LOCAL AUGMENTATION FOR GRAPH NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Data augmentation has been widely used in image data and linguistic data but remains under-explored on graph-structured data. Existing methods focus on augmenting the graph data from a global perspective and largely fall into two genres: structural manipulation and adversarial training with feature noise injection. However, the structural manipulation approach suffers information loss issues while the adversarial training approach may downgrade the feature quality by injecting noise. In this work, we introduce the local augmentation, which enhances node features by its local subgraph structures. Specifically, we model the data augmentation as a feature generation process. Given the central node's feature, our local augmentation approach learns the conditional distribution of its neighbors' features and generates the neighbors' optimal feature to boost the performance of downstream tasks. Based on the local augmentation, we further design a novel framework: LA-GNN, which can apply to any GNN models in a plug-and-play manner. Extensive experiments and analyses show that local augmentation consistently yields performance improvement for various GNN architectures across a diverse set of benchmarks.

1 INTRODUCTION

Graph Neural Networks (GNNs) and their variants (Abu-El-Haija et al., 2019; Kipf & Welling, 2017; Veličković et al., 2018) have achieved state-of-the-art performance on various tasks such as recommendation system (Ying et al., 2018) and traffic prediction (Guo et al., 2019). However, as Kipf & Welling (2017) points out, deep GCN are prone to over-fitting due to limited labeled data (Shorten & Khoshgoftaar, 2019) and additional model parameters as the number of parameters increase with model depth. Meanwhile, as a data-based solution of alleviating over-fitting, *data augmentation* has been successfully adopted in many areas, such as computer vision (Cubuk et al., 2019) and natural language processing (Fadaee et al., 2017). Therefore, graph data augmentation has been studied recently. For explicit graph data augmentation, the existing methods can be divided into two categories: either topology-level (Rong et al., 2020; Wang et al., 2020b; Zhao et al., 2021) or feature-level augmentation (Deng et al., 2019; Feng et al., 2019; Kong et al., 2020).

A standard operation on topology-level augmentation is to perturb the adjacency matrix, yielding different graph structures. For instance, DropEdge (Rong et al., 2020) randomly removes a certain percentage of edges of the input graph for each training epoch and can be interpreted as a topology-level augmenter. NodeAug (Wang et al., 2020b) adds and removes edges on subgraphs based on their self-defined importance score. Zhao et al. (2021) exploit their proposed neural edge predictor to promote intra-class edges and demote inter-class edges in a given graph structure. While feature-level augmentation recovers missing or interrupted vertex features using their context information. Existing feature-level augmentation mainly exploits perturbations guided by adversarial training. Some studies (Deng et al., 2019; Feng et al., 2019; Kong et al., 2020) inject adversarial noise to node attributes and regularize the training of GNNs. Nevertheless, these feature-level augmentation techniques have three drawbacks. 1) They employ full-batch adversarial training for augmentation, which is unnecessary, and introduce some additional side effects such as over-smoothing (Li et al., 2018). 2) This type of feature-level augmentation is coarse-grained, which focuses on global augmentation and abandons the local information of the neighborhood. 3) Injecting the perturbation into features degrades the quality of the feature representations. Moreover, to our best knowledge,

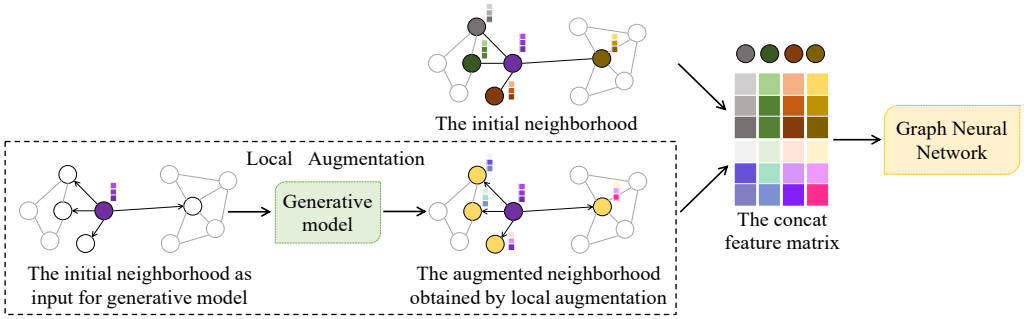


Figure 1: A schematic depiction of our local augmentation. The purple and yellow circles on the graph correspond to the central node and its augmented neighbors respectively. After augmenting the neighborhood, we concat the initial and the generated feature matrix as input for GNNs.

none of the existing approaches combines both the feature representations and the graph topology, especially the local subgraph structures, for graph-level data augmentation.

In this work, to consider both structure and features, we propose a framework: **L**ocal **A**ugmentation for **G**raph **N**eural **N**etworks (LA-GNNs). The term "local augmentation" refers to the generation of neighborhood information by **conditioning on local structures and node features** via a generative model. Specifically, our proposed framework learns the conditional distribution of the connected neighbors' representations given the representation of the central node, which has some similarities with the Skip-gram (Mikolov et al., 2013) and Deepwalk Perozzi et al. (2014). But our method does not focus on word or graph embedding. The motivation behind this work concludes three-fold. 1) Existing feature-level augmentation works primarily pay attention to global augmentation without considering the informative neighborhood. 2) The distribution of the representations of the neighbors has a close connection to the central node, making ample room for feature augmentation. 3) JKnet (Xu et al., 2018), APPNP (Klicpera et al., 2019) and GCNII (Chen et al., 2020) preserve the locality of the node representations to avoid the over-smoothing, yielding the demand of local augmentation. And there are several benefits in applying local augmentation for the GNN training. First, local augmentation is essentially a data augmentation technique that can improve the generalization of the GNN models and prevent over-fitting. Second, we can recover some missing or interrupted feature information in an attributed graph via the generative model (Jia & Benson, 2020). Third, our proposed framework is flexible and can be equipped with various popular backbone networks such as GCN (Kipf & Welling, 2017), GAT (Veličković et al., 2018), GCNII (Chen et al., 2020), and GRAND (Feng et al., 2020) to enhance their performance. Extensive experimental results demonstrate that our proposed framework could improve the performance of GNN variants on benchmark datasets.

2 Local Augmentation

The local augmentation framework we proposed consists of three modules: the generator, the importance sampling module, and GNN as an auxiliary downstream task, as illustrated in Figure 1. For each node, the pipeline utilizes the generative model to learn the conditional distribution of the feature representations of its connected neighbors. We then exploit importance sampling to determine the optimal generated data to augment the training set, according to the dropout uncertainty of the downstream GNN predictor, which is subsequently trained on the augmented dataset.

2.1 Notations and Preliminaries

Notations. Let $G = (V, E)$ represent the graph, where V is the set of vertices $\{v_1, \cdots, v_N\}$ with $|V| = N$ and E is the set of edges. The adjacency matrix is defined as $\mathbf{A} \in \{0,1\}^{N\times N}$, and $\mathbf{A}_{ij} = 1$ if and only if $(v_i, v_j) \in E$. Let $\mathcal{N}_i = \{v_j | \mathbf{A}_{ij} = 1\}$ denotes the neighborhood of node v_i and $\mathbf{D}$ denote the diagonal degree matrix, where $\mathbf{D}_{ii} = \sum_{j=1}^{n} \mathbf{A}_{ij}$. The feature matrix is denoted as $\mathbf{X} \in \mathbb{R}^{N\times F}$ where each node v is associated with a F-dimensional feature vector $\mathbf{X}_v$. $\mathbf{Y} \in \{0,1\}^{N\times C}$ denote the one-hot label matrix, where $\mathbf{Y}_i \in \{0,1\}^C$ is a one-hot vector and $\sum_{j=1}^{C} \mathbf{Y}_{ij} = 1$ for any $v_i \in V$.

