$	Calculates the distance between the input matrix and the subspace
$\mathcal{D}_{KL}\left(\cdots\right)$	The Kullback–Leibler divergence (KL divergence)
$\mathcal{Q}_\phi$	The generator
P_ϑ	The predictor
$J_\zeta\left(z\vert\mathbf{X}_j, \mathbf{X}_i\right)$	The encoder
$P_\kappa\left(\mathbf{X}_j\vert\mathbf{X}_i, z\right)$	The decoder
η_{CE}/η_{BCE}	Standard (binary) cross-entropy loss

3. Categorization of GDA

This section presents a detailed discussion of eight GDA algorithms based on *Topology-level* (graph structure representation learning) and *Feature-level* (graph feature representation learning) augmentation, respectively, considering practical and theoretical motivations. Accordingly, the attributed graph $\mathcal{G}$, can be divided into two parts: the topology part and the feature part [69].

3.1. Topology-level augmentation

Topological augmentation, which focuses on the generation of structure representations, can be perceived as a modification to a partition-based relation such as the direction relation matrix [70]. Topology-level enhancement is focused on the graph part, as shown in Fig. 2. It usually perturbs the adjacency matrix $\mathbf{A}$ to generate different graph structures, which can be formulated as:

$$\mathbf{A}' = \mathcal{H}_\varphi\Big(\mathbf{X}, \mathbf{A}\Big), \tag{8}$$

where $\mathcal{H}_\varphi\left(\cdot\right)$ is a structure generation function. The approach involves the removal or adding edges. While dropping and adding edges emerge as the most promising augmentation scheme for graphs, the question remains, which edges to alter or eliminate? In addition, some implementations treat the graph topology as ground-truth information, leading to a low prediction.

Adding/removing certain edges makes node connections sparse, therefore, avoiding over-smoothing to some extent when the GCN goes deeper [27]. In topology-level augmentation, Adaptive Edge Optimization (AdaEdge) [26] and DropEdge have proven such a capability. While the latter randomly terminates a substantial proportion of edges of the input graph by generating different random distorted copies of the original graph, the former engages the graph topology by iteratively adding the intra-class edges and removing the inter-class edges during training. In addition, DropEdge is treated as a message-passing reducer when the message passing between adjacent nodes is carried out along the edge paths in GCNs. The GAUG framework (GAUG-O) [7] engages in a dual step, that is; (1) to obtain edge probabilities for all potential and existing edges through the edge predictor function; (2) to use the predicted edge probabilities to add/remove existing edges to construct a modified graph to serve as input to a GNN node classifier. Other techniques include Memory Tower Augmentation (MeTA) [71]. This augmentation approach augments both the temporal and topology features by first perturbing the edge time to simulate time shifts and removing/adding edges to modify the topology.

The following subsections discuss practical and theoretical motivations, considerations, and strategies for topology-level GDA techniques [72]. Specifically, we focus on AdaEdge, DropEdge, GAUG-O, and Graph Contrastive Learning with Adaptive Augmentation (GCA) [13] that engaged a joint adaptive GDA strategy at the topology and node attribute levels.

3.1.1. DropEdge

Rong et al. [27] developed a novel approach called DropEdge that alleviates over-fitting and over-smoothing, the two primary impediments [19–21] of deep GNN. The approach acts as a data augmenter and a message-passing reducer. DropEdge draws additional virtual examples from neighborhood information for each node by randomly removing edges using the Vicinal Risk Minimization (VRM) principle [73] to increase support for the training distribution. The technique randomly removes a substantial proportion of the edges of the input graph by forcing nonzero elements $\mathcal{V}_s$ of the adjacency matrix $\mathbf{A}$ to be zeros. In this case $\mathcal{V}$ becomes the total number of edges and s represents the rate of drop. Therefore, the resulting adjacency matrix in relation to $\mathbf{A}$ is:

$$\mathcal{H}_\varphi\Big(\mathbf{X}, \mathbf{A}\Big) = \mathbf{A} - \mathbf{A}_q. \tag{9}$$

In particular, Eq. (9) is independent of $\mathbf{X}$, where $\mathbf{A}_q$ is a sparse matrix consisting of a random subset of size $\mathcal{V}_s$ from the original edges $\mathcal{E}$. Layer-wise, DropEdge formulation is a one-shot with all layers sharing a similar perturbed adjacency matrix, and hence it can be applied on the individual layer. Meaning $\mathcal{H}_\varphi\left(\mathbf{X}, \mathbf{A}\right)^{(l)}$ can be obtained independently by computing the lth layer, making different layers have a different adjacency matrix. The aggregation of neighborhood information for each node; the weighted sum (associated with the edges) of the neighbor features is the key in GCN. To make DropEdge an unbiased data augmentation approach for GNN variants during training, the authors used a multiplier s to anticipate neighbor aggregation with respect to the probability of edges to drop. In practice, s is terminated after weight normalization. Therefore, DropEdge models the vicinity for the nodes sharing the same class and not altering any change in anticipation of neighbor aggregation, hence combatting over-fitting. Another major issue of deep GCN training is the node features that converge to a fixed point as the network layer increases, resulting in over-smoothing [19]. For a more general purpose, DropEdge deals with this by working with the concept of subspace [74,75], introducing several useful definitions to simplify the idea.

Given a definition of a subspace, $\omega := \left\{\Lambda C \vert C \in \mathbb{R}^{M\times C}\right\}$ as a subspace of dimensions M in $\mathbb{R}^{N\times C}$, such that $\Lambda \in^{N\times M}$

is orthogonal, meaning $\Lambda^T \Lambda = I_M$ and $M \leq N$. The authors presented the ϵ-smoothing notation of node characteristics for a GCN, if the entire hidden vectors exceed a specific layer (L) with a distance not larger than ϵ ($\epsilon > 0$). Hence, represent a subspace independent of the input features as:

$$d_\omega\left(H^{(l)}\right) < \epsilon, \forall l \geq L. \tag{10}$$

Here $d_\omega(\cdot)$ calculates the distance between the input matrix and the subspace ω. In particular, the definition of subspace and ϵ-smoothing produces an ϵ-smoothing layer which engages a minimal value of layers to satisfy Eq. (10) as:

$$l^*\left(\omega, \epsilon\right) := min_l\left\{d_\omega\left(H^{(l)}\right) < \epsilon\right\}. \tag{11}$$

However, it is challenging to perform analysis based on the ϵ-smoothing layer; therefore, a final definition called the relaxed ϵ-smoothing layer proved to be an upper bound of l^* was introduced as:

$$\hat{l}\left(\omega, \epsilon\right) = \left\lceil \frac{log\left(\epsilon / d_\omega(\mathbf{X})\right)}{log \delta \iota} \right\rceil, \tag{12}$$

where $\lceil \cdot \rceil$ calculates the input ceil, δ is the supremum of the singular filter values on all layers and ι is the second largest eigenvalue of $\hat{\mathbf{A}}$ in addition with $\hat{l} \geq l^{*4}$. Note that $\hat{\mathbf{A}}$ is replaced with $\mathcal{H}_\varphi\left(\mathbf{X}, \mathbf{A}\right)$ in Eq. (9) for propagation and training. In conclusion, the authors [74] demonstrate the existence of ϵ-smoothing in the deep GCN; however, they did not address it. DropEdge eliminates the ϵ-smoothing problem by slowing down the convergence of over-smoothing when node connections are reduced. Secondly, DropEdge measures the amount of information loss between the interval in the original space N and the converging subspace M. A larger interval implies intense information loss. Therefore, DropEdge can increase the converging subspace dimension, while reducing information loss.

3.1.2. AdaEdge

The main objective of AdaEdge [26] is to conduct a systematic and quantitative investigation of the over-smoothing situation in GNN. The authors found that the critical element behind over-smoothing is the information-to-noise ratio influenced by the graph topology, which led to proposing two quantitative metrics; Mean Average Distance (MAD) and the gap of MAD values (MADGap). AdaEdge measures the smoothness of the graph representation using the cosine distance between each pair of nodes. To calculate the cosine distance, the authors formulate the adjacency matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ as the distance matrix for $\mathbf{X}$ in the form of:

$$\mathbf{A}_{ij} = 1 - \frac{\mathbf{X}_{i,:} \cdot \mathbf{X}_{j,:}}{\left|\mathbf{X}_{i,:}\right| \cdot \left|\mathbf{X}_{j,:}\right|} i, j \in \left[1, 2, \ldots, n\right], \tag{13}$$

where $\mathbf{X} \in \mathbb{R}^{n \times h}$ is the graph representation matrix with term h as hidden size, and $\mathbf{X}_k, :$ is the kth row of $\mathbf{X}$. This generates the possibility of further filtering the target node pairs with element-wise multiplication of $\mathbf{A}$ as:

$$\mathbf{A}^{tgt} = \mathbf{A} \otimes M^{tgt}. \tag{14}$$

Here, $M^{tgt} \in \{0, 1\}^{n \times n}$ is a mask matrix, notably, $M_{ij}^{tgt} = 1$ if the node pair (i, j) is the target node, otherwise $M_{ij}^{tgt} = 0$. More understandably, the nonzero values along each row in the target node pairs $\mathbf{A}^{tgt}$ can be used to compute the average distance as:

$$\bar{\mathbf{A}}_i^{tgt} = \frac{\sum_{j=0}^{n} \mathbf{A}_{ij}^{tgt}}{\sum_{j=0}^{n} \ell\left(\mathbf{A}_{ij}^{tgt}\right)}, \tag{15}$$

where, $\ell(x) = 1$ if $x > 0$, else 0. Therefore, given the target node pairs, the MAD values are estimated by averaging the non-zero values in $\bar{\mathbf{A}}_i^{tgt}$ as:

$$\text{MAD}^{tgt} = \frac{\sum_{j=0}^{n} \bar{\mathbf{A}}_{ij}^{tgt}}{\sum_{j=0}^{n} \ell\left(\bar{\mathbf{A}}_{ij}^{tgt}\right)}, \tag{16}$$

The authors measure the quality of the message obtained from the nodes by expressing the information-to-noise ratio as the balance of intra-class node pairs in all reference node pairs that have relations via the GNN model. They concluded that an increase in the network layer, where there is a small information-to-noise ratio, causes an interaction between high-order neighbors to bring too much noise and vice versa. Hence, it weakens the valuable information, which is why the issue of over-smoothing. Therefore, Chen et al. [26] further proposed to compute the gap of MAD values distinguishing remote and neighboring nodes to evaluate over-smoothness.

$$\text{MAD}_{Gap} = \text{MAD}^{rmt} - \text{MAD}^{neb}, \tag{17}$$

where, MAD^{rmt} and MAD^{neb} are the MAD values of the remote and neighboring nodes respectively in the graph topology. Based on these analysis, a MADGap-based regularizer (MADReg) objective is defined on top of AdaEdge as;

$$\mathcal{L} = \sum -llogp\left(\hat{l}|\mathbf{A}, \mathbf{X}, \Theta\right) - \chi \text{MAD}_{Gap}. \tag{18}$$

Here, l and $\hat{l}$ are the golden and predicted labels of the nodes, Θ denotes the model parameter, and χ is the regularization coefficient loss set to be consistent with the cross-entropy loss.

