Local Augmentation for Graph Neural Networks

Songtao Liu [1] [†] Rex Ying [2] Hanze Dong [3] Lanqing Li [4] Tingyang Xu [4] Yu Rong [4] Peilin Zhao [4]
Junzhou Huang [4] Dinghao Wu [1]

Abstract

Graph Neural Networks (GNNs) have achieved remarkable performance on graph-based tasks. The key idea for GNNs is to obtain informative representation through aggregating information from local neighborhoods. However, it remains an open question whether the neighborhood information is adequately aggregated for learning representations of nodes with few neighbors. To address this, we propose a simple and efficient data augmentation strategy, local augmentation, to learn the distribution of the node representations of the neighbors conditioned on the central node's representation and enhance GNN's expressive power with generated features. Local augmentation is a general framework that can be applied to any GNN model in a plug-and-play manner. It samples feature vectors associated with each node from the learned conditional distribution as additional input for the backbone model at each training iteration. Extensive experiments and analyses show that local augmentation consistently yields performance improvement when applied to various GNN architectures across a diverse set of benchmarks. For example, experiments show that plugging in local augmentation to GCN and GAT improves by an average of 3.4% and 1.6% in terms of test accuracy on Cora, Citeseer, and Pubmed. Besides, our experimental results on large graphs (OGB) show that our model consistently improves performance over backbones.

1. Introduction

Graph Neural Networks (GNNs) and their variants (Kipf & Welling, 2017; Hamilton et al., 2017; Veličković et al., 2018) have achieved state-of-the-art performance on a variety of graph-based tasks, including recommendation system (Ying et al., 2018), drug discovery (Dai et al., 2019) and traffic prediction (Guo et al., 2019). The core of GNNs is to employ a message-passing mechanism that passes and aggregates information from the local neighborhood to generate informative representations.

Recent development of deep GNNs, such as JKnet (Xu et al., 2018), GCNII (Chen et al., 2020b), and RevGNN-Deep (Li et al., 2021) adds the output of shallow layers to the deep layers with a residual-style design, to preserve the locality information of node representations (Chen et al., 2020b). Moreover, recent studies (Zeng et al., 2021; Zhang & Li, 2021; Wijesinghe & Wang, 2022) utilize structural information of the local neighborhood to design efficient message-passing aggregation schemes to enhance the expressive power of GNNs. These works demonstrate that local information plays a significant role in training GNN models and designing powerful GNNs.

Despite advances of GNNs in learning node representations from the local neighborhood, it remains an open problem whether the local neighborhood information is sufficient to obtain effective node representations, especially for nodes with *limited* number of neighbors. We argue that the *limited* number of neighbors in the local neighborhood restricts the expressive power of GNNs and hinders their performance, especially in sample-starving cases where some nodes have very few neighbors. Stacking graph layers to enlarge the receptive field can incorporate multi-hop neighboring information but leads to over-smoothing (Li et al., 2018) without residual connection to the input, and is not a direct solution to address this issue. Existing works on GNN model architecture cannot tackle the problem that the very limited neighbors are unfavorable to learning node representations. Therefore, here we focus on enriching the local information for low-degree nodes to obtain effective representations.

One promising solution is to generate more samples in the local neighborhood via data augmentation. Data augmentation has been well-studied in computer vision (Shorten &

[†] Part of the work was done during Songtao's internship at Tencent AI Lab [1] The Pennsylvania State University [2] Stanford University [3] Hong Kong University of Science and Technology [4] Tencent AI Lab. Correspondence to: Songtao Liu <skl5761@psu.edu>, Rex Ying <rexying@stanford.edu>, Lanqing Li <lanqingli1993@gmail.com>, Dinghao Wu <dinghao@psu.edu>.

Proceedings of the 39th International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022. Copyright 2022 by the author(s).

Khoshgoftaar, 2019; Cubuk et al., 2019; Zhao et al., 2019; Dong et al., 2022) and natural language processing (Fadaee et al., 2017; Şahin & Steedman, 2019; Xia et al., 2019), but remains under-explored on graph-structured data. Existing graph data augmentation approaches only perturb at the topology-level and feature-level from a global perspective, which can be divided into two categories: topology-level augmentation (Rong et al., 2020; Zhao et al., 2021) and feature-level augmentation (Deng et al., 2019; Feng et al., 2019; Kong et al., 2020; Fang et al., 2021). Topology-level augmentation perturbs the adjacency matrix, yielding different graph structures. On the other hand, existing feature-level augmentation (Deng et al., 2019; Feng et al., 2019; Kong et al., 2020) exploits perturbation of node attributes guided by adversarial training to boost generalization. These augmentation techniques have a prominent drawback: they focus on global augmentation concerning the properties of the whole distribution of the graph rather than a single node, and neglect the local information of the neighborhood.

In this work, in order to promote the aggregation scheme with more generated samples in the local neighborhood, we propose a novel and efficient data augmentation framework: **L**ocal **A**ugmentation for **G**raph **N**eural **N**etworks (LA-GNNs). The term "local augmentation" refers to generating neighborhood features via a generative model **conditioned on local structures and node features**. Specifically, our proposed framework includes a pre-training step, which learns the conditional distribution of the connected neighbors' node features given one center node's features via a generative model. As shown in Fig. 1, we then exploit this distribution to generate feature vectors associated with this center node as additional input at each training iteration. Furthermore, we decouple the pre-training of the generative model and downstream GNN training, allowing our data augmentation model to be applied to any GNN model in a plug-and-play manner.