GNN. Graph Neural Networks (GNNs) are neural network models, such as GCN and GAT (Kipf & Welling, 2017; Veličković et al., 2018), that capture the dependence of graphs via message passing between the nodes of a graph as

$$\mathbf{H}^{(\ell)} = f(\mathbf{A}, \mathbf{H}^{(\ell-1)}), \tag{1}$$

where f denotes the specific GNN layer for different models, $\mathbf{H}^{(\ell)}$ are the hidden vectors of the ℓ-th layer and $\mathbf{H}^{(0)} = \mathbf{X}$. For example, $f(\mathbf{A}, \mathbf{H}) = \sigma(\hat{\mathbf{A}}\mathbf{H}\mathbf{W})$ for GCN, where $\hat{\mathbf{A}} = \tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}$, $\tilde{\mathbf{D}}$ is the degree matrix of $\tilde{\mathbf{A}}$, i.e., $\tilde{\mathbf{D}}_{ii} = \sum_j \tilde{\mathbf{A}}_{ij}$, and $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$.

2.2 Learning The Conditional Distribution

We start by reviewing the semi-supervised learning of GNNs. Most existing GNN models (Kipf & Welling, 2017; Veličković et al., 2018) are viewed as a classification function to predict the class labels of the graph nodes. In this work, we use a GNN classification estimator $P_\theta(\mathbf{Y}|\mathbf{A}, \mathbf{X})$ (θ is the parameter) to model the conditional distribution of label $\mathbf{Y}$ with respect to the graph structure $\mathbf{A}$ and feature matrix $\mathbf{X}$. Given training samples $\{\mathbf{A}, \mathbf{X}, \mathbf{Y}\}$, the parameter θ can be estimated using Maximum Likelihood Estimation (MLE), by optimizing the following likelihood function:

$$\max \prod_{k \in \mathbf{K}} P_\theta \left(\mathbf{Y}_k | \mathbf{A}, \mathbf{X}\right), \tag{2}$$

where $\mathbf{K}$ is the set of node indices of the training dataset whose labels are visible during the semi-supervised training. To further boost the performance of GNN, a new model $P_\theta(\mathbf{Y}, \overline{\mathbf{X}}|\mathbf{A}, \mathbf{X})$ is introduced, where $\overline{\mathbf{X}}$ is generated features by feature-level augmentation (Kong et al., 2020; Zhu et al., 2020). For this model, the MLE method needs to optimize a marginalized probability P_θ over the generated feature matrix $\overline{\mathbf{X}}$:

$$\max \prod_{k \in \mathbf{K}} \int_{\overline{\mathbf{X}}} P_\theta \left(\mathbf{Y}_k, \overline{\mathbf{X}} | \mathbf{A}, \mathbf{X}\right). \tag{3}$$

For Bayesian tractability, we decompose P_θ in Eq.(3) as a product of two posterior probabilities:

$$P_{\theta,\phi}(\mathbf{Y}_k, \overline{\mathbf{X}}|\mathbf{A}, \mathbf{X}) := P_\theta(\mathbf{Y}_k|\mathbf{A}, \overline{\mathbf{X}}) Q_\phi(\overline{\mathbf{X}}|\mathbf{A}, \mathbf{X}), \tag{4}$$

where $P_\theta(\mathbf{Y}_k|\mathbf{A}, \overline{\mathbf{X}})$ and $Q_\phi(\overline{\mathbf{X}}|\mathbf{A}, \mathbf{X})$ denote the probabilistic distributions approximated by the downstream GNN and the (feature-level augmentation) generator respectively, parameterized by θ and ϕ. **There are two benefits in the decomposition above.** First, it allows us to decouple the training of the downstream predictor P_θ and the generator Q_ϕ, enabling the generator to easily generalize to other downstream tasks. Moreover, inspired by the successes of data augmentation via deep-learning-based generative modeling (Antoniou et al., 2017), the representation power of Eq.(4) is superior than that of a single predictor $P_\theta \left(\mathbf{Y}_k|\mathbf{A}, \mathbf{X}\right)$ without data augmentation.

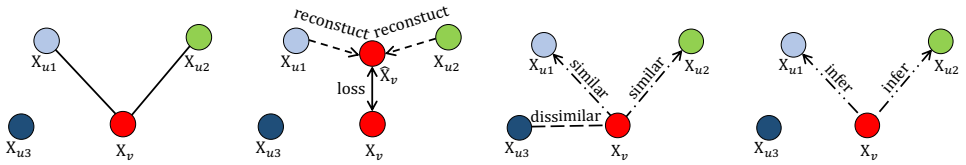
Consequently, once a generator Q_ϕ is trained very well, our training procedure can optimize $P_\theta(\mathbf{Y}_k|\mathbf{A}, \overline{\mathbf{X}})$ with samples $\overline{\mathbf{X}}$ drawn from the fixed conditional distribution Q_ϕ. Now, we show how to train the generator as follows.

Generator To learn a feature augmentation generator, a naive solution is to learn one single distribution for all the neighbors using the MLE method, i.e., solving the following optimization problem

$$\max_\psi \sum_{j \in \mathcal{N}_i} \log p_\psi \left(\mathbf{X}_j | \mathbf{X}_i\right) = \max_\psi \log \prod_{j \in \mathcal{N}_i} p_\psi \left(\mathbf{X}_j | \mathbf{X}_i\right), \tag{5}$$

where $\{\mathbf{X}_{j|j \in \mathcal{N}_i}, \mathbf{X}_i\}$. Then p_ψ can be used to augment features for all the neighbors. However, this method ignores the differences between all the neighbors, which may induce severe noise.

To overcome the limitation, we assume that each neighbor satisfies a different conditional distribution. Specifically, there exists a conditional distribution $p(\cdot|\mathbf{X}_i, \mathbf{z}_j)$ with latent random variable $\mathbf{z}_j$, such that we have $\mathbf{X}_j \sim p(\mathbf{X}|\mathbf{X}_i, \mathbf{z}_j)$ for $\mathbf{X}_{j|j \in \mathcal{N}_i}$. Once we obtain $p(\cdot|\mathbf{X}_i, \mathbf{z}_j)$ in some way, we can generate augmented features $\overline{\mathbf{X}}$, and then we can train $P_\theta(\mathbf{Y}_k|\mathbf{A}, \overline{\mathbf{X}})$ instead of $P_\theta(\mathbf{Y}_k|\mathbf{A}, \mathbf{X})$ to improve the final performance of P_θ. Below, we will present how to find $p(\cdot|\mathbf{X}_i, \mathbf{z}_j)$, which will produce the generator Q_ϕ.



(a) The Original Graph (b) EP-B (c) GraphSAGE (d) Local Augmentation

Figure 2: (a) The original graph. (b) EP-B exploits the neighbors to reconstruct the central node's embedding. (c) GraphSAGE encourages nearby nodes to have similar embeddings. (d) Given the representation of the central node, our aim is to infer the representations of the connected distribution of neighbors.

To achieve our purpose, a suitable method is the conditional variational auto-encoder (CVAE) (Kingma & Welling, 2013; Sohn et al., 2015), which can help learn the distribution of the latent variable $\mathbf{z}_j$, and the conditional distribution $p(\cdot|\mathbf{X}_i, \mathbf{z}_j)$. So, a CVAE model $Q_\phi\left(\overline{\mathbf{X}}|\mathbf{A}, \mathbf{X}\right)$ is adopted as our generator, where $\phi = \{\varphi, \psi\}$, φ denotes the variational parameters and ψ represents the generative parameters. To derive the optimization problem for CVAE, $\log p_\psi\left(\mathbf{X}_j|\mathbf{X}_i\right)$ can be written with latent variables $\mathbf{z}$ as follows, following previous work (Pandey & Dukkipati, 2017; Sohn et al., 2015):

$$\begin{aligned}\log p_\psi\left(\mathbf{X}_j|\mathbf{X}_i\right) &= \int q_\varphi(\mathbf{z}|\mathbf{X}_j, \mathbf{X}_i) \log \frac{p_\psi(\mathbf{X}_j, \mathbf{z}|\mathbf{X}_i)}{q_\varphi(\mathbf{z}|\mathbf{X}_j, \mathbf{X}_i)} \mathrm{d}\mathbf{z} + KL(q_\varphi(\mathbf{z}|\mathbf{X}_j, \mathbf{X}_i)\|p_\psi(\mathbf{z}|\mathbf{X}_j, \mathbf{X}_i)) \\ &\geq \int q_\varphi(\mathbf{z}|\mathbf{X}_j, \mathbf{X}_i) \log \frac{p_\psi(\mathbf{X}_j, \mathbf{z}|\mathbf{X}_i)}{q_\varphi(\mathbf{z}|\mathbf{X}_j, \mathbf{X}_i)} \mathrm{d}\mathbf{z},\end{aligned}$$