3.1.3. GAUG-O

The intricate non-Euclidean structure of graphs restricts the possible manipulation operations in graph-level data augmentation. Given graph structure, neural edge predictors can adequately encode class-homophilic structure to facilitate intra-class edges and downgrade inter-class edges to improve performance in GNN-based node classification using edge prediction. The objective of GAUG-O [7] includes two steps: *(1)* to obtain edge probabilities for all potential and existing edges in $\mathcal{G}$ via an edge predictor function where the edge predictor can generally be adjusted and replaced with an appropriate approach due to its flexibility. *(2)* Add/remove new or existing edges utilizing the predicted edge probabilities to construct a modified graph $\mathcal{G}_m$ to serve as input to a GNN node-classifier. With this knowledge, an edge predictor is defined as any model with graph input φ_{ep} : $\mathbf{A}, \mathbf{X} \rightarrow \mathcal{Z}$, where $\mathcal{Z}$ is an output edge probability matrix. Here, $\mathcal{Z}_{ab}$ denotes the likelihood of an edge between nodes a and b. The GAUG-O neural edge predictor produces a different structure sampled adjacency matrix as:

$$\mathbf{A}'_{ij} = \left\lceil \frac{1}{1 + e^{\left(\log P_{ij} + G\right)/\gamma}} + \frac{1}{2} \right\rceil, \tag{19}$$

where $P_{ij} = \alpha M_{ij} + (1 - \alpha)\mathbf{A}_{ij}$, γ is the temperature of the Gumbel-Softmax distribution, $G \sim Gumbel(0, 1)$ is a random variable of Gumbel and α is a hyperparameter mediating the influence of the edge predictor on the original graph. For these scenarios, it is clear that the graph autoencoder (GAE) [76] serves as a suitable module for the edge predictor due to its simplicity and competitive performance. In particular, the two-layer GAE GCN encoder and an inner product decoder are defined as follows:

$$\mathcal{M} = \lambda\left(\mathcal{Z}\mathcal{Z}^T\right). \tag{20}$$

Here, $\mathcal{Z} = \varphi_{GCL}^{(1)}\left(\mathbf{A}, \varphi_{GCL}^{(0)}\left(\mathbf{A}, \mathbf{X}\right)\right)$, where $\mathcal{Z}$ represents the hidden embedding learned by the encoder, and λ is an activation function. GAUG-O handles the arbitrary deviation from the original graph adjacency of the edge predictor to derive an adjacency **P** by interpolating the predicted $\mathcal{M}$ with the original **A**. Finally, with Bernoulli sampling at each edge, the adjacency of the graph variant $\mathbf{A}'$ is derived during the edge sampling phase. The graph variant $\mathbf{A}'$ is then processed along with the node features **X**, therefore, a backpropagate with joint node classification loss and edge prediction loss can be defined as:

$$\mathcal{L} = \mathcal{L}_{nc} + \chi \mathcal{L}_{ep}, \tag{21}$$

where, χ is the regularization coefficient to weight the reconstruction loss, $\mathcal{L}_{nc} = \eta_{CE}\left(\hat{y}, y\right)$ - the predicted and ground-truth node labels, $\mathcal{L}_{ep} = \eta_{BCE}\left(\sigma\left(\varphi_{ep}(\mathbf{A}, \mathbf{X})\right), \mathbf{A}\right)$, and η_{CE}/η_{BCE} denotes the standard (binary) cross-entropy loss. The $\mathcal{L}_{nc}$ together with $\mathcal{L}_{ep}$ hypothetically helps to control possible tasks-irrelevant edges in prediction performance during training.

3.1.4. GCA

GCA [13] engages a joint adaptive GDA strategy at the topology and attribute level of the node, i.e., edge perturbation and attribute masking by specifying essential edges and feature dimensions using prominent criteria. The approach provides different contexts for nodes in diverse views and representations consistent with specialized perturbations learned for various graph-level data [14], thus fostering optimization of the contrastive objective.

Given $\mathcal{Q}$ as the set of all possible augmentation functions, the authors sampled two stochastic augmentation functions $q \sim \mathcal{Q}$ and $q' \sim \mathcal{Q}$, in the GCN model, to generate two graph views $\bar{\mathcal{G}} = q\left(\bar{\mathcal{G}}\right)$ and $\bar{\mathcal{G}}' = q'\left(\bar{\mathcal{G}}\right)$ with two-node embeddings in view as $\mathcal{U} = \mathcal{H}\left(\bar{\mathbf{X}}, \bar{\mathbf{A}}\right)$ and $\mathcal{U}' = \mathcal{H}\left(\bar{\mathbf{X}}', \bar{\mathbf{A}}'\right)$, where $\mathcal{H}\left(\cdot\right)$ remains the perturbation function of the characteristic for the characteristic $\bar{\mathbf{X}}*$ and adjacent matrices $\bar{\mathbf{A}}*$. The goal is to score the agreement between positive and negative pairs; hence the model can be trained using the InfoNCE objective [77] for each positive pair (u_i, v_i) as:

$$\mathcal{L}_{\text{InfoNCE}} = \log \frac{e^{o(u_i, v_i)/\gamma}}{e^{o(u_i, v_i)/\gamma} + \sum_{i=1} e^{o(u_i, v_i)/\gamma} + \sum_{i=1} e^{o(u_i, v_i)/\gamma}}. \tag{22}$$

Here, γ remains the temperature parameter, and $o\left(u_i, v_i\right) = \cos_{sim}\left(\lambda\left(u\right), \lambda\left(v\right)\right)$, where $\cos_{sim}\left(\cdot\right)$ denotes the cosine similarity, and $\lambda\left(\cdot\right)$ is a non-linear projection. Therefore, the overall objective of the two symmetric generated views is maximized as the average over all positive pairs by:

$$\mathcal{L} = \frac{1}{2N} \sum_{i=1}^{N} \left[\mathcal{L}_{\text{InfoNCE}}\left(u_i, v_i\right) + \mathcal{L}_{\text{InfoNCE}}\left(u_i, v_i\right)\right] \tag{23}$$

Inspired by GRACE [78], GCA considered a direct means of corrupting input graphs by randomly eliminating edges in the graph. It simulates an altered subset $\mathcal{E}_i$ in $\mathcal{G}_i = \left(\mathcal{V}_i, \mathcal{E}_i\right)$ from the original set $\mathcal{E}_i$ in $\mathcal{G} = \left(\mathcal{V}, \mathcal{E}\right)$ with probability function:

$$P\left(\left(u, v\right) \in \mathcal{E}_i\right) = 1 - p^{e}_{uv}, \tag{24}$$

where, $\left(u, v\right) \in \mathcal{E}$ and p^{e}_{uv} denote the probability of dropping $\left(u, v\right)$ and $\mathcal{E}_i$ serves as the edge set in the generated sample. In particular, p^{e}_{uv} is an influential link structure of the edge $\left(u, v\right)$, allowing the augmentation operation to have more potential to corrupt trivial edges while preserving the essential connective structures during augmentation.

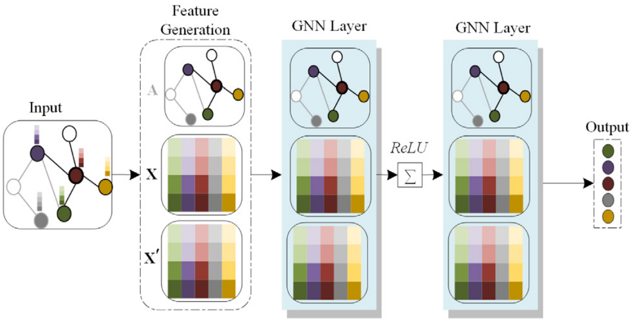

Fig. 3. Schematic diagram of a two-layer GNN to illustrate feature representation generation.

3.2. Feature-level augmentation

Unlike the topology-level augmentation function in Eq. (8), the feature-level augmentation function can be defined as:

$$\mathbf{X}' = \mathcal{H}^{\varphi}(\mathbf{X}, \mathbf{A}) \tag{25}$$

where, $\mathcal{H}^{\varphi}\left(\cdot\right)$ is the feature representation function. Although research argues that removing nodes decreases the data available during node classification tasks, additional nodes pose a challenge in labeling by assigning features and links of new nodes. However, it remains a fact that emphasizing the influential nodes by placing a probability on the trivial ones can enhance the classification performance. Feature-level augmentation, which can be termed feature representation generation (Fig. 3), mainly exploits the perturbation of nodes' attributes guided by adversarial training [6,38,79].

Global information for GNNs (G-GNN) [41], which preserves the learned global information, is known for its performance, easy implementation, and theoretical understanding. Summarily, G-GNN (a.k.a. global preservative) constructs the global information as the global structure and global attribute features to each node and pre-trained them for a final parallel GNN-based model to learn distinct characteristics from the pre-trained and original features. Local Augmentation for GNN (LA-GNN) [42] (a.k.a. local preservative) preserves the locality of node representations by their subgraph structure. Node-Parallel Augmentation (NodeAug) [80], another feature-level augmentation technique, offered three distinct GDA methods by (1) adjusting node attributes, (2) graph structure, and (3) introducing a subgraph mini-batch training for resourceful execution. The comprehensive idea is that adversarial training harms generalization, hence not performing well in model accuracy [81,82]. However, attention has been focused on utilizing adversarial perturbations to expand datasets and combat overfitting. Furthermore, despite its success in language [83–85] and vision [86], it remains ambiguous how to use adversarial augmentation to efficiently improve the accuracy of GNN.

The following subsection concentrates on NodeAug, FLAG, G-GNN, and LA-GNN. Most importantly, FLAG (an adversarial augmentation method) engaged a large-scale dataset for experimental analysis, which is vital in inductive learning.

3.2.1. NodeAug

Graph data augmentation for different nodes influence each other, leading to uncontrollable magnitudes and causing changes in ground-truth labels. The NodeAug technique [80] initiates a parallel system for each node to perform GDA tasks to help intercept such unwanted results. In addition, NodeAug normalizes the model prediction of both labeled and unlabeled nodes to be uniform with respect to modifications yielded by augmentation for effectiveness. The authors offered three distinct GDA methods by (*1*) adjusting the node attributes, and (*2*) the graph structure, and (3) introducing a subgraph mini-batch training for resourceful execution. That is, to achieve better generalization [87] and decrease computational costs, they proposed the subgraph mini-batch training approach. Given a subgraph $\mathcal{E}_i$ in $\mathcal{G}_i = \left(\mathcal{V}_i, \mathcal{E}_i\right)$ of the graph representation $\mathcal{E}_i$ in $\mathcal{G} = \left(\mathcal{V}, \mathcal{E}\right)$, generally GCNs offer a conditional outcome distribution of node i as:

$$p\left(y|\mathcal{G}_i = \left(\mathcal{V}_i, \mathcal{E}_i\right), \left\{x_j\right\}_{j\in v_i}, \Theta\right), \tag{26}$$

where y is the class label, x_j indicates the attributes of node j and Θ denotes the model parameter. Clearly, the receptive field $\mathcal{G}_i$ is smaller than $\mathcal{G}$. Therefore, $\mathcal{G}_i$ and $\left\{x_j\right\}_{j\in v_i}$ subsidized the prediction of node i as demonstrated in Eq. (26) instead of a full-batch training technique [7], which accumulates the entire graph at each iteration. Subgraph $\mathcal{G}_i$ is the result of the k-hop neighborhood around the selected center node i in a kth layer of GCN with row-normalization [41]. Therefore, to increase i, it would be more efficient to modify $\mathcal{G}_i$ and $\left\{x_j\right\}_{j\in v_i}$ than to alter x_i as $\mathcal{G}_i$ and $\left\{x_j\right\}_{j\in v_i}$ greatly influence input features of other nodes and x_i serves as the only part of the input characteristics. Due to such a significant influence, the authors proposed a 'parallel universes' GDA strategy for separate and various individual nodes, changing the input features without altering the class labels.