We verify the effectiveness of LAGNNs on three standard citation networks (Cora, Citeseer, Pubmed) and Open Graph Benchmark (OGB) (Hu et al., 2020). Extensive experimental results on semi-supervised node classification show that our local augmentation achieves new stat-of-the-art performance: LAGCN and LAGAT achieve up to by an average of 3.4% and 1.6% in terms of test accuracy over GCN and GAT respectively on Cora, Citeseer, and Pubmed. LAGNN also obtains superior performance on large-scale OGB datasets. We show that our model improves 1.7% and 0.2% of test accuracy on Pubmed for nodes with degrees in $[2, 5]$ and $[6, 20]$ respectively. Besides, our local augmentation model outperforms other feature/topology-level augmentation models, such as G-GNN (Zhu et al., 2020), DropEdge (Rong et al., 2020), GRAND (Feng et al., 2020), and GAUG (Zhao et al., 2021) on node classification tasks, which demonstrates the superiority of our model.

Contributions.

- We propose a general augmentation strategy to generate more features in the local neighborhood to enhance the expressive power of existing GNNs;
- We explore a new direction on pre-training generative models for graphs to improve downstream task performance;
- Our proposed framework is flexible and can be applied to various popular backbones. Extensive experimental results demonstrate that our proposed framework could improve the performance of GNN variants on different benchmark datasets.

2. Preliminaries

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

Graph Neural Networks. Many popular Graph Neural Networks (GNNs) directly operate on the graph structure and capture the dependence of graphs via message passing between the nodes of a graph. They repeatedly aggregate the representations of immediate neighbors $\mathcal{N}_v$ of node v and combine the aggregated information and its representation vector to obtain a representation vector $\boldsymbol{h}_v$. The k-th layer of the GNN message-passing scheme is:

$$\boldsymbol{h}_v^{(k)} = \mathsf{COM}\left(\boldsymbol{h}_v^{(k-1)}, \mathsf{AGG}\left(\left\{\boldsymbol{h}_u^{(k-1)}, \boldsymbol{e}_{u,v} | u \in \mathcal{N}_v\right\}\right)\right), \tag{1}$$

where $\mathsf{COM}(\cdot)$ and $\mathsf{AGG}(\cdot)$ denotes COMBINE and AGGREGATE functions respectively, $\boldsymbol{h}_v^{(k)}$ is the representation vector of node v in the k-th layer, and $\boldsymbol{e}_{u,v}$ is the edge vector between node u and node v. Specifically, $\boldsymbol{h}_v^{(0)} = \boldsymbol{X}_v$.

3. Local Augmentation for Graph Neural Networks (LAGNN)

In this section, we first present how to generate more samples in the local neighborhood via a generative model. Then we show how to decouple the pre-training of the generative model and downstream GNN training from a probabilistic perspective, so that our local augmentation model can be

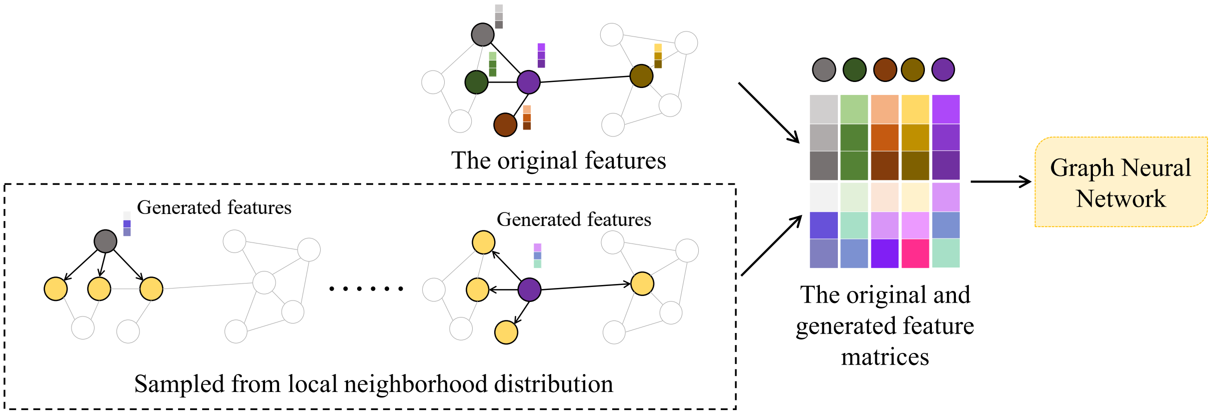

The original features
Generated features
Generated features
Sampled from local neighborhood distribution
The original and generated feature matrices
Graph Neural Network

Figure 1. A schematic depiction of our local augmentation. The yellow circles on the graph correspond to the neighbor nodes. Assume we have learned the distribution of the local neighborhood. We generate features from the local neighborhood distribution. And then we take the original features and the generated features as input for downstream GNNs.

applied to any GNN model in a plug-and-play manner. We then introduce the architecture of LA-GNNs and the training details. The overall framework is illustrated in Fig. 1.