and the evidence lower bound (ELBO) can be written as:

$$\mathcal{L}(\mathbf{X}_j, \mathbf{X}_i; \psi, \varphi) = -KL(q_\varphi(\mathbf{z}|\mathbf{X}_j, \mathbf{X}_i)\|p_\psi(\mathbf{z}|\mathbf{X}_i)) + \int q_\varphi(\mathbf{z}|\mathbf{X}_j, \mathbf{X}_i) \log p_\psi(\mathbf{X}_j|\mathbf{X}_i, \mathbf{z}) \mathrm{d}\mathbf{z}, \quad (6)$$

where the encoder $q_\varphi(\mathbf{z}|\mathbf{X}_j, \mathbf{X}_i) = \mathcal{N}(f(\mathbf{X}_j, \mathbf{X}_i), g(\mathbf{X}_j, \mathbf{X}_i))$ and decoder $p_\psi(\mathbf{X}_j|\mathbf{X}_i, \mathbf{z}) = \mathcal{N}(h(\mathbf{X}_i, \mathbf{z}), cI)$. The encoder is a two-layer MLP. f and g share the first layer, and their second layers use different parameters. The decoder h is two-layer MLP. For simplicity and tractability, the implemented generator $Q\left(\overline{\mathbf{X}}|\mathbf{A}, \mathbf{X}\right)$ uses the same parameters across all nodes $v_i \in V$.

Here, we discuss how our proposed model distinguishes from the classical representation learning models on graphs. Previous methods such as EP-B (García-Durán & Niepert, 2017) and GraphSAGE (Hamilton et al., 2017) rely on reconstruction loss function between the central node and its neighbors' embeddings. EP-B aims to minimize the reconstruction error by optimizing the objective $\min \sum_{u \in V \setminus \{v\}} \left[\gamma + d(\widetilde{\mathbf{X}}_v, \mathbf{X}_v) - d(\widetilde{\mathbf{X}}_v, \mathbf{X}_u)\right]$ where $\mathbf{X}_v$ represents the target node; $\mathbf{X}_u$ denotes the neighbor nodes; $\widetilde{\mathbf{X}}_v = \mathrm{AGG}(\mathbf{X}_l | l \in \mathcal{N}(v))$ indicates the reconstruction from neighbors; and γ refers to the bias. Besides, GraphSAGE exploits the negative sampling to differentiate the representations of remote node-pairs. GraphSAGE enforce nearby nodes to have similar representations and to enforce disparate nodes to be distinct by minimizing the objective $\min -E_{u \sim \mathcal{N}(v)} \log\left((\sigma(\mathbf{X}_u^T \mathbf{X}_v))\right) - \lambda E_{v_n \sim P_n(v)} \log\left((\sigma(-\mathbf{X}_{v_n}^T \mathbf{X}_v))\right)$ where $\mathbf{X}_v$ denotes target node; $\mathbf{X}_u$ represents the neighbor node; $\mathbf{X}_{v_n}$ is disparate node; and $P_n(v)$ is the negative sampling. These approaches build upon the assumption that adjacent nodes share similar attributes. In contrast, our model does not rely on such assumption and instead generates the neighboring node features from the conditional distribution of central node representations. Given the target node, $\mathbf{X}_v$, our aim is to learn the conditional distribution of the neighbor nodes, $\mathbf{X}_u$. A comparison between the reconstruction-based representation learning on graphs and our proposed framework is illustrated in Figure 2.

Optimization of the MLE Now, we present how to optimize the MLE Eq.(4) using the feature matrix produced from the generator. Once the augmented feature matrix can be sampled from the generator, we can optimize the parameters of Eq.(4) in the following way. Firstly, the parameter $\phi = \{\psi, \varphi\}$ can be optimized by maximizing the ELBO of the generator (6), i.e., we train the generator. Secondly, the parameter θ is optimized by maximizing the MLE Eq.(4) with ϕ fixed, which is the conditional distribution of $\mathbf{Y}_k$ given $\mathbf{A}$ and $\overline{\mathbf{X}}$. In this way, we iteratively and alternately optimize the conditional MLE and ELBO.

Algorithm 1 Updating the GNN parameters θ using the initial feature matrix $\mathbf{X}$ and the generated feature matrix $\overline{\mathbf{X}}$ selected by the acquisition function

Input: Adjacency matrix $\mathbf{A}$, feature matrix $\mathbf{X}$
Output: Generated feature matrix $\overline{\mathbf{X}}$, and generator Q_ϕ
Train GNN P_θ using $\mathbf{A}$ and $\mathbf{X}$ for the number of initial GNN training iterations
Initialize U=-inf, $\overline{\mathbf{X}}$, Q_ϕ, $\overline{\mathbf{X}}'$, and Q'_ϕ
for $i = 1$ to the number of generator iterations **do**
    Train the generator Q_ϕ using $\mathbf{A}$ and $\mathbf{X}$
    Generate feature matrix $\overline{\mathbf{X}}$ using Q_ϕ
    Compute $U(\overline{\mathbf{X}})$ using P_θ and $\overline{\mathbf{X}}$
    if $U(\overline{\mathbf{X}}) > U$ **then**
        $U = U(\overline{\mathbf{X}})$
        if $i > N_{warmup}$ **then**
            Train GNN P_θ using $\mathbf{A}$ and $\overline{\mathbf{X}}$ for the number of continued GNN training iterations
            $\overline{\mathbf{X}}' = \overline{\mathbf{X}}, \quad Q'_\phi = Q_\phi$
$\overline{\mathbf{X}} = \overline{\mathbf{X}}', \quad Q_\phi = Q'_\phi$
return Generated feature matrix $\overline{\mathbf{X}}$, and generator Q_ϕ

In this paper, the MLE is formulated by a GNN model as follows:

$$P_\theta\left(\mathbf{Y}_k \mid \mathbf{A}, \overline{\mathbf{X}}\right) \propto -\overline{\mathcal{L}}(\theta|\mathbf{A}, \overline{\mathbf{X}}, \phi), \tag{7}$$

where $\overline{\mathcal{L}}(\theta|\mathbf{A}, \overline{\mathbf{X}}, \phi) = -\sum_{k\in\mathbf{T}}\sum_{f=1}^{C}\mathbf{Y}_{kf}\ln\left(\mathrm{softmax}\left(\mathrm{GNN}(\mathbf{A}, \overline{\mathbf{X}})\right)_{kf}\right)$. In addition, a practical trick is introduced which firstly trains GNN parameter θ with $\mathbf{A}$ and the initial feature matrix $\mathbf{X}$ for some iterations, and then updates it using $\mathbf{A}$ and the augmented feature matrix $\overline{\mathbf{X}}$. Later, section 2.3 will further present a close relationship between the inference of the generator and the optimization of the MLE.

2.3 Active Learning

After the training of the generator finishes, it contains an issue of using Eq.(4) for inference because Q may generate some rare samples from the side part of the distribution. This critical question makes the inferences Eq.(4) inefficient. Inspired by the active learning (Nielsen & Okoniewski, 2019), we introduce importance sampling to capture the suitable generated feature matrix and the corresponding generator, which improves the inference efficiency and accelerates the optimization of the MLE. During the importance sampling, the probability of each feature is proportional to its uncertainty evaluated by an acquisition function. We adopt the Bayesian Active Learning by Disagreement (BALD) acquisition function (Houlsby et al., 2011) to sample the most important inferences with the approximation from the Monte Carlo (MC) dropout samples as

$$U(\overline{\mathbf{X}}) \approx H\left[\frac{1}{N}\sum_{n=1}^{N}P_\theta\left(\mathbf{Y}_k|\overline{\mathbf{X}}, \boldsymbol{\omega}_n\right)\right] - \frac{1}{N}\sum_{n=1}^{N}H\left[P_\theta\left(\mathbf{Y}_k|\overline{\mathbf{X}}, \boldsymbol{\omega}_n\right)\right], \tag{8}$$

where N is the number of MC samples and $\boldsymbol{\omega}_n$ are the parameters of the network sampled for the n-th MC dropout sample. A high BLAD score indicates a network with high uncertainty about the generated feature matrix. So it tends to be selected to improve the GNN model. Finally, the overall algorithm framework is summarized in Algorithm 2.3, which assists the optimization of Eq.(4) and generates a suitable generated feature matrix and the corresponding generator for downstream tasks.