Assuming that the attributes of the $\hat{\mathcal{G}}$ and $\left\{\hat{x}_j, j \in v\right\}$ nodes are enhanced from the graph $\mathcal{G}$ and the node i, the model minimized a consistency loss similar to UDA [70] as:

$$\mathcal{L}_c = \frac{1}{\left|\mathcal{V}\right|}\mathcal{D}_{KL}\left(p\left(y|\mathcal{G}_i, \left\{x_j\right\}_{j\in v_i}, \tilde{\Theta}\right)||p\left(y|\hat{\mathcal{G}}_i, \left\{\hat{x}_j\right\}_{j\in v_i}, \Theta\right)\right), \tag{27}$$

where $\hat{\mathcal{G}}_i$ denotes the subgraph corresponding to its receptive field, $\mathcal{D}_{KL}\left(\cdot||\cdot\right)$ is the Kullback–Leibler divergence (KL divergence) [88] between the actual and observed probability distribution, and $\tilde{\Theta}$ is a fixed copy of the parameter Θ. However, UDA engaged image data, while NodeAug focuses on graph-level data augmentation. This knowledge is embedded into the model to perform extraction, facilitating minimization of the consistency loss on both original and augmented graphs. When extraction is performed on only the labeled nodes of the original graph, the authors tuned Eq. (27) for a supervised classification loss as:

$$\mathcal{L}_s = \frac{1}{\left|\mathcal{V}_L\right|}\sum_{i\in V_L}\eta_{CE}\left(y_i^*, p\left(y|\mathcal{G}_i = \left(\mathcal{V}_i, \mathcal{E}_i\right), \left\{\hat{x}_j\right\}_{j\in v_i}, \Theta\right)\right). \tag{28}$$

Here, V_L is the set of all the labeled nodes, $\eta_{CE}\left(\cdot, \cdot\right)$ denotes the cross-entropy function utilized by previous GCN models, and y^* is a one-hot label of node i. Then the overall objective function of NodeAug to consistently classify both the labeled and unlabeled nodes before and after augmentation is computed as:

$$\mathcal{L} = \mathcal{L}_s + \chi\mathcal{L}_c, \tag{29}$$

where, χ is the regularization coefficient to control the influence of consistency loss with a default of 1 to balance $\mathcal{L}_s$ and $\mathcal{L}_c$. During augmentation, the authors engaged in three strategies: (1) replacing attributes; (2) removing edges; and (3) adding edges for GDA (Section 2.2).

3.2.2. FLAG

The FLAG [6] method iteratively augment node features with gradient-based adversarial perturbations for better performance during training. Following Eq. (25), the perturbation function in FLAG is defined as:

$$\mathcal{H}^{\varphi}(\mathbf{X}, \mathbf{A}) = \mathbf{X} + \delta \tag{30}$$

where perturbation δ is updated iteratively during the adversarial training phase. In particular, FLAG adversarial perturbation is considered a strong candidate method for enhancing input features based on the highly prominent out-of-distribution phenomenon of graph domain data [41]. FLAG has a two-way approach; (*1*) the problem of attacking a given neural network, (*2*) the problem of training a robust classifier using adversarial training strategies. Especially, their principle views the saddle point issue as forming the outer minimization and inner maximization problem. Thus, while inner maximization strives to craft an adversarial perturbation of the data point x, outer minimization seeks to find the best parameters to minimize the "adversarial loss" given the inner maximization problem. Therefore, the creation of pseudo-data points given samples from the distribution D, is expressed as:

$$\min_{\vartheta} E_{(x,y)} \sim D\left[\max_{\|\delta\|_{p\leq\varepsilon}}\mathcal{L}\left(f_{\vartheta}\left(x+\delta\right), y\right)\right] \tag{31}$$

where y is the label, $\|\delta\|_p$ is some norm distance metrics, ε is a perturbation parameter and $\mathcal{L}\left(\cdot\right)$ is an objective function. Previous work such as the Fast Gradient Sign Method (FGSM) [89] and multiple variations of FGSM [90] engaged an l_∞-bounded constraint attack and formulated an adversarial data point as:

$$x + \varepsilon\text{sign}\left(\Delta_x\mathcal{L}\left(\vartheta, x, y\right)\right). \tag{32}$$

However, the multi-step variant is a better practical adversary approach [91]. It essentially engages the projected gradient descent (PGD) in the negative loss function for the inner maximization. To satisfy the perturbation δ in Eq. (30), the estimate of inner maximization under an l_∞-norm constraint of the PGD algorithm is formulated as follows:

$$\delta_{t+1} = \prod_{\|\delta\|\infty\leq\varepsilon}\left(\delta_t + \alpha\cdot\text{sign}\left(\Delta_\delta\mathcal{L}\left(f_\vartheta\left(x+\delta_t\right), y\right)\right)\right), \tag{33}$$

where $\prod_{\|\delta\|\infty\leq\varepsilon}$ performs projection onto the ε-ball in the l_∞-norm. The process generates the most vicious noise δ_M for maximum robustness by iteratively engaging multiple loops M times via end-to-end forward and backward passes to serve as input features.

Unraveling Min–Max optimization with PGD is effective; however, practice slows the model training M times due to the model weights ε updating once in the final δ_M. The ultimate purpose of FLAG is to improve the diversity and quality of adversarial perturbations by leveraging multi-scale augmentations [92] to influence the generalizing capability comprehensively. With 'free' adversarial training [35], the process simultaneously produces a perturbation that updates the model parameter ε in the exact backward pass in parallel, while estimating the gradient for the perturbation δ with effectively zero extra cost. In particular, a mini-batch M is triggered in a row to simulate inner maximization in Eq. (31), resulting in competitive accuracy with fewer epochs. The optimization step is formulated as follows:

$$\vartheta_{i+1} = \vartheta_i - \frac{\tau}{M}\sum_{t=1}^{M}\Delta_\vartheta\mathcal{L}\left(f_\vartheta\left(x+\delta_t\right), y\right), \tag{34}$$

where τ is the learning rate and δ_1 is uniform noise. To achieve multi-scale augmentation, $\mathbf{X}$ is augmented with additive perturbations $\delta_1 : M$, using the "free" algorithm approach instead of a single perturbation δ_M with scaling $M\alpha$ in PGD. This approach offers space for each perturbation budget to control a maximum scale of $m\alpha$, $m \in \left\{1, \ldots, M\right\}$, thus dramatically adding to the multiplicity of the augmented data. However, the "free" algorithm encounters a suboptimal Min–Max optimization problem during the batch-replay procedure. Therefore, the perturbation estimated for the target maximization ϑ_t is used to optimize

$\vartheta_t + 1$ relatively to ϑ_t robustly. FLAG alleviates such instances by exploiting the acquired gradient's "by-product" from the gradient ascent step on δ. Instead of updating ϑ directly, the gradients $\Delta_\vartheta \mathcal{L}$ are accumulated and are later applied to the model parameters.

3.2.3. G-GNN

Global information for **GNN**s [41] constructs the global information as (*1*) the global structure and (*2*) the global attribute features to each node and pre-trained them. Finally, a proposed parallel GNN-based model learns distinct characteristics from the pre-trained global and original features via a hidden state by iteratively aggregating the neighborhood attributes. Mathematically, the two-layer GCN using hidden states is defined as:

$$H = \lambda\left(\hat{\hat{\mathbf{A}}}\mathbf{X}W_0\right)W_1, \tag{35}$$

where, each row in H represents the final hidden states of a node and each row corresponds to a prediction category. W_0 and W_1 are the trainable weight matrices.

From Eq. (25), G-GNN first maximizes the probability source context pairs, by minimizing the global structure feature matrix with dimension vector d_s using the objective function:

$$\prod_{v \in V} \prod_{u \in N(v)} \frac{\exp\left(X_v^{(s)} \cdot X_u^{(s)}\right)}{\sum_{k \in V} \exp\left(X_v^{(s)} \cdot X_k^{(s)}\right)}, \tag{36}$$

where, v_i denotes the ith row of the global structure feature vector. In particular, neighborhood nodes within the convinced distance to the source node v are reflected as context nodes v, which are represented as $N_{(v)} \subset V$ in node sequences. Second, global attribute features are obtained by maximizing the probability of context attributes.

Hypothetically, if the context attributes can be retrieved from the source node, the learning approach has already preserved vital information. Therefore, for each sampled context node $u \in N_{(v)}$ some attributes of u are resampled as context attributes v. Given the global attribute feature matrix $\mathbf{X}^{(a)} \in \mathbb{R}^{n \times d_a}$ and $S \in \mathbb{R}^{d_a \times |U|}$, as the parameters for the prediction of attributes, the objective function is minimized as follows:

$$\prod_{v \in V} \prod_{a \in C_{sam}(v)} \frac{\exp\left(X_v^{(a)} \cdot S_a\right)}{\sum_{k \in U} \exp\left(X_v^{(a)} \cdot S_k\right)}, \tag{37}$$

where, $C_{sam}(v)$ is the context attributes sampled from v, and U is the set of all attributes where $|U| = \mathit{shape}(\mathbf{X})$ in Eq. (25) v_i is the ith row of the global attributes feature vector. Next, we have a parallel model with GNN kernels to learn from the input matrices; heterogeneous features $\mathbf{X}^{(s)}$, $\mathbf{X}^{(a)}$ and the original $\mathbf{X}$. In particular, the G-GNN methods are viewed as a transform from the original $\mathbf{X}$ and $\mathbf{A}$ to the final hidden states H_{final} as:

$$\Theta\left(\mathbf{X}, \mathbf{A}\right) : \mathbf{X}, \mathbf{A} \rightarrow H_{final} \tag{38}$$

where, $\Theta\left(\cdot\right)$ is the learning kernel of the G-GNNs and each kernel satisfies Eq. (35). Thus, these matrices serve as input to a parallel GNN based model, to learn their respective hidden states $H^{(1)}$, $H^{(2)}$ and the original $H^{(0)}$ to a final hidden state H_{final}. Observably, feature matrices $H^{(1)}$ and $H^{(2)}$ and are highly correlated since the amplitude varies in dimension on pre-trained. Therefore, a transformation $r = \frac{r - \mu(r)}{\sigma(r)}$ where, r denotes each row in $H^{(1)}$ or $H^{(2)}$, $\mu\left(\cdot\right)$ is the mean and $\sigma\left(\cdot\right)$ represents the standard deviation for normalization. To fulfill Eq. (35), the kernels to learn from the three feature matrices are formulated as;

$$\begin{aligned} H^{(0)} &= \Theta\left(\mathbf{X}, \mathbf{A}\right) \\ H^{(1)} &= \Theta\left(\mathbf{X}^{(s)}, \mathbf{A}\right) \\ H^{(2)} &= \Theta\left(\mathbf{X}^{(a)}, \mathbf{A}\right). \end{aligned} \tag{39}$$

Finally, the overall hidden state matrix to linearly combine the three obtained hidden state matrices can be formulated as follows:

$$H_{final} = H^{(0)} + \chi H^{(1)} + \eta H^{(2)}, \tag{40}$$

where, χ and η are regularization coefficients between 0 and 1 to influence the tuning of the weight of the pre-trained features for optimization.