3.1. Local Augmentation

Motivation. Existing GNNs focus on designing a message-passing scheme to exploit local information to obtain node representations. We explore a new direction in that we can generate more samples in the local neighborhood, especially for nodes with few neighbors, to enhance the expressive power of various GNNs. In order to produce more samples in a node v's neighborhood $\mathcal{N}_v$, we need to know the distribution of the node representation of its neighbors. As this distribution is related to the center node v, we can learn it conditioned on the center node's representation via a generative model.

Benefits. Compared to training generative models for each node, there are three benefits to train a single generative model for all nodes 1) Learning the conditional distribution across all nodes on the graph via a generative model reduces the computational cost. 2) In the generation stage, we can apply a specific node's feature vector as input (condition) and generate feature vectors associated with this node. 3) It has better scalability and generalization. For a new node added to a dynamic graph, we can directly generate feature vectors without re-training a new generative model since the generative model contains such generalized information. So our local augmentation model can be applied to inductive learning tasks, such as graph classification.

Approach. We exploit the conditional variational autoencoder (CVAE) (Kingma & Welling, 2013; Sohn et al., 2015) to learn the conditional distribution of the node features of connected neighbors $u(u \in \mathcal{N}_v)$ given the center node v. In our CVAE setting, we use $\boldsymbol{X}_v$ as a condition since the distribution of $\boldsymbol{X}_u(u \in \mathcal{N}_v)$ is related to $\boldsymbol{X}_v$. Following Sohn et al. (2015), the latent variable $\mathbf{z}$ is generated from the prior distribution $p_\theta(\mathbf{z}|\boldsymbol{X}_v)$ and the data $\boldsymbol{X}_u$ is generated by the generative distribution $p_\theta(\boldsymbol{X}|\boldsymbol{X}_v, \mathbf{z})$ conditioned on $\mathbf{z}$ and $\boldsymbol{X}_v$: $\mathbf{z} \sim p_\theta(\mathbf{z}|\boldsymbol{X}_v)$, $\boldsymbol{X}_u \sim p_\theta(\boldsymbol{X}|\boldsymbol{X}_v, \boldsymbol{z}_v)$. Let ϕ denote the variational parameters and θ represent the generative parameters, we have

$$\begin{aligned}\log p_\theta\left(\boldsymbol{X}_u|\boldsymbol{X}_v\right) = &\int q_\phi(\mathbf{z}|\boldsymbol{X}_u, \boldsymbol{X}_v) \log \frac{p_\theta(\boldsymbol{X}_u, \mathbf{z}|\boldsymbol{X}_v)}{q_\phi(\mathbf{z}|\boldsymbol{X}_u, \boldsymbol{X}_v)} \mathrm{d}\mathbf{z} \\ &+ KL\left(q_\phi(\mathbf{z}|\boldsymbol{X}_u, \boldsymbol{X}_v) \| p_\theta(\mathbf{z}|\boldsymbol{X}_u, \boldsymbol{X}_v)\right) \\ \geq &\int q_\phi(\mathbf{z}|\boldsymbol{X}_u, \boldsymbol{X}_v) \log \frac{p_\theta(\boldsymbol{X}_u, \mathbf{z}|\boldsymbol{X}_v)}{q_\phi(\mathbf{z}|\boldsymbol{X}_u, \boldsymbol{X}_v)} \mathrm{d}\mathbf{z},\end{aligned}$$

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

$$\begin{aligned}\mathcal{L}(\boldsymbol{X}_u, \boldsymbol{X}_v; \theta, \phi) = &- KL(q_\phi(\mathbf{z}|\boldsymbol{X}_u, \boldsymbol{X}_v) \| p_\theta(\mathbf{z}|\boldsymbol{X}_v)) \\ &+ \frac{1}{L}\sum_{l=1}^{L} \log p_\theta(\boldsymbol{X}_u|\boldsymbol{X}_v, \mathbf{z}^{(l)})\end{aligned} \tag{2}$$

where $\mathbf{z}^{(l)} = g_\phi(\boldsymbol{X}_v, \boldsymbol{X}_u, \epsilon^{(l)})$, $\epsilon^{(l)} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and L is the number of neighbors of node v. Note that as we have discussed before, we just train one CVAE for all nodes. In the training stage, the objective is to use the neighboring pairs ($\boldsymbol{X}_v$, $\boldsymbol{X}_u$, $u \in \mathcal{N}_v$) as input to maximize the ELBO, i.e., Eq. (2). In the generation stage, we use the node feature $\boldsymbol{X}_v$ as the condition and sample a latent variable $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ as input for the decoder. Then we can get generated feature vector $\overline{\boldsymbol{X}}_v$ associated with node v.

Discussion. When learning the distribution of the neighbors' feature conditioned on the central node, we do not consider the effect of other nodes connected to each neighbor on the neighbors' feature. If we regard the center node as a parent node, and its neighbors as the children nodes of the center node, then our assumption is similar to the Causal Markov Condition in a Bayesian network (Hausman & Woodward, 1999): the distribution of the neighbor's features is independent of its non-descendants given its parent node. This assumption is important and common in the literature of the probabilistic graphical model. The advantage is that this assumption avoids the exponential complexity of conditioning on multi-hop neighbors, significantly improving scalability. Our experimental results show that our method still achieves remarkable performance across all benchmarks, thanks to the expressive power of deep generative models (similar to how the assumption of variational inference does not limit the performance of deep VAEs in real datasets).