2.4 The Architecture of LA-GNN

In this section, we discuss the details of how to modify backbone models to exploit the augmented feature matrix produced by the generator. We use GCN, GAT, GCNII, and GRAND as the backbone network of the proposed framework and test them on semi-supervised node classification tasks.

When the above steps finish, both the initial data and the generated feature matrix are useful for the training of the backbone networks. Therefore, we mix the initial and generated feature representations

using concatenation operation. Then we feed the mixed representation to the next GNN layer to obtain our new model, named LA-GNN, where LA means local augmentation. Specifically, a 2-layer LA-GCN is defined as follows:

$$\mathbf{H}^{(2)} = \sigma\left(\hat{\mathbf{A}}\left(\sigma\left(\hat{\mathbf{A}}\mathbf{X}\mathbf{W}_1^{(1)}\right)\Big\|\sigma\left(\hat{\mathbf{A}}\overline{\mathbf{X}}_1\mathbf{W}_2^{(1)}\right)\Big\|\cdots\Big\|\sigma\left(\hat{\mathbf{A}}\overline{\mathbf{X}}_n\mathbf{W}_{n+1}^{(1)}\right)\right)\mathbf{W}^{(2)}\right), \tag{9}$$

where $\overline{\mathbf{X}}_i$ $(i = 1, 2, \cdots, n)$ is the augmented feature matrix produced by the generator, $\|$ denotes an operator of column-wise concatenation, $\mathbf{W}_i^{(1)}$ $(i = 1, 2, \cdots, n)$ denotes the parameters of the first LA-GCN layer, and $\mathbf{W}^{(2)}$ denotes the parameters of the second LA-GCN layer. Since GCNII (Chen et al., 2020) applies a fully-connected neural network on $\mathbf{X}$ to obtain a lower-dimensional initial representation $\mathbf{H}^{(0)}$ before the forward propagation, we apply a fully-connected neural network on $\mathbf{X}$ and $\overline{\mathbf{X}}$ to obtain $\mathbf{H}^{(0)}$ for LA-GCNII as follows:

$$\mathbf{H}^{(0)} = \sigma\left(\mathbf{X}\mathbf{W}_1^{(0)}\right)\|\sigma\left(\overline{\mathbf{X}}_1\mathbf{W}_2^{(0)}\right)\|\cdots\|\sigma\left(\overline{\mathbf{X}}_n\mathbf{W}_{n+1}^{(0)}\right). \tag{10}$$

$\mathbf{H}^{(0)}$ is fed into the next forward propagation layer. LA-GAT has a similar network architecture with LA-GCN, but the graph convolutional layer needs to be replaced by the graph attention layer.

2.5 Discussion

In this section, we discuss the motivation of this work and provide some analysis.

Local Augmentation vs. General Augmentation General image augmentation algorithms include geometric transformations, feature space augmentation, adversarial training, and generative adversarial networks (Shorten & Khoshgoftaar, 2019). It is impossible to apply geometric transformations directly to graph data augmentation since graphs are sensitive to node permutation. General adversarial training, feature space augmentation, and generative adversarial networks don't take the graph structure into account. Graphs consist of a set of identities with certain pairs of these identities connected by edges. We need to consider node features and the graph structure when designing the graph data augmentation framework. Our proposed method of local augmentation fully considers these two points. By extracting the neighbors' feature vectors, we have enough data points to learn the distribution. There are two benefits to designing local augmentation. First, by taking the sub-graph structure and feature representation associated with this sub-graph structure as input for the generative model, we can learn the information of the sub-graph structure. Second, the number of data points to learn the distribution depends on the node degree. This assures that we have enough data points compared with the general feature augmentation and we can learn a better distribution.

Complementing missing information Jia & Benson (2020) points out that some attribute information might be missing on a subset of vertices. By learning the distribution of node representations from the observed data, we can utilize the produced node representations from the generative model to complement the information missing in the nodes' attributes, which boosts the robustness of downstream tasks. And we show that our model still works in the scenario that nodes lose a certain percentage of attributes.

3 Related Work

Graph Neural Networks In general, convolution in the graph domain involves non-spectral (spatial) and spectral approaches. Non-spectral methods generalize convolutions operating on spatially close neighbors to the graph domain. Duvenaud et al. (2015) define the convolutional operation with a specific weight matrix for each node degree. Diffusion-convolution neural networks (DCNNs) (Atwood & Towsley, 2016) employs the transition matrix power series to propagate features. Niepert et al. (2016) extract local features with a fixed number of neighbors. Monet (Monti et al., 2017) proposes a unified framework to generalize CNN architectures on non-Euclidean domains. Spectral approaches define the convolution operations based on the spectral formulation. Bruna et al. (2014) specify the convolution operator with the spectrum of the graph Laplacian. Defferrard et al. (2016) approximate the filters by computing the Chebyshev polynomial recurrently, yielding fast and localized spectral ones. Graph Convolutional Network (GCN) (Kipf & Welling, 2017), one of

the most influential techniques, applies the first-order approximation of localized spectral filters to operate the connected neighbors. Recently, several methods (Abu-El-Haija et al., 2019; Liao et al., 2019) based on GCN have been proposed to obtain the higher-order filters. Besides, GAT (Veličković et al., 2018), Graph U-Nets (Gao & Ji, 2019) combine attention networks and pooling operation with GNN separately, which achieve state-of-the-art performance on node and link classification tasks. In this work, local augmentation can be applied on various backbone models to improve performance.

Graph Generative Models Generative models (Goodfellow et al., 2014; Kingma & Welling, 2013) are powerful tools of learning data distribution through unsupervised learning, and they have achieved tremendous success in various applications. Recently, researchers have proposed several interesting generative models for graph data generation. Variational graph auto-encoder (VGAE) (Kipf & Welling, 2016) makes use of latent variables and learns interpretable latent representations for undirected graphs. Salha et al. (2019) replace the GCN encoder in VGAE with a simple linear model and emphasize the effectiveness of a simple node encoding scheme. Xu et al. (2019) propose a generative model framework to learn node representations, by sampling graph generation sequences constructed from observed graph data. ConDgen (Yang et al., 2019) exploits the GCN encoder to handle the inherent challenges of flexible context-structure conditioning and permutation-invariant generation. Besides, some methods have been proposed to apply the graph generative models in various applications such as graph matching (Simonovsky & Komodakis, 2018), molecule design (Liu et al., 2018), retrosynthesis prediction (Shi et al., 2020) and chemical design (Samanta et al., 2018). Compared with these approaches mainly focusing on structure generation, our model takes full use of the power of the generative model for feature representation generation, which can serve as an enhanced technique for the downstream backbone models.

Topology-level Augmentation. Topology-level augmentation usually perturbs $\mathbf{A}$ to generate different graph structures, which can be formulated as $\mathbf{A}' = \mathcal{F}(\mathbf{A}, \mathbf{X})$, where $\mathcal{F}(\cdot)$ is a structure perturbation function. For example, DropEdge (Rong et al., 2020) considers $\mathcal{F}(\mathbf{A}, \mathbf{X}) = \mathbf{A} - \mathbf{A_s}$ which is independent of $\mathbf{X}$, where $\mathbf{A_s}$ is a sparse matrix consists of a subset of the original edges E. GAUG-O (Zhao et al., 2021) employs a non-close-form formulation of the function and leverages their proposed neural edge predictors to produce a different structure $\mathbf{A}'$ where $\mathbf{A}'_{ij} = \left\lfloor \frac{1}{1+e^{-(\log \mathbf{P}_{ij}+G)/\tau}} + \frac{1}{2} \right\rfloor$, $\mathbf{P}_{ij} = \alpha \mathbf{M}_{ij} + (1-\alpha)\mathbf{A}_{ij}$, $\mathbf{M} = \sigma\left(\mathbf{Z}\mathbf{Z}^T\right)$, $\mathbf{Z} = f\left(\mathbf{A}, f(\mathbf{A}, \mathbf{X})\right)$, τ is the temperature of Gumbel-Softmax distribution, $G \sim \mathrm{Gumbel}(0, 1)$ is a Gumbel random variate, and α is a hyperparameter mediating the influence of edge predictor on the original graph.