3.2.4. LA-GNN

The **L**ocal **A**ugmentation for **GNN** (LA-GNN) [39], approach is motivated by three-folds: *(1)* prior feature-level augmentation methods focus on global information, escaping the informative neighborhood; *(2)* the representation distributions of the neighbors are closely related to the central node, creating immense space for feature-level perturbation; and *(3)* preserving locally learned information leads to combatting over-smoothing [18,19, 93]. With respect to Eq. (25), LA-GNN uses a GNN classification estimator to model a conditional distribution as:

$$\mathcal{H}^{\varphi}\left(\mathbf{X}, \mathbf{A}\right) = P_\vartheta\left(y|\mathbf{X}, \mathbf{A}\right). \tag{41}$$

Here, ϑ denotes the model parameter and y is the class label. Therefore, given $\left\{\mathbf{X}, \mathbf{A}, y\right\}$ as training samples, the authors estimated the parameter ϑ with Maximum Likelihood Estimation (MLE), using optimization function:

$$\max \prod_{k \in K} P_\vartheta\left(y_k|\mathbf{X}, \mathbf{A}\right), \tag{42}$$

where K is a set of node indices of the dataset with visible labels during semi-supervised training. LA-GNN is treated as a node feature generation model, producing additional $\bar{\mathbf{X}}$ generated features that produce a new model $P_\vartheta\left(y, \bar{\mathbf{X}}|\mathbf{X}, \mathbf{A}\right)$. Then to benefit from decomposition, a marginalized likelihood P_ϑ over $\bar{\mathbf{X}}$ is optimized as:

$$\max \prod_{k \in K} \int_{\bar{\mathbf{X}}} P_\vartheta\left(y_k, \bar{\mathbf{X}}|\mathbf{X}, \mathbf{A}\right). \tag{43}$$

P_ϑ can further be decomposed with Bayesian tractability as the outcome of two posterior probabilities of $P_\vartheta\left(y|\bar{\mathbf{X}}, \mathbf{X}, \mathbf{A}\right)$ and $\mathcal{Q}_\phi$ $\left(\bar{\mathbf{X}}|\mathbf{X}, \mathbf{A}\right)$ as:

$$P_{\vartheta\phi}\left(y_k|\bar{\mathbf{X}}, \mathbf{X}, \mathbf{A}\right) := P_\vartheta\left(y_k|\bar{\mathbf{X}}, \mathbf{X}, \mathbf{A}\right) \cdot \mathcal{Q}_\phi\left(\bar{\mathbf{X}}|\mathbf{X}, \mathbf{A}\right), \tag{44}$$

where, the first indicates the probabilistic distributions estimated by the downstream GNN, and the latter denotes the feature-level generator (augmentation) which happens to be a conditional variational auto-encoder (CVAE) [91,92]. The advantage of decomposition is that it allows the model to decouple the generator $\mathcal{Q}_\phi$ and the predictor P_ϑ, allowing the generator to generalize to other downstream GNN tasks efficiently, therefore being superior to a single predictor $P_\vartheta\left(y_k|\mathbf{X}, \mathbf{A}\right)$. Therefore, with additional generated $\bar{\mathbf{X}}$ feature samples from the conditional distribution $\mathcal{Q}_\phi$, the training process can optimize $P_\vartheta\left(y_k|\bar{\mathbf{X}}, \mathbf{X}, \mathbf{A}\right)$ once the generator is sufficiently trained.

In particular, given $\left\{\mathbf{X}_{j|j \in N_i}, \mathbf{X}_i\right\}$, the authors engaged in a naive solution to learn a single distribution for all neighbors that employ the MLE scheme during feature generation by solving the optimization problem.

$$\max_{\kappa} \sum_{j \in N_i} \log p_{\kappa}\left(\mathbf{X}_j|\mathbf{X}_i\right) = \max_{\kappa} \log \prod_{j \in N_i} p_{\kappa}\left(\mathbf{X}_j|\mathbf{X}_i\right), \tag{45}$$

where p_{κ} indicates the feature augmentation parameter for the whole neighbor. The consequence is that the differences between neighbors are ignored in such a solution, generating extreme noise. However, the authors overcome these limitations presuming that each neighbor benefits from various conditional distributions and used CVAE to learn the distribution of a latent random variable z_j and an existing conditional distribution $p\ \left(\ \cdot|\mathbf{X}_i, z_j\right)$. Particularly with z_j and $p\ \left(\ \cdot|\mathbf{X}_i, z_j\right)$, one can compute $\mathbf{X}_i \sim p\left(\mathbf{X}|\mathbf{X}_i, z_j\right)$ for $\mathbf{X}_{j|j \in N_i}$ to generate augmented features $\bar{\mathbf{X}}$, which can then be trained on $P_{\vartheta}\left(y_k|\bar{\mathbf{X}}, \mathbf{X}, \mathbf{A}\right)$ to improve the predictor P_{ϑ} performance.

The following previous work [94,95] $\log p_{\kappa}\left(\mathbf{X}_j|\mathbf{X}_i\right)$ is written with latent variables z to estimate CVAE optimization. Finally, given the encoder and decoder preambles with two-layer Multilayer Perceptron (MLP) each, an estimation is made to satisfy the Evidence Lower Bound (ELBO) as:

$$\begin{aligned} \mathcal{L}\left(\mathbf{X}_j, \mathbf{X}_i; \kappa, \zeta\right) = {} & \mathcal{D}_{KL}\left(\mathcal{J}_{\zeta}\left(z|\mathbf{X}_j, \mathbf{X}_i\right)\big|\big|P_{\kappa}\left(z|\mathbf{X}_i\right)\right) \\ & + \int \mathcal{J}_{\zeta}\left(z|\mathbf{X}_j, \mathbf{X}_i\right) \log P_{\kappa}\left(\mathbf{X}_j|\mathbf{X}_i, z\right) dz, \end{aligned} \tag{46}$$

where $\left\{\kappa, \zeta\right\}$ represents the variational and generative parameters derived from ϕ in the CVAE model $\mathcal{Q}_{\phi}\left(\bar{\mathbf{X}}|\mathbf{X}, \mathbf{A}\right)$, whereby $\mathcal{J}_{\zeta}\left(z|\mathbf{X}_j, \mathbf{X}_i\right)$ denotes the encoder and $P_{\kappa}\left(\mathbf{X}_j|\mathbf{X}_i, z\right)$ indicates the decoder, and $\mathcal{D}_{KL}\left(\cdot\big|\big|\cdot\right)$ is the KL divergence. The generated feature matrix is final optimized with MLE by a downstream GNN model with respect to Eq. (44) as:

$$P_{\vartheta}\left(y_k|\bar{\mathbf{X}}, \mathbf{X}, \mathbf{A}\right) \infty - \bar{\mathcal{L}}\left(\bar{\mathbf{X}}, \mathbf{X}, \mathbf{A}, \vartheta\right), \tag{47}$$

where, $\bar{\mathcal{L}}\left(\cdot\right) = -\sum_{k \in T} \sum_{f=1}^{C} y_{kf} \in \left(softmax\left(\Theta\left(\cdots\right)\right)_{kf}\right)$. In particular, $\Theta\left(\cdots\right)$ denotes the model parameter that makes up $\bar{\mathbf{X}}, \mathbf{X}, and\ \mathbf{A}$.

4. Experiments

In this section, we perform an analysis on the most publicly used graph datasets based on the algorithms discussed for GDA.

4.1. Dataset

We utilize eleven public graph datasets Cora, Citeseer, Pubmed, ogbn-products, ogbn-proteins, ogbn-arxiv, Wiki-CS Amazon-Computers, Amazon-Photo, Coauthor-CS, and Coauthor-Physics. Specifically, we used these datasets to cover all ranges of graph datasets from small and medium to large. Table 3 shows the statistics of the datasets grouped in various ranges. As observed, the ogbn products contain more than two million nodes and sixty-one billion edges. The ogbn-proteins contain a hundred and thirty-two thousand nodes and thirty-nine billion edges; hence, we denote the OGB dataset as large-scale. Cora, Citeseer, and Pubmed are grouped under a small/standard dataset, and the rest under medium. In particular, the small- and medium-datasets are for transductive learning, and the large-scale datasets are for an inductive setting. The details of these datasets are described below.

I. **Cora**, **Citeseer**, and **Pubmed** [48] are standard citation network datasets with nodes representing documents and edges denoting citation links between documents. The node feature corresponds to each document's normalized bag-of-words feature vector, coordinating a node class label to an academic topic. The direction of edges is omitted since a citation is assumed to impact the predictions of two associated documents equally. In particular, the Cora dataset comprises seven categories of 2708 machine learning (ML) publications, while the Citeseer dataset comprises 3327 scientific papers in six categories. Respective document in Cora and Citeseer is denoted by a one-hot vector exhibiting the presence/absence of a dictionary word. However, the Pubmed data set comprises about 19 717 diabetes-related publications, where each document is described by a term frequency-inverse document frequency (TF-IDF) vector.

Table 3
Statistics of the utilized datasets.

	Dataset	Nodes	Edges	Features	Classes
Small	Cora	2708	5429	1433	7
	Citeseer	3327	4732	3703	6
	Pubmed	19 717	44,338	300	3
Medium	Wiki-CS	11,701	216,123	300	10
	Amazon-Comp	13,381	245,778	767	10
	Amazon-Photo	7487	119,043	745	8
	Coauthor-CS	18,333	81,894	6805	67
	Coauthor-Phy.	34,493	247,962	8415	5
Large	ogbn-products	2,449,029	61,859,140	100	47
	ogbn-proteins	132,534	39,561,252	8	2
	ogbn-arxiv	169,343	1,166,243	128	40

II. **Wiki-CS** [96] is a reference network in computer science built from Wikipedia. The nodes represent various articles, and the edges are links that connect the articles. The labeled nodes consist of ten classes, each depicting a field branch with node features in each article computed as the average of pre-trained GloVe [97] word embeddings.