3.2. Decoupling the Generative Model Training from Downstream Graph Learning

Most existing GNN models follow the message passing mechanism (Gilmer et al., 2017) and can be regarded as a learned classification or regression function. In order to make predictions, GNN models need to estimate the posterior distribution $P_\Theta(\cdot|\boldsymbol{A}, \boldsymbol{X})$ with respect to the graph structure $\boldsymbol{A}$ and feature matrix $\boldsymbol{X}$. For example, $\cdot$ can be class labels $\boldsymbol{Y}$ on the node classification task. We can use Maximum Likelihood Estimation (MLE) to estimate the parameter Θ by optimizing the following likelihood function:

$$\max \prod_i P_\Theta\left(\cdot|\boldsymbol{A}, \boldsymbol{X}\right), \tag{3}$$

where i represents the i-th data point in the training dataset. In our local augmentation model, in order to further improve the expressive power of GNNs, we introduce a generated feature vector $\overline{\boldsymbol{X}}_v$ for the center node v by using $\boldsymbol{X}_v$ as input condition and sampling from the generative model. Let $\overline{\boldsymbol{X}}$ denote the generated feature matrix where the j-th row corresponds to the generated feature vector $\overline{\boldsymbol{X}}_j$. We incorporate $\overline{\boldsymbol{X}}$ in Eq. (3) and rewrite it as follows:

$$\max \prod_i \int_{\overline{\boldsymbol{X}}} P_\Theta\left(\cdot, \overline{\boldsymbol{X}}|\boldsymbol{A}, \boldsymbol{X}\right). \tag{4}$$

For Bayesian tractability, we decompose P_Θ in Eq.(4) as a product of two posterior probabilities:

$$P_{\Theta,\Phi}(\cdot, \overline{\boldsymbol{X}}|\boldsymbol{A}, \boldsymbol{X}) := P_\Theta(\cdot|\boldsymbol{A}, \boldsymbol{X}, \overline{\boldsymbol{X}})Q_\Phi(\overline{\boldsymbol{X}}|\boldsymbol{A}, \boldsymbol{X}), \tag{5}$$

where $P_\Theta(\cdot|\boldsymbol{A}, \boldsymbol{X}, \overline{\boldsymbol{X}})$ and $Q_\Phi(\overline{\boldsymbol{X}}|\boldsymbol{A}, \boldsymbol{X})$ denote the probabilistic distributions approximated by the GNN models and the generative model respectively, parameterized by Θ and Φ. By doing this, we can decouple our proposed local augmentation and the specific graph learning, allowing our augmentation model to be applied to various GNN models with only one-time pre-training for the generative model. Therefore, local augmentation can be regarded as an unsupervised pre-training model prior to the GNN training. The representation power of Eq. (5) is superior than that of a single predictor $P_\Theta\left(\boldsymbol{Y}_k|\boldsymbol{A}, \boldsymbol{X}\right)$ since we provide GNN models with more generated samples in the local neighborhood.

3.3. Architecture

In this section, we present the details of how to train GNNs with generated samples from our local augmentation model as additional input. To illustrate the effectiveness of our local augmentation model, we provide two different ways of exploiting our generated samples, leading to average and concatenation design of the architectures.

LAGCN. For GCN, we only make a small change on the first graph convolution layer with

$$\boldsymbol{H}^{(1)} = \sigma\left(\widetilde{\boldsymbol{P}}\boldsymbol{X}\boldsymbol{W}_0^{(1)}\right) \Big\| \, \sigma\left(\widetilde{\boldsymbol{P}}\overline{\boldsymbol{X}}\boldsymbol{W}_1^{(1)}\right), \tag{6}$$

where $\widetilde{\boldsymbol{P}} = \widetilde{\boldsymbol{D}}^{-\frac{1}{2}}\widetilde{\boldsymbol{A}}\widetilde{\boldsymbol{D}}^{-\frac{1}{2}}$. The notation $\Big\|$ means concatenation of the matrices on the second dimension. The sub- and super- scripts in the weight matrix $\boldsymbol{W}$ denote the layers' and the parameters' ordinal numbers. In order not to change the parameter size of the GCN model, the sum of the second dimension of $\boldsymbol{W}_0^{(1)}$ and $\boldsymbol{W}_1^{(1)}$ is equal to the second dimension of $\boldsymbol{W}^{(1)}$ of GCN. For other architectures (LAGAT, LASAGE, LAGCNII) which we will discuss later, we keep the same setting on the parameter size in the first layer as LAGCN. For GraphSAGE and GCNII, they have similar architectures to GCN and we employ the same modification strategy as LAGCN for LASAGE and LAGCNII. In addition to the concatenation-style design, we can also average $\boldsymbol{X}$ and $\overline{\boldsymbol{X}}$ as input for GNNs and do not change the architecture.

LAGAT. Similarly, the first layer of LAGAT is defined as follows:

$$\begin{aligned}\boldsymbol{H}^{(1)} = &\left(\Big\|_{k=1}^{K/2} \sigma\left(\sum_{u\in\mathcal{N}_v} \alpha_{vu}^k \mathbf{W}_k^{(1)}\boldsymbol{X}_u\right)\right) \\ &\Big\| \left(\Big\|_{k=K/2+1}^{K} \sigma\left(\sum_{u\in\mathcal{N}_v} \alpha_{vu}^k \mathbf{W}_k^{(1)}\overline{\boldsymbol{X}}_u\right)\right),\end{aligned} \tag{7}$$

where α_{vu}^k is computed on $\boldsymbol{X}$ $(1 \leq k \leq K/2)$ or $\overline{\boldsymbol{X}}$ $(K/2 + 1 \leq k \leq K)$. Note that the second dimension of $\mathbf{W}_k^{(1)}$

is the same as GAT. We just replace the input of the half of the attention heads with $\overline{\boldsymbol{X}}$. And we also provide the average-style design for GAT.