Feature-level Augmentation. Besides, feature-level augmentation function can be defines as $\mathbf{X}' = \mathcal{H}(\mathbf{A}, \mathbf{X})$, where $\mathcal{H}(\cdot)$ is a feature perturbation function. FLAG (Kong et al., 2020) defines the perturbation function as $\mathcal{H}(\mathbf{A}, \mathbf{X}) = \mathbf{X} + \boldsymbol{\delta}$ where perturbation $\boldsymbol{\delta}$ is updated iteratively during the adversarial training phase. G-GCN (plain) (Zhu et al., 2020) obtains the global attribute feature matrix $\mathbf{X}^{(a)} \in \mathbb{R}^{N \times d_a}$ through minimizing the objective $\prod_{v \in V} \prod_{a \in CA(v)} \frac{\exp\left(\mathbf{X}_v^{(a)} \cdot \mathbf{V}_a\right)}{\sum_{k \in U} \exp\left(\mathbf{X}_v^{(a)} \cdot \mathbf{V}_k\right)}$ where U is the set of all attributes, $CA(v)$ is the sampled context attributes of v, and $\mathbf{V} \in \mathbb{R}^{d_a \times F}$ denotes the parameters. Obviously, the perturbation function of G-GCN has no close-form solution. In this work, we propose a novel feature perturbation function, named local augmentation, whose differences with existing methods are presented. And the comparison of the details of various graph data augmentation techniques can be found in Table 1.

Table 1: Comparison of existing graph data augmentation.

Graph Data Augmentation			
Method	Considered Part	Type	Perturbed Part
DropEdge	$\mathbf{A}$	Sampling	$\mathbf{A}$
GAUG-O	$\mathbf{A}$&$\mathbf{X}$	Reconstruction	$\mathbf{A}$
FLAG	$\mathbf{X}$	Noise Injection	$\mathbf{X}$
G-GCN	$\mathbf{A}$&$\mathbf{X}$	Reconstruction	$\mathbf{X}$
Local Augmentation	$\mathbf{A}$&$\mathbf{X}$	Generation	$\mathbf{X}$

Table 2: Classification results on random split (%)

Dataset	APPNP	S^2GC	GCN	DropEdge-GCN	LA-GCN	GAT	LA-GAT	GCNII	LA-GCNII
Squirrel	21.6	21.3	22.5	21.9	23.2	24.2	28.2	25.3	**28.6**
Actor	32.1	27.8	26.2	26.5	27.0	27.2	27.4	31.9	**32.7**
Chameleon	33.0	30.2	25.1	25.0	28.9	34.8	**38.6**	30.2	32.5
Cornell	**58.7**	57.2	55.7	53.6	56.1	55.8	56.5	57.3	56.6

4 EXPERIMENTS

In this section, we evaluate the performance of our proposed model on semi-supervised node classification tasks on a variety of public graph datasets and compare our model with the state-of-the-art graph neural networks. We also carry out additional experiments to showcase the necessity of our design and its robustness to missing information.

4.1 DATASETS

We utilize seven public graph datasets (Cora, Citeseer, Pubmed, Squirrel, Actor, Chameleon, and Cornell) for semi-supervised node classification tasks. The details of these datasets can be found in the appendix.

Table 3: Classification results on fixed split (%)

Method	Cora	Citeseer	Pubmed
Chebyshev (Defferrard et al., 2016)	81.2	69.8	74.4
APPNP (Klicpera et al., 2019)	83.8	71.6	79.7
MixHop (Abu-El-Haija et al., 2019)	81.9	71.4	80.8
Graph U-net (Gao & Ji, 2019)	84.4	73.2	79.6
GSNN-M (Wang et al., 2020a)	83.9	72.2	79.1
S^2GC (Zhu & Koniusz, 2021)	83.5	73.6	80.2
GCN (Kipf & Welling, 2017)	81.6	70.3	78.9
G-GCN (Zhu et al., 2020)	83.7	71.3	80.9
DropEdge-GCN (Rong et al., 2020)	82.8	72.3	79.6
GAUG-O-GCN (Zhao et al., 2021)	83.6	73.3	79.3
LA-GCN	84.1	72.5	81.3
GAT (Veličković et al., 2018)	83.0	70.4	OOM
LA-GAT	83.9	72.3	OOM
GCNII (Chen et al., 2020)	85.2	73.1	80.0
LA-GCNII	85.2	73.7	81.6
GRAND (Feng et al., 2020)	85.4	75.4	82.7
LA-GRAND	**85.8**	**75.8**	**83.3**

4.2 SEMI-SUPERVISED NODE CLASSIFICATION

Baselines and Experimental Setup. We apply the standard fixed splits (Yang et al., 2016) on three datasets Cora, Citeseer, and Pubmed, with 20 nodes per class for training, 500 nodes for validation, and 1,000 nodes for testing. And we consider four backbones: GCN (Kipf & Welling, 2017), GAT (Veličković et al., 2018), GCNII (Chen et al., 2020), and GRAND (Feng et al., 2020) to evaluate our proposed framework and compare our model against state-of-the-art models including 1) backbone models: Chebyshev (Defferrard et al., 2016), GCN, GAT, APPNP (Klicpera et al., 2019), Graph U-net (Gao & Ji, 2019), MixHop (Abu-El-Haija et al., 2019), GCNII, GSNN-M (Wang et al., 2020a), S^2GC (Zhu & Koniusz, 2021), and GRAND and 2) feature-level and topology-level augmentation models: G-GNNs (Zhu et al., 2020), DropEdge (Rong et al., 2020) and GAUG-O (Zhao et al., 2021). For four datasets Squirrel, Actor, Chameleon, and Cornell, we take 10 random splits (Shchur et al., 2018) where 10%, 30%, and 60% of the date for training, validation, testing; measure the performance of GCN, GAT, GCNII, and corresponding modified models.

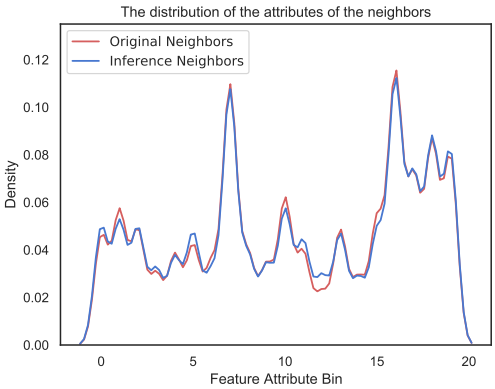


Figure 3: The distribution of the attribute bin of the inference neighbors vs. the distribution of the attribute bin of the original neighbors, with KL divergence = 0.0026. The value of each feature bin is the sum of the attribute values of multiple dimensions of the feature vector. We split the feature vector into multiple feature bins.

Results Figure 3 shows the distribution of the attributes of the original and inference neighbors, which can demonstrate our inference feature matrix follow the distribution of the initial feature matrix. For three datasets Cora, Citeseer, and Pubmed, we report the mean classification accuracy on the test nodes of all our models after 100 runs and report the values after running the experiments of their models with our server under their setting hyperparameters in their original papers. The results of the evaluation experiments are summarized in Tables 2, 3, and in the appendix, which demonstrate that the backbone models equipped with our method achieve the best performance across all the datasets except the Cornell dataset. More specifically, we can improve upon GCN by a margin of 2.5%, 2.2%, and 2.4% on Cora, Citeseer, and Pubmed respectively. Moreover, LA-GNN outperforms

other backbone models including GAT and GCNII as well as data augmentation models (Zhu et al., 2020; Rong et al., 2020; Zhao et al., 2021) on these citation network datasets.