III. **Amazon-Computers** and **Amazon-Photo** [21] are two copurchase network connections scraped and built from Amazon. In this network, nodes represent items and two frequently purchased items are linked. Each node is labeled with a category based on a sparse bag-of-words feature encoding of product assessments.

IV. **Coauthor-CS** and **Coauthor-Physics** [21]; these are two academic networks that comprise co-authorship of the Microsoft Academic Graph KDD Cup 2016 challenge. In these networks, nodes stand for authors, and edges denote coauthorship relationships. In simple terms, two nodes are linked if they have coauthored a paper. A sparse bag-of-words feature represents the paper keywords of the authors for each node, and the author's active research field forms the label.

V. The **OGB** datasets [23] are large-scale, with numerous vital graphs for ML tasks. In addition, it covers various domains, from information and social networks to biological networks, molecular graphs, and knowledge graphs. Thus, it is diverse in scale, domains, and task categories. We only engaged ogbn-products, ogbn-proteins, and ogbn-arxiv in the Node-ogbn category of the OGB datasets for this survey.

4.2. Parameter settings

The various parameters involved in this work are set according to the best performance in original works. We apply the standard fixed splits [48] for Cora, Citeseer, and Pubmed, public splits shipped with the Wiki-CS dataset, random split of 80%, 10%, and 10% for the training, validation, and test set, respectively, for Amazon-Computers, Amazon-Photo, Coauthor-CS, and Coauthor-Physics, and the rest from original work for all algorithms.

Table 4
GDA algorithm accuracy performance on different datasets based on the GNN variants: GCN, GAT, and GSAGE. The white background is copied from the original works. The green background is the average accuracy after running 10 experiments on 1000 epochs. The light gray background indicates that there was no source code availability when this experiment was performed. **Bold** for best accuracy performance and * for the overall best on a particular dataset.

GCN architecture									
Datasets/Model	Original	+AdaEdge	+DropEdge	+GAUG-O	+G-GNN	+LA-GNN	+NodeAug	+FLAG	+GCA
Cora	81.5 ± 0.7	82.3 ± 0.8	82.0 ± 0.8	83.6 ± 0.5	83.7 ± 1.8	84.1	84.3 ± 0.5	83.9 ± 0.3	**85.8 ± 0.1**
Citeseer	70.3 ± 0.4	72.8 ± 0.7	71.8 ± 0.2	73.3 ± 1.1	71.3 ± 1.4	72.5	**74.9 ± 0.5**	71.2 ± 0.2	73.6 ± 1.0
Pubmed	79.0 ± 1.3	77.4 ± 0.5	82.9 ± 0.4	OOM	80.9 ± 1.1	81.3	81.5 ± 0.5	82.6 ± 0.7	**83.2 ± 0.2**
ogbn-products	–	–	72.1 ± 0.3	71.1 ± 0.2	71.5 ± 0.3	–	–	–	**82.1 ± 0.3***
ogbn-proteins	72.5 ± 0.4	–	70.3 ± 1.5	71.8 ± 1.1	75.7 ± 0.2	–	–	71.7 ± 0.5	**76.3 ± 1.1**
ogbn-arxiv	71.7 ± 0.3	–	74.3 ± 1.1	72.3 ± 0.5	**73.0 ± 0.1**	–	–	72.0 ± 0.2	72.3 ± 1.0
Wiki-CS	77.2 ± 0.1	–	73.1 ± 0.3	74.4 ± 0.4	77.2 ± 0.0	–	–	77.5 ± 1.6	**78.4 ± 0.1**
Amazon-Comp	81.7 ± 0.7	82.4 ± 1.1	85.4 ± 0.6	85.7 ± 0.4	86.3 ± 0.5	–	–	85.8 ± 0.1	**87.8 ± 0.2**
Amazon-Photo	90.6 ± 0.7	91.5 ± 0.5	89.7 ± 0.1	89.4 ± 0.1	90.1 ± 0.6	–	–	**95.6 ± 0.7***	92.5 ± 0.2
Coauthor-CS	89.8 ± 0.3	90.3 ± 0.4	85.9 ± 0.3	84.6 ± 0.2	**93.5 ± 0.0**	–	–	OOM	93.1 ± 0.0
Coauthor-Phy.	92.8 ± 1.6	93.0 ± 1.1	91.2 ± 0.2	91.8 ± 0.2	94.9 ± 0.7	–	–	OOM	**95.7 ± 0.0**
GAT architecture									
Cora	83.0 ± 0.7	82.0 ± 0.6	81.9 ± 0.6	82.2 ± 0.8	84.6 ± 0.1	83.9	84.8 ± 0.2	83.7 ± 0.5	**85.2 ± 0.7**
Citeseer	72.5 ± 0.7	71.1 ± 0.8	71.0 ± 0.5	71.6 ± 1.1	75.0 ± 0.1	72.3	**75.1 ± 0.4**	74.1 ± 0.2	73.3 ± 1.2
Pubmed	79.0 ± 1.0	76.6 ± 0.2	**82.6 ± 0.0**	OOM	82.2 ± 0.5	OOM	81.6 ± 0.6	79.5 ± 0.9	80.2 ± 0.5
ogbn-products	79.5 ± 0.6	–	79.8 ± 0.1	75.7 ± 0.3	78.3 ± 0.0	–	–	**81.8 ± 0.5**	80.6 ± 0.4
ogbn-proteins	–	–	70.3 ± 1.5	72.4 ± 0.2	75.1 ± 1.0	–	–	–	**75.5 ± 1.4**
ogbn-arxiv	73.7 ± 0.1	–	73.6 ± 0.3	70.4 ± 0.6	71.8 ± 0.0	–	–	73.7 ± 0.1	**74.3 ± 1.1***
Wiki-CS	77.7 ± 0.1	–	78.3 ± 0.2	78.1 ± 0.1	77.3 ± 0.0	–	–	**78.6 ± 0.1***	78.2 ± 0.4
Amazon-Comp	86.9 ± 0.3	81.1 ± 1.6	86.7 ± 0.5	**88.0 ± 1.2***	87.5 ± 0.1	–	–	87.8 ± 0.2	87.5 ± 0.5
Amazon-Photo	92.6 ± 0.4	90.8 ± 0.9	92.2 ± 0.3	92.1 ± 0.4	91.7 ± 0.1	–	–	**93.5 ± 0.2**	92.2 ± 0.2
Coauthor-CS	92.3 ± 0.2	86.6 ± 1.6	**93.7 ± 0.1**	93.3 ± 0.1	92.1 ± 0.2	–	–	OOM	93.0 ± 0.1
Coauthor-Phy.	95.5 ± 0.1	91.4 ± 1.0	95.5 ± 0.1	95.6 ± 0.2	95.3 ± 0.0			OOM	**95.7 ± 0.1**
GSAGE architecture									
Cora	81.3 ± 0.5	81.5 ± 0.6	81.6 ± 0.5	82.0 ± 0.5	83.9 ± 1.1	–	–	83.5 ± 0.7	**85.9 ± 03***
Citeseer	70.6 ± 0.5	71.3 ± 0.8	70.8 ± 0.5	72.7 ± 0.7	71.2 ± 1.2	–	–	**77.1 ± 0.2***	73.2 ± 0.0
Pubmed	81.3 ± 0.5	77.4 ± 0.5	82.8 ± 0.6	OOM	78.9 ± 1.6	–	–	79.5 ± 0.3	**83.7 ± 0.4***
ogbn-products	70.6 ± 0.5	–	**80.4 ± 0.2**	OOM	79.3 ± 0.7	–	–	79.4 ± 0.6	79.0 ± 0.4
ogbn-proteins	**78.6 ± 0.6***	–	71.7 ± 0.3	OOM	75.2 ± 1.3	–	–	76.6 ± 0.8	77.1 ± 1.1
ogbn-arxiv	78.7 ± 0.4	–	71.4 ± 1.8	69.8 ± 0.7	71.7 ± 0.4	–	–	72.2 ± 0.2	**73.8 ± 1.5**
Wiki-CS	76.1 ± 0.0	–	74.5 ± 0.1	73.9 ± 0.6	76.5 ± 0.3	–	–	**77.5 ± 0.1**	75.6 ± 0.2
Amazon-Comp	80.2 ± 1.0	81.1 ± 1.0	85.6 ± 0.1	85.3 ± 0.5	**87.7 ± 0.3**	–	–	87.5 ± 0.1	86.4 ± 0.5
Amazon-Photo	90.1 ± 1.4	90.6 ± 0.5	89.3 ± 0.1	90.1 ± 1.1	**92.7 ± 0.5**	–	–	91.4 ± 0.7	92.2 ± 0.1
Coauthor-CS	90.1 ± 0.4	90.3 ± 0.4	84.5 ± 0.3	OOM	**93.8 ± 0.3***	–	–	OOM	92.3 ± 0.0
Coauthor-Phy.	93.0 ± 0.4	92.7 ± 0.2	91.7 ± 0.2	OOM	**96.1 ± 0.2***	–	–	OOM	94.5 ± 0.0

4.3. Experimental results

This section emphasizes the precise performance of the algorithms on 11 publicly available datasets. At the same time, we consider conducting experiments on datasets not used in the original papers. For instance, FLAG only engaged ogbn-products, ogbn-proteins, and ogbn-arxiv of the Open Graph Benchmark (OGB) datasets. However, we engaged different 9 datasets to test the algorithm's effectiveness by applying the proposed GDA methods (for instance, FLAG) to GCN, GAT, and GSAGE models in a plug-and-play manner. Therefore, to distinguish the performance divergence between them, we detail the accuracy based on the average precision (AP), mean values (%) and standard deviation values in Table 4. The results in white background are directly copied from the original work, while those in green background are after performing experiments on the different datasets. The codes for the implementations were taken from the original work and the Deep Graph Library (DGL)[2] repository. Areas marked gray are papers without codes; however, their practical, theoretical motivations and results are vital for this work. For example, AdaEdge adds some edges and makes node connections sparse, avoiding over-smoothing to some extent when GCN goes deeper. NodeAug took advantage of the attention mechanism in GAT to perform remarkably on the Citeseer dataset in the inductive setting. A critical challenge in GDA is the lack of sufficient theoretical foundation. Unlike other works, LA-GNN provided enough theoretical information to capture its achievement, but the approach lacks sufficient experimental evidence. We report accuracy based on three GNN architectures: GCN, GAT, and GSAGE. The primary objectives to engage these most widely used GNN variants are clearly stated previously in Section (Section 2).