3.4. Loss Functions

In this section, we explain two loss functions to train LAGNNs and the underlying motivation: *supervised loss* and *consistency loss*.

Supervised Loss. Once we have completed the pre-training of the local augmentation model, we then use the generated feature matrix $\overline{\boldsymbol{X}}$ produced by it as additional input to enhance the expressive power of GNN models. Given the training labels $\boldsymbol{K}$ and S augmented feature matrix $\overline{\boldsymbol{X}}^{(s)}$, we can write the supervised loss function for node classification tasks as follows:

$$\mathcal{L}_{sup} = -\frac{1}{S}\sum_{s=1}^{S}\sum_{i\in\boldsymbol{K}} \boldsymbol{Y}_i \log \boldsymbol{Z}_i^{(s)}, \tag{8}$$

where $\boldsymbol{Z}^{(s)} = f_{LAGNN}(\boldsymbol{A}, \boldsymbol{X}, \overline{\boldsymbol{X}}^{(s)}, \Theta)$. Note that we just provide a type of supervised loss function. For other graph learning tasks such as link prediction and graph classification, the supervised loss functions can be adjusted accordingly.

Consistency Regularization Loss. Inspired by the huge success of consistency training (Wang et al., 2020c; Feng et al., 2020; Sajjadi et al., 2016; Samuli & Timo, 2017; Berthelot et al., 2019; Verma et al., 2019) on semi-supervised learning tasks, we provide an optional loss function for specific GNNs and graph learning tasks. Intuitively, the consistency regularization encourages invariant prediction of different inputs at each training iteration (Verma et al., 2019). Specifically, the consistency regularization loss function has the following form:

$$\mathcal{L}_{con} = \frac{1}{S}\sum_{s=1}^{S}\sum_{i=1}^{N} \left\| \overline{\boldsymbol{Z}}_i' - \boldsymbol{Z}_i^{(s)} \right\|_2^2, \tag{9}$$

where $\overline{\boldsymbol{Z}}_i = \frac{1}{S}\sum_{s=1}^{S} \boldsymbol{Z}_i^{(s)}$, $\overline{\boldsymbol{Z}}_i' = \overline{\boldsymbol{Z}}_i^{\frac{1}{T}} / \sum_{c=1}^{C} \overline{\boldsymbol{Z}}_{ic}^{\frac{1}{T}}$ is the sharpening trick (Berthelot et al., 2019), and T is a hyper-parameter which adjusts the "temperature" of this categorical distribution. The sharpening trick can reduce the entropy of the predictions.

Training and Inference. The details of our training and inference process are outlined in Algorithm 1. First, we train CVAE *i.e.* our local augmentation model. And then we sample a different feature matrix generated by CVAE as additional input at each training iteration to train the GNN models. But for GRAND (Feng et al., 2020), we just sample one feature matrix during training stage since we find we can get better performance with such a sample strategy. Supervised loss functions are computed on the initial feature matrix $\boldsymbol{X}$ and the generated feature matrix $\overline{\boldsymbol{X}}$. Besides, we optionally compute the consistency regularization loss function $\mathcal{L}_{con}$ based on $f_{LAGNN}(\boldsymbol{A}, \boldsymbol{X}, \overline{\boldsymbol{X}}^{(s)}, \Theta)$ for specific tasks. Moreover, we resample another feature matrix that is different from the one to compute the training loss function to compute the validation loss function and validation accuracy at each training iteration. At the inference stage, we don't need to generate $\overline{\boldsymbol{X}}$ again since we select $\overline{\boldsymbol{X}}$ with the smallest validation loss function on citation datasets (Cora, Citeseer, Pubmed) or the highest validation accuracy on OGB datasets. The computational complexity of pre-training and GNN training is $\mathcal{O}(|\mathcal{E}|(FD + LD^2))$ and $\mathcal{O}(S|\mathcal{E}|(FD + LD^2))$ respectively, where D is the number of hidden channels and L is the number of layers. The number of pre-training epochs is usually fewer than 10, which demonstrates that pre-training introduces little computational overhead.

Algorithm 1 Local Augmentation for Graph Neural Networks

```
Input: Adjacency matrix $A$, feature matrix $X$
Output: Prediction $Z$
Pre-train the generative model $Q_\Phi$ using Eq. (2), given $A$ and $X$ as input.
while not convergence do
    for $s = 1 : S$ do
        Generate the augmented feature matrix: $\overline{X}^{(s)} \sim Q_\Phi$.
        Obtain the prediction using LAGNN $P_\Theta$: $Z^{(s)} = f_{LAGNN}(A, X, \overline{X}^{(s)}, \Theta)$
    end for
    Compute supervised classification loss $\mathcal{L}_{sup}$ via Eq. (8)
    Optionally compute the consistency regularization loss $\mathcal{L}_{con}$ via Eq. (9).
    Update the parameters $\Theta$ by gradients descending: $\Theta = \Theta - \eta\nabla_\Theta(\mathcal{L}_{sup}(+\beta\mathcal{L}_{con}))$
    Regenerate the augmented feature matrix: $\overline{X} \sim Q_\Phi$.
    Compute the validation loss function or the validation accuracy via $Z = f_{LAGNN}(A, X, \overline{X}, \Theta)$ and Eq. (8).
end while
Predict via: $Z = f_{LAGNN}(A, X, \overline{X}, \Theta)$, where we select $\overline{X}$ with the smallest validation loss function or the highest validation accuracy.
```