4.3 Ablation Study

In this section, to demonstrate the effectiveness of our proposed generative framework, we conduct experiments that compare LA-GNN to several of its ablated variants without generative modeling. The results are shown in Table 4. "GCN + width" only increases the first network layer width for GCN and GCNII to match LA-GNN without giving generated samples as input. "+ concatenation" only replaces the generated feature matrix of LA-GNN with the original feature matrix of the central node. "+ plain neighborhood" replaces the generated feature matrix of LA-GNN with a neighborhood feature matrix where each row corresponds to the feature vector of the randomly sampled neighbor. The results show that the first two variants provide no notable improvement for the backbone models, and the third variant even results in degradation. By eliminating the possibility that these confounding factors irrelevant to our core approach may contribute to the final performance, it's evident that the performance gain in Table 2 and 3 are due to our proposed generative local augmentation framework.

Table 4: Effects of different components of our framework evaluated on the standard split of the Cora, Citeseer and Pubmed dataset.

Method	**Cora**	**Citeseer**	**Pubmed**
GCN	81.6	70.3	78.9
GCNII	85.2	73.1	80.0
GCN + width	82.0	71.4	79.5
GCN + concatenation	81.8	71.6	78.8
GCN + plain neighborhood	80.9	68.8	75.0
GCNII + width	85.1	73.1	80.2
GCNII + concatenation	85.2	73.3	80.2
GCNII + plain neighborhood	83.3	71.9	78.1
LA-GCN	84.1	72.5	81.3
LA-GCNII	85.2	73.7	81.6

4.4 Robustness to Missing Information

In this section, we conduct experiments to verify that our proposed framework can robustify downstream tasks against missing information in the feature attributes. Specifically, we mask a certain percentage of the attributes of each feature vector and use the same pipeline to do augmentation for the masked feature matrix. As shown in Table 5, we can see that as the mask ratio increases, the gap of the performance between the GCN and LA-GCN enlarges in most cases in Cora and Citeseer, which corroborates our insight discussed in Section 2.5. Since there exists large redundancy in the features of the Pubmed dataset, the performance of GCN and LA-GCN decreases little as the mask ratio increases. And so the gap of the performance between the GCN and LA-GCN does not enlarge in the Pubmed dataset.

Table 5: Summary of results on recovering study in terms of classification accuracy (%). $\downarrow$ means a decrease compared with the accuracy if features are not masked.

Dataset	Cora				Citeseer				Pubmed			
Mask Ratio	0.1	0.2	0.4	0.8	0.1	0.2	0.4	0.8	0.1	0.2	0.4	0.8
GCN	81.0 ($\downarrow$0.6)	80.6 ($\downarrow$1.0)	80.1 ($\downarrow$1.5)	76.0 ($\downarrow$5.6)	70.1 ($\downarrow$0.2)	69.3 ($\downarrow$1.0)	67.2 ($\downarrow$3.1)	61.0 ($\downarrow$9.3)	78.5 ($\downarrow$0.4)	78.5 ($\downarrow$0.4)	77.5 ($\downarrow$1.4)	76.9 ($\downarrow$2.0)
LA-GCN	83.5 ($\downarrow$0.6)	83.1 ($\downarrow$1.0)	81.6 ($\downarrow$2.5)	81.1 ($\downarrow$3.0)	72.2 ($\downarrow$0.3)	71.7 ($\downarrow$0.8)	69.3 ($\downarrow$3.2)	65.9 ($\downarrow$6.6)	81.4 ($\downarrow$0.1)	80.9 ($\downarrow$0.6)	80.5 ($\downarrow$1.0)	79.4 ($\downarrow$2.1)

5 Conclusion

We propose local augmentation, a brand-new technique that exploits the generative model to learn the conditional distribution of the central node's neighbors' feature representations given its representation. We can feed the generated feature matrix from a well-trained generative model to some modified backbone GNN models to enhance model performance. Experiments show that our model can improve performance across various GNN architectures and benchmark datasets. Besides, our model achieves new state-of-the-art results on various semi-supervised node classification tasks. One limitation of our proposed framework is that we do not exploit the 2-hop neighbors or use the random walk to find more related neighbors for the central node. And one future work is that we can extract more 2/3-hop neighbors if the central node's degree is small and learn the conditional distribution for random sampling nodes if the graph is large.

REFERENCES

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pp. 265–283, 2016.

Sami Abu-El-Haija, Bryan Perozzi, Amol Kapoor, Nazanin Alipourfard, Kristina Lerman, Hrayr Harutyunyan, Greg Ver Steeg, and Aram Galstyan. Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing. In *international conference on machine learning*, pp. 21–29. PMLR, 2019.

Antreas Antoniou, Amos Storkey, and Harrison Edwards. Data augmentation generative adversarial networks. *arXiv preprint arXiv:1711.04340*, 2017.

James Atwood and Don Towsley. Diffusion-convolutional neural networks. In *Advances in neural information processing systems*, pp. 1993–2001, 2016.

Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. In *International Conference on Learning Representations*, 2014.

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

Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. Autoaugment: Learning augmentation strategies from data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 113–123, 2019.

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*, 2016.

Zhijie Deng, Yinpeng Dong, and Jun Zhu. Batch virtual adversarial training for graph convolutional networks. *arXiv preprint arXiv:1902.09192*, 2019.

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

Marzieh Fadaee, Arianna Bisazza, and Christof Monz. Data augmentation for low-resource neural machine translation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, 2017.

Fuli Feng, Xiangnan He, Jie Tang, and Tat-Seng Chua. Graph adversarial training: Dynamically regularizing based on graph structure. *IEEE Transactions on Knowledge and Data Engineering*, 2019.

Wenzheng Feng, Jie Zhang, Yuxiao Dong, Yu Han, Huanbo Luan, Qian Xu, Qiang Yang, Evgeny Kharlamov, and Jie Tang. Graph random neural network for semi-supervised learning on graphs. In *NeurIPS'20*, 2020.

Hongyang Gao and Shuiwang Ji. Graph u-nets. In *international conference on machine learning*, pp. 2083–2092. PMLR, 2019.

Alberto García-Durán and Mathias Niepert. Learning graph representations with embedding propagation. In *Advances in Neural Information Processing Systems*, 2017.

Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. In *Advances in Neural Information Processing Systems*, 2014.

Shengnan Guo, Youfang Lin, Ning Feng, Chao Song, and Huaiyu Wan. Attention based spatial-temporal graph convolutional networks for traffic flow forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 922–929, 2019.

William L Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, 2017.

Neil Houlsby, Ferenc Huszár, Zoubin Ghahramani, and Máté Lengyel. Bayesian active learning for classification and preference learning. *arXiv preprint arXiv:1112.5745*, 2011.

Junteng Jia and Austion R Benson. Residual correlation in graph neural network regression. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 588–598, 2020.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Thomas N Kipf and Max Welling. Variational graph auto-encoders. *NIPS Workshop on Bayesian Deep Learning*, 2016.

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

Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. In *International Conference on Learning Representation*, 2019.

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

Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018.

Renjie Liao, Zhizhen Zhao, Raquel Urtasun, and Richard S Zemel. Lanczosnet: Multi-scale deep graph convolutional networks. In *International Conference on Learning Representations*, 2019.

Qi Liu, Miltiadis Allamanis, Marc Brockschmidt, and Alexander L Gaunt. Constrained graph variational autoencoders for molecule design. In *Advances in Neural Information Processing Systems*, 2018.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, 2013.

Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda, and Michael M Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 5115–5124, 2017.

Christopher Nielsen and Michal M Okoniewski. Gan data augmentation through active learning inspired sample acquisition. In *CVPR Workshops*, pp. 109–112, 2019.

Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In *International conference on machine learning*, pp. 2014–2023. PMLR, 2016.

Gaurav Pandey and Ambedkar Dukkipati. Variational methods for conditional multimodal deep learning. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pp. 308–315. IEEE, 2017.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, 2019.

Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. Geom-gcn: Geometric graph convolutional networks. In *International Conference on Learning Representations*, 2020.

Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 701–710, 2014.