Observing Table 4, we can draw the following conclusions. First, the GDA models have increased the accuracy of the state-of-the-art test by absolute values of 4.20%, 5.50%, and 4.40% on Cora, Citeseer, and PubMed, respectively. On the basis of the results, we can see that GCA performs well across the various datasets when paired with the GNN architectures. Although originally designed for Wiki-CS, Amazon-Computers, Amazon-Photo, Coauthor-CS, and Coauthor-Physics datasets, GCA recorded the highest accuracy on the Cora dataset across various architectures. GCN+GCA is also leading in the accuracy of Pubmed, ogbn-products, ogbn-proteins, Wiki-CS, and Amazon-Computers dataset. +NodeAug leads the accuracies in Citeseer and +G-GNN in ogbn-arxiv. FLAG, originally designed for OGB to address the significant issue of limited data sizes of traditional graph datasets, [23] recorded the highest accuracy on the Amazon-Photo dataset when manipulated with GCN.

In the GAT architecture; ogbn-products, Wiki-CS, and Amazon-Photo benefit from +FLAG as shown in Table 4. The method iteratively increases the features of the nodes with gradient-based adversarial perturbations during training. GAT is unique in automatically learning the importance of each neighbor based on the attention mechanism. Thus, an attention layer helps to transfer the hidden node representations at a layer to new node representations, which estimate the attention coefficients for any pair of nodes. Therefore, it is beneficial for the features

[2] https://github.com/dmlc/dgl/.

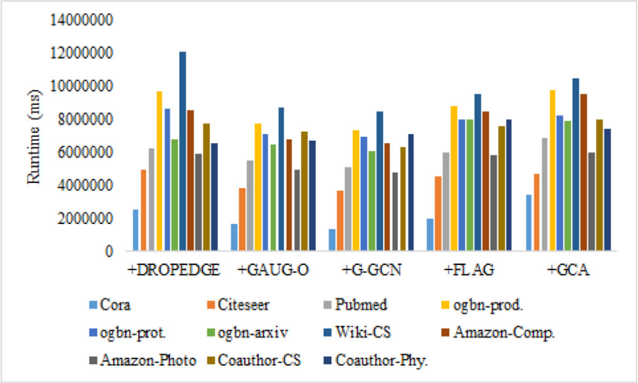

Fig. 4. Runtime of different GDA algorithms with GCN as plug-and-play on all datasets.

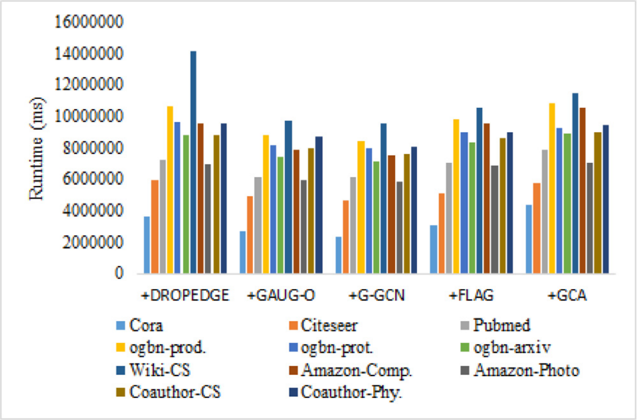

Fig. 5. Runtime of different GDA algorithms with GAT as plug-and-play on all datasets.

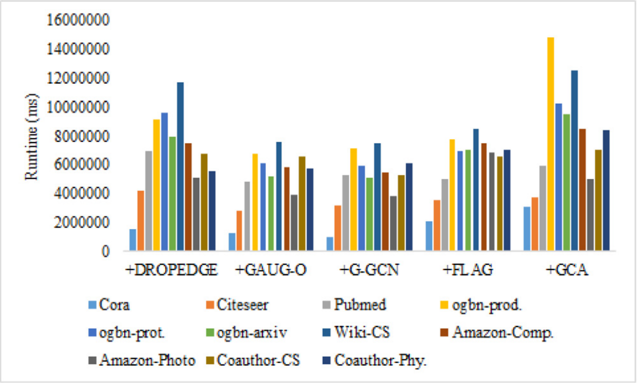

Fig. 6. Runtime of different GDA algorithms with GSAGE as plug-and-play on all datasets.

of the augmented nodes with adversarial perturbations such as +FLAG. Ogbn-proteins, ogbn-arxiv, and Coauthor-Physics benefit from +GCA while +GAUG-O leads the scoreboard on Amazon-Computers. Finally, the DROPEDGE technique, which randomly removes a substantial proportion of edges of the input graph, benefits the Pubmed dataset.

Given the GSAGE architecture, +GCA performs remarkably on Pubmed and ogbn-arxiv datasets, while +G-GNN lifted the accuracies on Amazon-Computers, Amazon-Photo, Coauthor-CS, and Coauthor-Physics datasets. Ultimately, global information extraction to serve as structure and attribute features for each node benefits the Amazon and Coauthor dataset when G-GNN is engineered with GSAGE. Therefore, more scalable to inductive settings as GSAGE is node-wise sampling-based, which constructs subgraphs for the model inference. +DROPEDGE leads the GSAGE group with remarkable performance on ogbn-products, and finally +FLAG leads the Citeseer and Wiki-CS datasets. Indeed, some algorithms do not achieve better results on some datasets. For instance, +DROPEDGE, +GAUG-O, +G-GNN pried with GCN architecture with +GAUG-O running out of memory (OOM) on Pubmed dataset as shown in the Tables.

As stated, the GAUG framework engages a dual-step, which is to obtain edge probabilities for all potential and existing edges via edge predictor function followed by adding/removing new or existing edges utilizing the predicted edge probabilities to construct a modified graph to serve as an input to a GNN node classifier. Hence, presumably, the reason for OOM on some datasets. Also, while the DropEdge technique randomly removes a substantial proportion of edges of the input graph, AdaEdge adjusts the graph topology adaptively by iteratively adding the intra-class edges and removing the inter-class edges during training. Although LA-GNN, NodeAug, and AdaEdge were proposed recently, the source code was not available online when this survey was conducted.

Figs. 4, 5 and 6 depict the running time of the algorithms discussed on all data sets. Although it took some time since the result is based on the average value of 10 experiments with 1000 epochs, it is clear that feature-level augmentation algorithms are far ahead in terms of operational efficiency. However, it is easy to understand the reason for this difference. Topology-level augmentation algorithms need to consider effective probability actions sensitive to the different levels to add/remove linkages built by edges. Intuitively, GCA uses adaptive measures to perturb the edges on the topology level by assigning large drop probabilities to trivial edges to emphasize influential linked structures. Additionally, the attribute masking level enforces the model to corrupt node attributes by employing additional noise to trivial feature dimensions to identify the underlying context information. Hence, presumably, the upsurge in computational complexity. Nevertheless, it is worth mentioning that the running time of GAUG-O is comparable to G-GNN and FLAG on GSAGE architecture across various datasets.

Also, GAT and GCN are ineffective on large-scale graphs. For a node in the graph, computing its embedding requires aggregate embeddings of its neighbors by the aggregator established by the model. Therefore, the number of nodes needed to embed this node increases exponentially with the number of layers. In this case, an attention model like GAT is introduced to another complexity due to the computation of pairwise attention weights between connected nodes, which involves the softmax operator. These two operations increase the computation when the degree (the number of neighbors of each node) is high. This problem is even more alarming after data augmentation and when performing backpropagation on the GNN during training. Therefore, training the GDA algorithms with GAT and GCN architectures takes more time compared to GSAGE.

5. Current challenges and future direction

This section comprehensively investigates the challenges and future direction for GDA by identifying several open conceivable issues that may require further study based on the interpretation of the vast literature and empirical results.

I. *Theoretical interpretability*: Regardless of GDA improving the generalization of graph learning, and combatting over-smoothing, there is little information to apprehending its achievement. Therefore, there is a lack of sufficient proof or theoretical foundation for GDA compared to several works [98–100] that provided theoretical insights into data augmentation in CV. For example, DropEdge draws additional virtual examples from neighborhood information for each node by randomly removing edges to increase the support of the training distribution. It is based on the assumption that the node's class labels are unchanged after dropping the edge. However, such a hypothesis is dependent on the dataset and thus requires proficient knowledge. Again, although DropEdge aggregates neighborhood information for each node to share the same class, it does not describe the neighborhood relation across samples of different classes. Likewise, there are no theoretical guarantees for the sampling quality in subgraph mini-batch training in NodeAug.

II. *Scalability*: GDA and datasets such as OGB have addressed the significant issue related to unrealistic and arbitrary data splits and the common neglect of the validation set of traditional graph datasets. However, they have revealed more practical challenges in the GNN research society. As the graph dataset increases exponentially, the question remains as to how to properly extend the scalability. Most importantly, how to design GDA techniques to produce more significant performance improvements in accuracy and time complexity. Observably, some algorithm takes too much time to train, with some out-of-memory on the OGB datasets as it is large-scale. For example, GAUG-O required backpropagating on the entire adjacency matrix for end-to-end training, therefore presenting a space complexity $O(N^2)$. DropEdge required pre-calculation of a score for each edge in the graph before training, demanding extra memory on GPU. Hence, the introduction of a subgraph mini-batch training for creative execution in the NodeAug model is a keen interest to focus. Notably, Graph mini-batch algorithms are critical to training GNNs on large-scale datasets. In GAUG-O, the authors engaged in graph mini-batch training [29] to achieve a satisfactory space complexity of $O(M^2)$, where M is the batch size. Hence, it is necessary to explore the compatibility of different GDA algorithms with a mini-batch model.

III. *Information-to-noise ratio*: The standard GCN for node classification reveals that they are usually shallow. Therefore, an attempt to go deeper is entrenched with over-smoothing. From the point of view of Chen et al. [24], an increase in the network layer, where there is a small information-to-noise ratio, causes an interaction between high-order neighbors to bring too much noise and vice versa. Hence, it weakens the valuable information, which is why the issue of over-smoothing. Balancing such phenomena is relatively underexplored due to the challenges brought by the complex and non-Euclidean structure of graph data. A GDA framework to de-noise the small information-to-noise ratio in high-order neighbors and increase the noise in a high information-to-noise ratio in low-order neighbors still remains an open question.

IV. *Efficiency*: Given a small proportion of node labels, a semi-supervised approach to graph data aims at recovering labels for all nodes. This practice is widespread in various real-world applications, as numerous labels are always costly and difficult to assemble. Moreover, various GNNs have significantly influenced semi-supervised learning on attributed graphs. GNN usually comprises multiple layers. Training such layers requires recursively updating each node in the graph, which becomes infeasible and ineffective. The nodes iteratively accumulate information from the neighborhood layer to the next layer. G-GNN constructs the global information as a global structure with global attribute features to each node and pre-training them for a parallel GNN. However, learning global information often requires learning from the entire graph. Consequently, inefficient in each training iteration. LA-GNN, on the other hand, engages the central node's feature to learn the conditional distribution of its neighbors' features for a new optimal feature, which is said to have improved the training efficiency. However, the approach lacks enough experimental evidence based on applicability to real-world datasets.

V. *Domain Adaptation*: The involvement of various pairs of augmentation of a graph relative to the graph and its augmentation has been established to further improve the performance. GCA composed an edge perturbation and applied attribute masking to achieve adequate performance in denser graphs. The authors hypothesize that masking patterns matter, and masking more central nodes with high degrees benefits denser graphs because GNNs cannot reconstruct the missing information of isolated nodes, according to the message passing mechanism [42]. It is also established that edge perturbation benefits social networks; however, it damages some biochemical molecules. Therefore, a GDA pair that can adapt to datasets of different domains and sizes is feasible for further research. More importantly, automated data augmentation [49], since most algorithm manually picks data augmentations per dataset by tedious trial and error, significantly limits generality and practicality.