Table 1. Classification results on three citation networks (%)

Method	**Cora**	**Citeseer**	**Pubmed**
Chebyshev	81.2	69.8	74.4
APPNP	83.8±0.3	71.6±0.5	79.7±0.3
MixHop	81.9±0.4	71.4±0.8	80.8±0.6
Graph U-net	84.4±0.6	73.2±0.5	79.6±0.2
GSNN-M	83.9±0.5	72.2±0.5	79.1±0.3
S^{2}GC	83.5±0.02	73.6±0.09	80.2±0.02
GCN	81.5±0.5	70.3±0.7	79.0±0.5
G-GCN	83.7	71.3	80.9
DropEdge-GCN	82.8	72.3	79.6
GAUG-O-GCN	83.6±0.5	73.3±1.1	79.3±0.4
GraphSNN$_{GCN}$	83.1±1.8	72.3±1.5	79.8±1.2
GRAND-GCN	84.5±0.3	74.2±0.3	80.0±0.3
LA-GCN	**84.6±0.5**	**74.7±0.5**	**81.7±0.7**
GAT	83.0±0.7	72.5±0.7	79.0±0.3
GAUG-O-GAT	82.2±0.2	71.6±1.1	OOM
GraphSNN$_{GAT}$	83.8±1.2	73.5±1.6	79.6±1.4
GRAND-GAT	84.3±0.4	73.2±0.4	79.2±0.6
LA-GAT	**84.7±0.4**	**73.7±0.5**	**81.0±0.4**
GCNII	85.5±0.5	73.4±0.6	80.2±0.4
LA-GCNII	**85.7±0.3**	**74.1±0.5**	**80.6±0.7**
GRAND	85.4±0.4	75.4±0.4	82.7±0.6
LA-GRAND	**85.7±0.3**	**75.8±0.5**	**83.4±0.6**

4. Experiments

In this section, we evaluate the performance of our local augmentation model on various tasks including node classification, link prediction, and graph classification. All the experiments are conducted on open graph datasets.

4.1. Semi-supervised Learning

Datasets. We utilize three public citation network datasets Cora, Citeseer, and Pubmed (Sen et al., 2008) for semi-supervised node classification. All the dataset statistics can be found in Appendix D.

Baselines. We consider three popular graph neural networks: GCN (Kipf & Welling, 2017), GAT (Veličković et al., 2018), and GCNII (Chen et al., 2020b) as our implemented backbones. GCN and GAT are representatives of the state-of-the-art GNN architectures, while GCNII is a deep GNN model with a skip connection design. For each of these backbones, we employ the concatenation-style design discussed in Sec. 3.3 as our LAGNN architecture. But we keep the size of the learnable weight matrix of LAGNN the same as the corresponding GNN model, which is detailed in the Appendix D. We also combine our method with other data augmentation models - GRAND (Feng et al., 2020). To evaluate our proposed framework, we compare our model against state-of-the-art models of four categories:

- Backbone models: Chebyshev (Defferrard et al., 2016), GCN (Kipf & Welling, 2017), GAT (Veličković et al., 2018), APPNP (Klicpera et al., 2019), Graph U-net (Gao & Ji, 2019), MixHop (Abu-El-Haija et al., 2019), GCNII (Chen et al., 2020b), GSNN-M (Wang et al., 2020a), and S^{2}GC (Zhu & Koniusz, 2021)
- Feature-level augmentation models: G-GNNs (Zhu et al., 2020), and GRAND (Feng et al., 2020).
- Topology-level augmentation modes: DropEdge (Rong et al., 2020) and GAUG-O (Zhao et al., 2021).
- Subgraph GNN: GraphSNN (Wijesinghe & Wang, 2022).

The choice of baselines aims to show that existing GNNs benefit from our proposed local data augmentation, and our model outperforms other data augmentation models.

Experimental setup. We apply the standard fixed splits (Yang et al., 2016) on Cora, Citeseer, and Pubmed, with 20 nodes per class for training, 500 nodes for validation, and 1,000 nodes for testing. See more details on experimental setup and hyper-parameters in the Appendix D.

Comparison with SOTA. We report the mean node classification accuracy after 100 runs in Table 1. We reuse the metrics of the baselines already reported in the corresponding papers. The results demonstrate that the backbone models equipped with our method achieve better performance across all three datasets. Specifically, local augmentation can improve upon GCN by a margin of 3.1%, 4.4%, and 2.7% on Cora, Citeseer, and Pubmed respectively, while the improved margins of LAGAT over GAT are 1.7%, 1.2%, and 2.0% respectively. Moreover, when combined with other data augmentation methods - GRAND, we can still improve by a margin of 0.3%, 0.4%, and 0.7% respectively. Furthermore, based on the std information of the experimental results of GRAND and our LA-GRAND, we compute the p-value by t-test to verify the improvements. Except for LA-GCN v.s. GRAND-GCN on Cora (with p-value 0.046), all the p-value $\ll 0.01$ by t-test (the same test is also employed by GRAND), which shows the improvements of LA- over GRAND- are statistically significant. Compared with other data augmentation models (Zhu et al., 2020; Rong et al., 2020; Zhao et al., 2021), LA-GNN achieves the best performance on two popular backbones GCN and GAT, showing local information is indeed better than the augmentation approaches from a global perspective, such as DropEdge (Rong et al., 2020) and GAUG (Zhao et al., 2021). Both our model and GraphSNN start from the perspective