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

Benedek Rozemberczki, Carl Allen, and Rik Sarkar. Multi-scale attributed node embedding. *Journal of Complex Networks*, 9(2):cnab014, 2021.

Guillaume Salha, Romain Hennequin, and Michalis Vazirgiannis. Keep it simple: Graph autoencoders without graph convolutional networks. Workshop on Graph Representation Learning, 33rd Conference on Neural Information Processing Systems (NeurIPS), 2019.

Bidisha Samanta, Abir De, Niloy Ganguly, and Manuel Gomez-Rodriguez. Designing random graph models using variational autoencoders with applications to chemical design. *arXiv preprint arXiv:1802.05283*, 2018.

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

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

Chence Shi, Minkai Xu, Hongyu Guo, Ming Zhang, and Jian Tang. A graph to graphs framework for retrosynthesis prediction. In *International Conference on Machine Learning*, pp. 8818–8827. PMLR, 2020.

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

Martin Simonovsky and Nikos Komodakis. Graphvae: Towards generation of small graphs using variational autoencoders. In *International Conference on Artificial Neural Networks*, pp. 412–422. Springer, 2018.

Kihyuk Sohn, Honglak Lee, and Xinchen Yan. Learning structured output representation using deep conditional generative models. *Advances in neural information processing systems*, 28:3483–3491, 2015.

Jie Tang, Jimeng Sun, Chi Wang, and Zi Yang. Social influence analysis in large-scale networks. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 807–816, 2009.

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

Haibo Wang, Chuan Zhou, Xin Chen, Jia Wu, Shirui Pan, and Jilong Wang. Graph stochastic neural networks for semi-supervised learning. In *Advances in Neural Information Processing Systems*, 2020a.

Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, Chao Ma, Lingfan Yu, Yu Gai, Tianjun Xiao, Tong He, George Karypis, Jinyang Li, and Zheng Zhang. Deep graph library: A graph-centric, highly-performant package for graph neural networks. *arXiv preprint arXiv:1909.01315*, 2019.

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

Da Xu, Chuanwei Ruan, Kamiya Motwani, Evren Korpeoglu, Sushant Kumar, and Kannan Achan. Generative graph convolutional network for growing graphs. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 3167–3171. IEEE, 2019.

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

Carl Yang, Peiye Zhuang, Wenhan Shi, Alan Luu, and Pan Li. Conditional structure generation through graph variational generative adversarial nets. In *NeurIPS*, pp. 1338–1349, 2019.

Zhilin Yang, William Cohen, and Ruslan Salakhudinov. Revisiting semi-supervised learning with graph embeddings. In *International Conference on Machine Learning*, pp. 40–48, 2016.

Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and 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*, pp. 974–983, 2018.

Tong Zhao, Yozen Liu, Leonardo Neves, Oliver Woodford, Meng Jiang, and Neil Shah. Data augmentation for graph neural networks. In *The Thirty-Fifth AAAI Conference on Artificial Intelligence*, 2021.

Danhao Zhu, Xin-Yu Dai, and Jiajun Chen. Pre-train and learn: Preserve global information for graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020.

Hao Zhu and Piotr Koniusz. Simple spectral graph convolution. In *International Conference on Learning Representations*, 2021.

A PROOFS OF EQ.(6) AND EQ.(7)

A.1 PROOF OF EQ.(6): THE DERIVATION OF THE ELBO OF THE GENERATOR

We give more details of the derivation of the generator ELBO as follows:

$$
\begin{aligned}
\log p_\psi(\mathbf{X}_j|\mathbf{X}_i) &= \int q_\varphi(\mathbf{z}|\mathbf{X}_j,\mathbf{X}_i)\log p_\psi(\mathbf{X}_i|\mathbf{X}_i)\mathrm{d}\mathbf{z} \\
&= \int q_\varphi(\mathbf{z}|\mathbf{X}_j,\mathbf{X}_i)\log \frac{p_\psi(\mathbf{X}_j,\mathbf{X}_i)}{p_\psi(\mathbf{X}_i)}\mathrm{d}\mathbf{z} \\
&= \int q_\varphi(\mathbf{z}|\mathbf{X}_j,\mathbf{X}_i)\log \frac{p_\psi(\mathbf{X}_j,\mathbf{X}_i)p_\psi(\mathbf{X}_i,\mathbf{X}_i,\mathbf{z})}{p_\psi(\mathbf{X}_i)p_\psi(\mathbf{X}_i,\mathbf{X}_i,\mathbf{z})}\mathrm{d}\mathbf{z} \\
&= \int q_\varphi(\mathbf{z}|\mathbf{X}_i,\mathbf{X}_i)\log \frac{p_\psi(\mathbf{X}_j,\mathbf{X}_i,\mathbf{z})}{p_\psi(\mathbf{X}_i)}\frac{1}{\frac{p_\psi(\mathbf{X}_j,\mathbf{X}_i,\mathbf{z})}{p_\psi(\mathbf{X}_j,\mathbf{X}_i)}}\mathrm{d}\mathbf{z} \\
&= \int q_\varphi(\mathbf{z}|\mathbf{X}_j,\mathbf{X}_i)\log \frac{p_\psi(\mathbf{X}_j,\mathbf{z}|\mathbf{X}_i)}{p_\psi(\mathbf{z}|\mathbf{X}_j,\mathbf{X}_i)}\mathrm{d}\mathbf{z} \\
&= \int q_\varphi(\mathbf{z}|\mathbf{X}_j,\mathbf{X}_i)\log \frac{p_\psi(\mathbf{X}_j,\mathbf{z}|\mathbf{X}_i)}{p_\psi(\mathbf{z}|\mathbf{X}_j,\mathbf{X}_i)}\frac{q_\varphi(\mathbf{z}|\mathbf{X}_j,\mathbf{X}_i)}{q_\varphi(\mathbf{z}|\mathbf{X}_j,\mathbf{X}_i)}\mathrm{d}\mathbf{z} \\
&= \int q_\varphi(\mathbf{z}|\mathbf{X}_j,\mathbf{X}_i)\left(\log \frac{p_\psi(\mathbf{X}_j,\mathbf{z}|\mathbf{X}_i)}{q_\varphi(\mathbf{z}|\mathbf{X}_j,\mathbf{X}_i)} + \log \frac{q_\varphi(\mathbf{z}|\mathbf{X}_j,\mathbf{X}_i)}{p_\psi(\mathbf{z}|\mathbf{X}_j,\mathbf{X}_i)}\right)\mathrm{d}\mathbf{z} \\
&= \int q_\varphi(\mathbf{z}|\mathbf{X}_j,\mathbf{X}_i)\log \frac{p_\psi(\mathbf{X}_j,\mathbf{z}|\mathbf{X}_i)}{q_\varphi(\mathbf{z}|\mathbf{X}_j,\mathbf{X}_i)}\mathrm{d}\mathbf{z} + KL(q_\varphi(\mathbf{z}|\mathbf{X}_j,\mathbf{X}_i)||p_\psi(\mathbf{z}|\mathbf{X}_j,\mathbf{X}_i)) \\
&\geq \int q_\varphi(\mathbf{z}|\mathbf{X}_j,\mathbf{X}_i)\log \frac{p_\psi(\mathbf{X}_j,\mathbf{z}|\mathbf{X}_i)}{q_\varphi(\mathbf{z}|\mathbf{X}_j,\mathbf{X}_i)}\mathrm{d}\mathbf{z}
\end{aligned}
$$