6. Conclusion

In this work, we comprehensively evaluated eight GDA algorithms and tested them on additional educational datasets (eleven in total) different from the datasets used in the original works. We offer to group the state-of-the-art GDA algorithms into two broad categories: topology-level and feature-level augmentation to comprehend the similarity and difference between various works. Along with the grouping described above, we interpret the algorithms proposed for each category to improve understanding for each category. While paying attention to the accuracy performance, we consider the running time of these algorithms, which has an important guiding influence for unraveling practical and real-world problems. In summary, our work is beneficial for researchers in the GNN community to grasp the advantages and weaknesses of these described algorithms. Furthermore, the unification of the mathematical formulation surrounding these algorithms would guide researchers on what to look for in a GNN augmentation work with relations to feature and topology perturbation.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was supported by the Zhejiang Normal University Postdoctoral Research Fund (Grant No. ZC304021943), the Natural Science Foundation of China (Project No. 61976196) and the Zhejiang Provincial Natural Science Foundation of China under Grant No. LZ22F030003.

References

[1] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton, ImageNet classification with deep convolutional neural networks, in: F. Pereira, C.J.C. Burges, L. Bottou, K.Q. Weinberger (Eds.), Advances in Neural Information Processing Systems, Vol. 25, Curran Associates, Inc., 2012, pp. 84–90.

[2] Xifeng Guo, Xinwang Liu, En Zhu, Xinzhong Zhu, Miaomiao Li, Xin Xu, Jianping Yin, Adaptive self-paced deep clustering with data augmentation, IEEE Trans. Knowl. Data Eng. 32 (9) (2020) 1680–1693.

[3] Connor Shorten, Taghi M. Khoshgoftaar, A survey on image data augmentation for deep learning, J. Big Data 6 (1) (2019) 1–48.

[4] Michael Adjeisah, Guohua Liu, Douglas Omwenga Nyabuga, Richard Nuetey Nortey, Jinling Song, Pseudotext injection and advance filtering of low-resource corpus for neural machine translation, Comput. Intell. Neurosci. 2021 (2021).

[5] Jason Wei, Kai Zou, EDA: Easy data augmentation techniques for boosting performance on text classification tasks, in: Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), Association for Computational Linguistics, Hong Kong, China, 2019, pp. 6382–6388.

[6] K. Kong, G. Li, M. Ding, Z. Wu, C. Zhu, B. Ghanem, G. Taylor, T. Goldstein, FLAG: adversarial data augmentation for graph neural networks 2020, 2021, arXiv preprint arXiv:2010.09891.

[7] Tong Zhao, Yozen Liu, Leonardo Neves, Oliver Woodford, Meng Jiang, Neil Shah, Data augmentation for graph neural networks, in: The Thirty-Fifth AAAI Conference on Artificial Intelligence (AAAI-21), 2021, pp. 11015–11023.

[8] Thomas N. Kipf, Max Welling, Semi-supervised classification with graph convolutional networks, in: 5th International Conference on Learning Representations (ICLR-2017), 2017.

[9] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, Yoshua Bengio, Graph attention networks, in: 6th International Conference on Learning Representations (ICLR-2018), 2018.

[10] Keyulu Xu, Weihua Hu, Jure Leskovec, Stefanie Jegelka, How powerful are graph neural networks? in: 7th International Conference on Learning Representations, ICLR-2019, 2019.

[11] Yuanfu Lu, Xunqiang Jiang, Yuan Fang, Chuan Shi, Learning to pre-train graph neural networks, in: Proceedings of the AAAI Conference on Artificial Intelligence (AAAI-21), AAAI, 2021, pp. 4276—4284.

[12] Yawei Zhao, Kai Xu, En Zhu, Xinwang Liu, Xinzhong Zhu, Jianping Yin, Triangle lasso for simultaneous clustering and optimization in graph datasets, IEEE Trans. Knowl. Data Eng. 31 (8) (2019) 1610–1623.

[13] Yanqiao Zhu, Yichen Xu, Feng Yu, Qiang Liu, Shu Wu, Liang Wang, Graph contrastive learning with adaptive augmentation, in: Proceedings of the Web Conference 2021, 2021, pp. 2069–2080.

[14] Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, Yang Shen, Graph contrastive learning with augmentations, Adv. Neural Inf. Process. Syst. 33 (2020) 5812–5823.

[15] Muhan Zhang, Yixin Chen, Link prediction based on graph neural networks, Adv. Neural Inf. Process. Syst. 31 (2018) 5165—5175.

[16] Difan Zou, Ziniu Hu, Yewen Wang, Song Jiang, Yizhou Sun, Quanquan Gu, Layer-dependent importance sampling for training deep and large graph convolutional networks, Adv. Neural Inf. Process. Syst. 32 (2019) 11249—11259.

[17] Meng Liu, Hongyang Gao, Shuiwang Ji, Towards deeper graph neural networks, in: Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2020, pp. 338–348.

[18] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, Jure Leskovec, Hierarchical graph representation learning with differentiable pooling, Adv. Neural Inf. Process. Syst. 31 (2018) 4800–4810.

[19] Qimai Li, Zhichao Han, Xiao-Ming Wu, Deeper insights into graph convolutional networks for semi-supervised learning, in: Thirty-Second AAAI Conference on Artificial Intelligence, 2018, pp. 3538–3545.

[20] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, Stefanie Jegelka, Representation learning on graphs with jumping knowledge networks, in: International Conference on Machine Learning, PMLR, 2018, pp. 5453–5462.

[21] Johannes Klicpera, Aleksandar Bojchevski, Stephan Günnemann, Predict then propagate: Graph neural networks meet personalized pagerank, in: 7th International Conference on Learning Representations, ICLR-19, 2019.

[22] Vijay Prakash Dwivedi, Chaitanya K. Joshi, Thomas Laurent, Yoshua Bengio, Xavier Bresson, Benchmarking graph neural networks, 2020, arXiv preprint arXiv:2003.00982.

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

[24] Federico Errica, Marco Podda, Davide Bacciu, Alessio Micheli, A fair comparison of graph neural networks for graph classification, in: Proceedings of the International Conference on Learning Representations (ICLR-20), 2019.

[25] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, Jure Leskovec, Open graph benchmark: Datasets for machine learning on graphs, Adv. Neural Inf. Process. Syst. 33 (2020) 22118–22133.

[26] Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, Xu Sun, Measuring and relieving the over-smoothing problem for graph neural networks from the topological view, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 34, 2020, pp. 3438–3445.

[27] Yu Rong, Wenbing Huang, Tingyang Xu, Junzhou Huang, Dropedge: Towards deep graph convolutional networks on node classification, 2019, arXiv preprint arXiv:1907.10903.

[28] Jianfei Chen, Jun Zhu, Le Song, Stochastic training of graph convolutional networks with variance reduction, in: 35th International Conference on Machine Learning, (ICML-18), 2018, pp. 1503—1532.

[29] Will Hamilton, Zhitao Ying, Jure Leskovec, Inductive representation learning on large graphs, Adv. Neural Inf. Process. Syst. 30 (2017) 1024—1034.

[30] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, Jure Leskovec, Graph convolutional neural networks for web-scale recommender systems, in: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2018, pp. 974–983.

[31] Wenbing Huang, Tong Zhang, Yu Rong, Junzhou Huang, Adaptive sampling towards fast graph representation learning, Adv. Neural Inf. Process. Syst. 31 (2018) 4558–4567.

[32] Jie Chen, Tengfei Ma, Cao Xiao, Fastgcn: fast learning with graph convolutional networks via importance sampling, in: 6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings, 2018.

[33] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, Cho-Jui Hsieh, Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks, in: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2019, pp. 257–266.

[34] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, Viktor Prasanna, Graphsaint: Graph sampling based inductive learning method, in: Proceedings of International Conference on Learning Representations (ICLR-20), 2020.

[35] Xiaoyun Wang, Minhao Cheng, Joe Eaton, Cho-Jui Hsieh, Felix Wu, Attack graph convolutional networks by adding fake nodes, 2018, arXiv preprint arXiv:1810.10751.

[36] Hanjun Dai, Hui Li, Tian Tian, Xin Huang, Lin Wang, Jun Zhu, Le Song, Adversarial attack on graph structured data, in: International Conference on Machine Learning, PMLR, 2018, pp. 1115–1124.

[37] Ali Shafahi, Mahyar Najibi, Mohammad Amin Ghiasi, Zheng Xu, John Dickerson, Christoph Studer, Larry S Davis, Gavin Taylor, Tom Goldstein, Adversarial training for free!, Adv. Neural Inf. Process. Syst. 32 (2019) 3358—3369.

[38] Zhijie Deng, Yinpeng Dong, Jun Zhu, Batch virtual adversarial training for graph convolutional networks, in: ICML 2019 Workshop on Learning and Reasoning with Graph-Structured Data, 2019.

[39] Susheel Suresh, Pan Li, Cong Hao, Jennifer Neville, Adversarial graph augmentation to improve graph contrastive learning, Adv. Neural Inf. Process. Syst. 34 (2021).

[40] Tong Zhao, Gang Liu, Stephan Günnemann, Meng Jiang, Graph data augmentation for graph machine learning: A survey, 2022, arXiv preprint arXiv:2202.08871.

[41] Dan-Hao Zhu, Xin-Yu Dai, Jia-Jun Chen, Pre-train and learn: Preserving global information for graph neural networks, J. Comput. Sci. Tech. 36 (6) (2021) 1420–1430.

[42] Songtao Liu, Hanze Dong, Lanqing Li, Tingyang Xu, Yu Rong, Peilin Zhao, Junzhou Huang, Dinghao Wu, Local augmentation for graph neural networks, 2021, arXiv preprint arXiv:2109.03856.

[43] Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, Jure Leskovec, Strategies for pre-training graph neural networks, in: International Conference on Learning Representations (ICLR-20), 2020.

[44] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, George E. Dahl, Neural message passing for quantum chemistry, in: International Conference on Machine Learning, PMLR, 2017, pp. 1263–1272.

[45] Paridhi Maheshwari, Self-supervised learning for graphs | stanford CS224W GraphML tutorials, 2022, [Online]. Available: https://medium.com/stanford-cs224w/self-supervised-learning-for-graphs-963e03b9f809. [Accessed: 11-Feb-2022].

[46] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, Kilian Weinberger, Simplifying graph convolutional networks, in: International Conference on Machine Learning, PMLR, 2019, pp. 6861–6871.

[47] Shunxin Xiao, Shiping Wang, Yuanfei Dai, Wenzhong Guo, Graph neural networks in node classification: survey and evaluation, Mach. Vis. Appl. 33 (1) (2022) 1–19.