Table 2. Test performance (%) averaged over 10 runs on node property prediction. Blank denotes no statistics on the leaderboard or in the paper.

	products	proteins	arxiv
Model	Acc	ROC-AUC	Acc
MLP	61.06±0.08	72.04±0.48	55.50±0.23
CoLinkDistMLP	62.59±0.10	-	56.38±0.16
Node2vec	72.49±0.10	68.81±0.65	70.07±0.13
GraphZoom	74.06±0.26	-	71.18±0.18
GCN	75.64±0.21	72.51±0.35	71.74±0.29
+FLAG	-	71.71±0.50	72.04±0.20
+GraphSNN	-	-	**72.20±0.90**
+LA	**76.11±0.09**	**73.25±0.51**	72.08±0.14
GraphSAGE	78.70±0.36	77.68±0.20	71.49±0.27
+FLAG	79.36±0.57	76.57±0.75	72.19±0.21
+GraphSNN	-	-	71.80±0.70
+LA	**79.44±0.25**	**77.86±0.37**	**72.30±0.12**
GAT	79.45±0.59	-	73.65±0.11
+FLAG	**81.76±0.45**	-	73.71±0.13
+LA	80.46±0.54	-	**73.77±0.12**

of the subgraph. Results show that local augmentation is more effective than GraphSNN in capturing the feature information of the local neighborhood, which demonstrates that it's better to consider feature and structure information in designing subgraph-related GNNs.

4.2. Full-supervised Learning

Datasets. To demonstrate the effectiveness of our model on large graphs for full-supervised node and link classification tasks, we utilize ogbn-products, ogbn-proteins, ogbn-arxiv, and ogbl-collab datasets from Open Graph Benchmark (OGB) (Hu et al., 2020) for evaluation. All the dataset statistics can be found in the Appendix D.

Baselines. We consider four popular message-passing GNNs: GCN (Kipf & Welling, 2017), GAT (Veličković et al., 2018), and GraphSAGE (Hamilton et al., 2017) as backbones. For each of these backbones, we apply the concatenation-style or average-style design discussed in Sec. 3.3 as our LAGNN architecture, which is detailed in the Appendix D. For node classification on arxiv, proteins, and products, we compare it against MLP, Node2vec (Grover & Leskovec, 2016), GCN, GAT, GraphSAGE, FLAG (Kong et al., 2020), GraphSNN (Wijesinghe & Wang, 2022), GraphZoom (Deng et al., 2020), and CoLinkDistMLP (Luo et al., 2021). Besides, we use ogbl-collab to evaluate the performance of our model on the link prediction task, and compare it against MLP, Node2vec, GCN, GraphSAGE.

Table 3. Test performance (%) averaged over 10 runs on link prediction.

	ogbl-collab
Model	Hits@50 (%)
MLP	19.27±1.29
Node2vec	48.88±0.54
GCN	44.75±1.07
+LA	**47.49±1.40**
GraphSAGE	48.10±0.81
+LA	**49.23±0.55**

Table 4. Test performance (%) averaged over 10 runs on graph property prediction.

	ogbg-molhiv	ogbg-molpcba
Model	ROC-AUC	AP
GCN	76.06±0.97	20.20±0.24
+LA	**76.18±1.11**	**20.28±0.16**
GIN	**75.58±1.40**	**22.66±0.28**
+LA	75.20±1.74	22.38±0.24

Experimental Setup and Results. We follow the experimental setup as in OGB (Hu et al., 2020). For the detailed setup, such as the split ratio and evaluation metric, we just follow the same setting from the OGB implementation. Note that the test results of the baselines are from the official OGB leaderboard website [1] or corresponding papers. For a fair comparison, we implement our models on OGB tasks from the open-resource codes with only touching the first layer. From the OGB leaderboard, we can know that the test results are sensitive concerning model size and various tricks. So we do not change the model size of the backbones as suggested in Sec. 3.3 and do not add other tricks. The details of our LAGNN architecture can be found in the Appendix D. Results are summarized in Table 2 and Table 3. Following common practice, we report the test accuracy associated with the best validation performance. The results on node and link prediction demonstrate that our augmentation model consistently improves performance over backbones.

4.3. Inductive Learning

To evaluate the effectiveness of our model on inductive learning tasks, we take ogbg-molhiv and ogbg-molpcba datasets from OGB for evaluation. For the experimental setup, we just follow the official OGB implementation. We consider GCN and GIN (Xu et al., 2019b) as backbones and the architecture of LAGCN and LAGIN can be found in the Appendix D. The results are summarized in Table 4. The

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

Table 5. Effects of different components of our LAGCN evaluated on Pubmed dataset.