A.2 PROOF OF EQ.(7)

$$
\begin{aligned}
L_{ELBO} &= \int q_\varphi(\mathbf{z}|\mathbf{X}_j,\mathbf{X}_i)\log \frac{p_\psi(\mathbf{X}_j,\mathbf{z}|\mathbf{X}_i)}{q_\varphi(\mathbf{z}|\mathbf{X}_j,\mathbf{X}_i)}\mathrm{d}\mathbf{z} \\
&= \int q_\varphi(\mathbf{z}|\mathbf{X}_j,\mathbf{X}_i)\log \frac{p_\psi(\mathbf{X}_j,\mathbf{X}_i,\mathbf{z})}{q_\varphi(\mathbf{z}|\mathbf{X}_j,\mathbf{X}_i)p_\psi(\mathbf{X}_i)}\mathrm{d}\mathbf{z} \\
&= \int q_\varphi(\mathbf{z}|\mathbf{X}_j,\mathbf{X}_i)\log \frac{p_\psi(\mathbf{X}_j|\mathbf{X}_i,\mathbf{z})p_\psi(\mathbf{X}_i,\mathbf{z})}{q_\varphi(\mathbf{z}|\mathbf{X}_j,\mathbf{X}_i)p_\psi(\mathbf{X}_i)}\mathrm{d}\mathbf{z} \\
&= \int q_\varphi(\mathbf{z}|\mathbf{X}_j,\mathbf{X}_i)\log \frac{p_\psi(\mathbf{X}_j|\mathbf{X}_i,\mathbf{z})p_\psi(\mathbf{z}|\mathbf{X}_i)}{q_\varphi(\mathbf{z}|\mathbf{X}_j,\mathbf{X}_i)}\mathrm{d}\mathbf{z} \\
&= \int q_\varphi(\mathbf{z}|\mathbf{X}_j,\mathbf{X}_i)\log \frac{p_\psi(\mathbf{z}|\mathbf{X}_i)}{q_\varphi(\mathbf{z}|\mathbf{X}_j,\mathbf{X}_i)}\mathrm{d}\mathbf{z} + \int q_\varphi(\mathbf{z}|\mathbf{X}_j,\mathbf{X}_i)\log p_\psi(\mathbf{X}_j|\mathbf{X}_i,\mathbf{z})\mathrm{d}\mathbf{z} \\
&= -KL(q_\varphi(\mathbf{z}|\mathbf{X}_j,\mathbf{X}_i)||p_\psi(\mathbf{z}|\mathbf{X}_i)) + \int q_\varphi(\mathbf{z}|\mathbf{X}_j,\mathbf{X}_i)\log p_\psi(\mathbf{X}_j|\mathbf{X}_i,\mathbf{z})\mathrm{d}\mathbf{z}
\end{aligned}
$$

B REPRODUCIBILITY

B.1 DATASETS DETAILS

Cora, Citeseer, and Pubmed are standard citation network benchmark datasets Sen et al. (2008). In these datasets, nodes represent documents, and edges denote citations; node feature corresponds to elements of a bag-of-words representation of a document, and node label corresponds to one of the academic topics. Besides, we utilize four datasets used in Pei et al. (2020) for evaluation. Chameleon and squirrel are two page-page networks on specific topics in Wikipedia Rozemberczki et al. (2021). In these datasets, nodes represent web pages, and edges denote mutual links between pages; node features correspond to several informative nouns in the Wikipedia pages and labels correspond to the number of the average monthly traffic of the web page. WebKB[1] is a webpage dataset collected from various universities. We use the one subdataset of it, Cornell. In this dataset, nodes represent web pages, and edges are hyperlinks between them; node features correspond to the bag-of-words representation of web pages and labels correspond to five categories, student, project, course, staff, and faculty. Film dataset is the actor-only induced subgraph of the film-directoractor-writer network Tang et al. (2009). In this dataset, Nodes represent actors, and edges denote co-occurrence on the same Wikipedia page; node features correspond to some keywords in the Wikipedia pages and labels correspond to five categories in terms of words of actor's Wikipedia. All the dataset statistics are summarized in Table 6.

Table 6: Datasets statistics

Dataset	Cora	Cite.	Pubm.	Cham.	Squi.	Actor	Corn.
# Nodes	2708	3327	19717	2277	5201	7600	183
# Edges	5429	4732	44338	36101	217073	33544	295
# Features	1433	3703	500	2325	2089	931	1703
# Classes	7	6	3	5	5	5	5

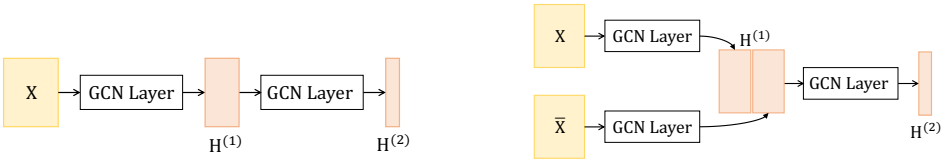
B.2 IMPLEMENTATION DETAILS

We use Pytorch (Paszke et al., 2019) to implement LA-GNNs. The codes of S^2GC (Zhu & Koniusz, 2021), LA-GCN, LA-GAT, LA-GCNII, LA-GRAND, and DropEdge-GCN are implemented referring to Pytorch implementation of S^2GC[2], GCN[3] (Kipf & Welling, 2017), GAT[4] (Veličković et al., 2018), GCNII[5] (Chen et al., 2020) GRAND[6] (Feng et al., 2020), and DropEdge-GCN[7] (Rong et al., 2020). Besides, we implement APPNP (Klicpera et al., 2019) with DGL (Wang et al., 2019) version of APPNP[8]. The datasets Cora, Citeseer, Pubmed are downloaded from TensorFlow (Abadi et al., 2016) implementation of GCN[9], and the datasets Chameleon, Squirrel, Actor, and Cornell are downloaded from the implementation of Geom-GCN[10](Pei et al., 2020). All the experiments in this work are conducted on a single NVIDIA Tesla V100 with 32GB memory size. The operating system behind the Docker where the experiments are running is Red Hat 4.8.2-16. And the software that we use for experiments are Python 3.6.8, numpy 1.19.2, sklearn 0.0, scipy 1.5.4, networkx 2.5.1, torch 1.6.0, torchvision 0.7.0, CUDA 10.2.89, and CUDNN 8.0.2.

[1] http://www.cs.cmu.edu/afs/cs.cmu.edu/project/theo-11/www/wwkb
[2] https://github.com/allenhaozhu/SSGC
[3] https://github.com/tkipf/pygcn
[4] https://github.com/Diego999/pyGAT
[5] https://github.com/chennnM/GCNII
[6] https://github.com/THUDM/GRAND
[7] https://github.com/DropEdge/DropEdge
[8] https://github.com/dmlc/dgl/tree/master/examples/pytorch/appnp
[9] https://github.com/tkipf/gcn/tree/master/gcn/data
[10] https://github.com/graphdml-uiuc-jlu/geom-gcn/tree/master/new_data

B.3 HYPERPARAMETER DETAILS

LA-GNNs introduce an additional parameter, that is the hidden layer for generated feature matrix $\overline{\mathbf{X}}$ before concatenation. The difference of architectures between GCN and LA-GCN can be found in Figure 4, and the LA-GCNII architecture can be found in Figure 5.



(a) GCN (b) LA-GCN

Figure 4: GCN and LA-GCN architectures. The difference between GCN and LA-GCN architectures is that the LA-GCN has an additional convolutional layer for $\overline{\mathbf{X}}$ and it uses a concatenation operation to mix the hidden representations.

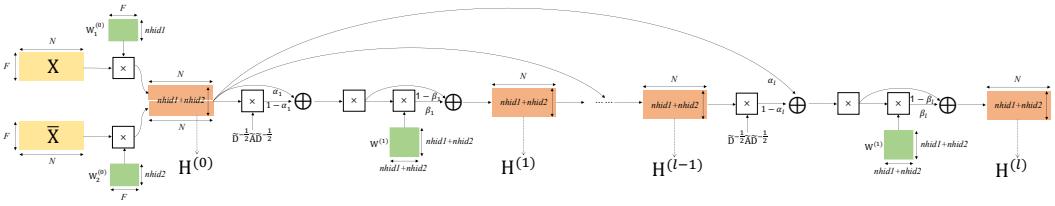


Figure 5: LA-GCNII architecture. The difference between GCNII and LA-GCNII is that the LA-GCNII has an additional MLP layer for $\overline{\mathbf{X}}$ and it uses a concatenation operation to mix the hidden representations.

The difference of hyperparameters between the GCN and LA-GCN is only the hidden layer size before concatenation. For the LA-GCNII, LA-GAT, LA-GRAND, we tune the hyperparameters in the same way as described in their original papers with validation set.