[48] Zhilin Yang, William Cohen, Ruslan Salakhudinov, Revisiting semi-supervised learning with graph embeddings, in: International Conference on Machine Learning, PMLR, 2016, pp. 40–48.

[49] Yuning You, Tianlong Chen, Yang Shen, Zhangyang Wang, Graph contrastive learning automated, in: International Conference on Machine Learning, PMLR, 2021, pp. 12121–12132.

[50] Jun Xia, Lirong Wu, Jintao Chen, Bozhen Hu, Stan Z. Li, SimGRACE: A simple framework for graph contrastive learning without data augmentation, in: Proceedings of the ACM Web Conference 2022, 2022, pp. 1070–1079.

[51] Jiezhong Qiu, Qibin Chen, Yuxiao Dong, Jing Zhang, Hongxia Yang, Ming Ding, Kuansan Wang, Jie Tang, Gcc: Graph contrastive coding for graph neural network pre-training, in: Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2020, pp. 1150–1160.

[52] Daniel Zügner, Amir Akbarnejad, Stephan Günnemann, Adversarial attacks on neural networks for graph data, in: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2018, pp. 2847–2856.

[53] Zhan Gao, Subhrajit Bhattacharya, Leiming Zhang, Rick S. Blum, Alejandro Ribeiro, Brian M. Sadler, Training robust graph neural networks with topology adaptive edge dropping, 2021, arXiv preprint arXiv:2106.02892.

[54] Cheng Zheng, Bo Zong, Wei Cheng, Dongjin Song, Jingchao Ni, Wenchao Yu, Haifeng Chen, Wei Wang, Robust graph representation learning via neural sparsification, in: International Conference on Machine Learning, PMLR, 2020, pp. 11458–11468.

[55] Dongsheng Luo, Wei Cheng, Wenchao Yu, Bo Zong, Jingchao Ni, Haifeng Chen, Xiang Zhang, Learning to drop: Robust graph neural network via topological denoising, in: Proceedings of the 14th ACM International Conference on Web Search and Data Mining, 2021, pp. 779–787.

[56] Wenzheng Feng, Jie Zhang, Yuxiao Dong, Yu Han, Huanbo Luan, Qian Xu, Qiang Yang, Evgeny Kharlamov, Jie Tang, Graph random neural networks for semi-supervised learning on graphs, Adv. Neural Inf. Process. Syst. 33 (2020) 22092–22103.

[57] Yiwei Wang, Wei Wang, Yuxuan Liang, Yujun Cai, Bryan Hooi, Mixup for node and graph classification, in: Proceedings of the Web Conference 2021, 2021, pp. 3663–3674.

[58] Kaveh Hassani, Amir Hosein Khasahmadi, Contrastive multi-view representation learning on graphs, in: International Conference on Machine Learning, PMLR, 2020, pp. 4116–4126.

[59] Chang Tang, Xinwang Liu, Xinzhong Zhu, En Zhu, Zhigang Luo, Lizhe Wang, Wen Gao, CGD: Multi-view clustering via cross-view graph diffusion, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 34, 2020, pp. 5924–5931.

[60] Johannes Klicpera, Stefan Weißenberger, Stephan Günnemann, Diffusion improves graph learning, 2019, arXiv preprint arXiv:1911.05485.

[61] Jinliang Yuan, Hualei Yu, Meng Cao, Ming Xu, Junyuan Xie, Chongjun Wang, Semi-supervised and self-supervised classification with multi-view graph neural networks, in: Proceedings of the 30th ACM International Conference on Information & Knowledge Management, 2021, pp. 2466–2476.

[62] Aleksandar Bojchevski, Oleksandr Shchur, Daniel Zügner, Stephan Günnemann, Netgan: Generating graphs via random walks, in: International Conference on Machine Learning, PMLR, 2018, pp. 610–619.

[63] Petar Veličković, William Fedus, William L. Hamilton, Pietro Liò, Yoshua Bengio, R. Devon Hjelm, Deep graph infomax, in: 7th International Conference on Learning Representations, ICLR-19, 2019.

[64] Fan-Yun Sun, Jordan Hoffmann, Vikas Verma, Jian Tang, Infograph: Unsupervised and semi-supervised graph-level representation learning via mutual information maximization, in: International Conference on Learning Representations (ICLR-20), 2020.

[65] Yiwei Wang, Wei Wang, Yuxuan Liang, Yujun Cai, Bryan Hooi, Graphcrop: Subgraph cropping for graph classification, 2020, arXiv preprint arXiv:2009.10564.

[66] Hongyu Guo, Yongyi Mao, ifMixup: Towards intrusion-free graph mixup for graph classification, 2021, arXiv, 2110.

[67] Vikas Verma, Meng Qu, Kenji Kawaguchi, Alex Lamb, Yoshua Bengio, Juho Kannala, Jian Tang, Graphmix: Improved training of gnns for semi-supervised learning, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 35, 2021, pp. 10024–10032.

[68] Joonhyung Park, Hajin Shim, Eunho Yang, Graph transplant: Node saliency-guided graph mixup with local structure preservation, in: Proceedings of the First MiniCon Conference, 2022.

[69] Hengyuan Ma, Qi Yang, Bowen Sun, Long Shun, Junkui Li, Jianfeng Feng, Sampling before training: Rethinking the effect of edges in the process of training graph neural networks, 2021.

[70] Matthew P. Dube, Topological augmentation: A step forward for qualitative partition reasoning, J. Spat. Inf. Sci. (14) (2017) 1–29.

[71] Yiwei Wang, Yujun Cai, Yuxuan Liang, Henghui Ding, Changhu Wang, Siddharth Bhatia, Bryan Hooi, Adaptive data augmentation on temporal graphs, Adv. Neural Inf. Process. Syst. 34 (2021).

[72] Yingxue Zhang, Soumyasundar Pal, Mark Coates, Deniz Ustebay, Bayesian graph convolutional neural networks for semi-supervised classification, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 33, 2019, pp. 5829–5836.

[73] Olivier Chapelle, Jason Weston, Léon Bottou, Vladimir Vapnik, Vicinal risk minimization, Adv. Neural Inf. Process. Syst. 13 (2000) 416–422.

[74] Kenta Oono, Taiji Suzuki, Graph neural networks exponentially lose expressive power for node classification, in: International Conference on Learning Representations (ICLR-20), 2020.

[75] Kenta Oono, Taiji Suzuki, On asymptotic behaviors of graph cnns from dynamical systems perspective, 2019, arXiv preprint arXiv:1905.10947.

[76] Thomas N. Kipf, Max Welling, Variational graph auto-encoders, in: NIPS Workshop on Bayesian Deep Learning, 2016.

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

[78] Yanqiao Zhu, Yichen Xu, Feng Yu, Qiang Liu, Shu Wu, Liang Wang, Deep graph contrastive representation learning, in: ICML Workshop on Graph Representation Learning and beyond, 2020.

[79] Fuli Feng, Xiangnan He, Jie Tang, Tat-Seng Chua, Graph adversarial training: Dynamically regularizing based on graph structure, IEEE Trans. Knowl. Data Eng. 33 (6) (2019) 2493–2504.

[80] Yiwei Wang, Wei Wang, Yuxuan Liang, Yujun Cai, Juncheng Liu, Bryan Hooi, Nodeaug: Semi-supervised node classification with data augmentation, in: Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2020, pp. 207–217.

[81] Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Alexander Turner, Aleksander Madry, Robustness may be at odds with accuracy, in: 7th International Conference on Learning Representations (ICLR-19), 2019.

[82] Yogesh Balaji, Tom Goldstein, Judy Hoffman, Instance adaptive adversarial training: Improved accuracy tradeoffs in neural nets, in: International Conference on Learning Representations (ICLR-20), 2020.

[83] Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, Tuo Zhao, Smart: Robust and efficient fine-tuning for pre-trained natural language models through principled regularized optimization, in: Proc. 58th Annu. Meet. Assoc. Comput. Linguist., 2020, pp. 2177—2190.

[84] Takeru Miyato, Shin-ichi Maeda, Masanori Koyama, Shin Ishii, Virtual adversarial training: a regularization method for supervised and semi-supervised learning, IEEE Trans. Pattern Anal. Mach. Intell. 41 (8) (2018) 1979–1993.

[85] Danhao Zhu, Xin-yu Dai, Kaijia Yang, Jiajun Chen, Yong He, PCANE: Preserving context attributes for network embedding, in: Pacific-Asia Conference on Knowledge Discovery and Data Mining, Springer, 2019, pp. 156–168.

[86] Cihang Xie, Mingxing Tan, Boqing Gong, Jiang Wang, Alan L. Yuille, Quoc V. Le, Adversarial examples improve image recognition, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 819–828.

[87] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, Ping Tak Peter Tang, On large-batch training for deep learning: Generalization gap and sharp minima, in: 5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings, 2017.

[88] James M. Joyce, Kullback-Leibler divergence, in: International Encyclopedia of Statistical Science, Springer, 2011, pp. 720–722.

[89] Ian J. Goodfellow, Jonathon Shlens, Christian Szegedy, Explaining and harnessing adversarial examples, in: 3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings, 2015.

[90] Alexey Kurakin, Ian Goodfellow, Samy Bengio, Adversarial machine learning at scale, in: 5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings, 2017.

[91] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, Adrian Vladu, Towards deep learning models resistant to adversarial attacks, in: N 6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings, 2018.

[92] Ting Chen, Simon Kornblith, Mohammad Norouzi, Geoffrey Hinton, A simple framework for contrastive learning of visual representations, in: International Conference on Machine Learning, PMLR, 2020, pp. 1597–1607.

[93] Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, Yaliang Li, Simple and deep graph convolutional networks, in: International Conference on Machine Learning, PMLR, 2020, pp. 1725–1735.

[94] Kihyuk Sohn, Honglak Lee, Xinchen Yan, Learning structured output representation using deep conditional generative models, Adv. Neural Inf. Process. Syst. 28, 3483—3491.

[95] Bryan Perozzi, Rami Al-Rfou, Steven Skiena, Deepwalk: Online learning of social representations, in: Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2014, pp. 701–710.

[96] Péter Mernyei, Cătălina Cangea, Wiki-cs: A wikipedia-based benchmark for graph neural networks, in: ICML Workshop on Graph Representation Learning and beyond, 2020.

[97] Jeffrey Pennington, Richard Socher, Christopher D. Manning, Glove: Global vectors for word representation, in: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP, 2014 pp. 1532–1543.

[98] Sang Michael Xie, Aditi Raghunathan, Fanny Yang, John C. Duchi, Percy Liang, When covariate-shifted data augmentation increases test error and how to fix it, 2019.

[99] Peter L. Bartlett, Philip M. Long, Gábor Lugosi, Alexander Tsigler, Benign overfitting in linear regression, Proc. Natl. Acad. Sci. 117 (48) (2020) 30063–30070.

[100] Sen Wu, Hongyang Zhang, Gregory Valiant, Christopher Ré, On the generalization effects of linear transformations in data augmentation, in: International Conference on Machine Learning, PMLR, 2020 pp. 10410–10420.