Technique	Accuracy (%)	Δ	Cumu Δ
GCN	79.0	0	0
+ Concatenation	79.3±0.4	0.3	0.3
+ Local Augmentation	81.1±0.5	1.8	2.1
+ Consistency Training	81.4±0.5	0.3	2.4
+ Sharpening Trick	81.7±0.7	0.3	2.7

Table 6. Summary of results of GCN evaluated on Citeseer on recovering study in terms of classification accuracy (%). ↓ means a decrease compared with the accuracy if features are not masked.

Mask Ratio	0.1	0.2	0.4	0.8
GCN	70.4(↑0.1)	69.2(↓1.1)	67.2(↓3.1)	61.1(↓9.2)
LAGCN	73.8(↓0.9)	74.0(↓0.7)	71.8(↓2.9)	68.7(↓6.0)

experimental results show that our model still works for GCN on inductive learning tasks. Our generative model is only trained on the training dataset. As long as the graphs on the test dataset and the training dataset have similar distributions, $i.e.$, similar subgraph structures and feature vectors, our generative model can make reasonable inferences and generate effective augmented feature vectors.

4.4. Ablation Study

To demonstrate the effectiveness of our proposed local augmentation model, we conduct experiments of LAGCN on Pubmed that compare it to several of its ablated variants. The results are shown in Table 5. "+ concatenation" means that we only apply our concatenation-style design architecture of LAGCN in Sec. 3.3 with the original feature matrix as additional concatenated input. The improvement is 0.3%, which shows that our modification of architecture does not have a lot of effect on the result. "+ local augmentation" means we use the generated feature matrix as additional concatenated input without consistency training. Although we do not use consistency training, the generated feature matrix as additional input improves the GCN's test accuracy by a margin of 1.8%. With the consistency training and sharpening trick, we can enhance the performance further. From the ablation study, it is evident that the performance gain is due to our proposed generative local augmentation framework.

4.5. Robustness to Missing Information

In this section, we conduct experiments to verify that our proposed framework is robust 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 6, 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 Citeseer, which corroborates our insight that our local augmentation can complement the contextual information of the local neighborhood.

Table 7. Summary of results of GCN and LAGCN evaluated on Pubmed on case study in terms of classification accuracy (%). Node degree is computed on $\widetilde{\boldsymbol{D}}^{-\frac{1}{2}}\widetilde{\boldsymbol{A}}\widetilde{\boldsymbol{D}}^{-\frac{1}{2}}$.

Degree	$[2, 5]$	$[6, 20]$
#Nodes	761	189
GCN	78.2	82.0
LAGCN	79.9	82.2
Δ	1.7	0.2

Table 8. The MADgap metric of LAGCN v.s. GCN of 10 runs on Cora (on different layers).

Layer	GCN	LAGCN
Layer2	0.63±0.02	0.68±0.02
Layer3	0.61±0.08	0.61±0.07
Layer4	0.55±0.06	0.64±0.05
Layer5	0.39±0.23	0.61±0.06
Layer6	0.24±0.47	0.31±0.17

4.6. Case Study

In this section, we explore the change in test accuracy of different nodes after applying our local augmentation method. Note that we only apply local augmentation without consistency training, and we set S to 1. The results are summarized in Table 7. From the results, we can draw the following conclusions: 1) The degree of most nodes on the Pubmed test set is relatively small, of which with degree fewer than 6 accounts for about 76.1%. 2) Nodes with smaller degree tend to have lower test accuracy. However, our local augmentation can enrich local information for these nodes and thus enhance their performance.

4.7. Over-smoothing Analysis

It is well known that stacking GNN layers leads to over-smoothing (Li et al., 2018). In this section, we discuss how our proposed approach prevents the over-smoothing issue in GNNs compared to existing approaches. We utilize the MADgap (Chen et al., 2020a) metric to compare our method with existing GNNs. Table 8 reports the MADgap metric of LAGCN and GCN on Cora (on different layers). We can observe that the MADgap metric of LAGCN is larger than or the same as that of GCN on different layers. Although our approach is not to address over-smoothing, our method can enrich the local neighborhood information and thus can improve the locality of node representations. Therefore, we can alleviate over-smoothing.

5. Related Work

Unsupervised Representation Learning on Graphs. In general, unsupervised representation learning methods on graphs include contrastive-based self-supervision methods (Velickovic et al., 2019; Sun et al., 2020; Hassani & Khasahmadi, 2020; You et al., 2020), graph embedding methods (García-Durán & Niepert, 2017; Hamilton et al., 2017), and random walk methods (Perozzi et al., 2014; Tang et al., 2015; Grover & Leskovec, 2016). Contrastive learning works (Hassani & Khasahmadi, 2020; You et al., 2020) employ contrastive loss functions to enforce minimizing the representation distance of the positive pairs and maximizing the distance of the negative pairs. Random walk methods get sentences by taking random walks across nodes and use NLP word embedding models to learn node representations. Our local augmentation is also an unsupervised method for learning local neighborhood information.

Graph Generative Models. Generative models (Goodfellow et al., 2014; Kingma & Welling, 2013) are powerful tools of learning data distribution through unsupervised learning. Recently, researchers have proposed several interesting generative models for graph data generation. Variational graph auto-encoder (VGAE) (Kipf & Welling, 2016) exploits the latent variables to learn interpretable representations for undirected graphs. Salha et al. (2019) make use of a simple linear model to replace the GCN encoder in VGAE and reduce the complexity of encoding schemes. Xu et al. (2019a) propose a generative GCN model to learn node representations for growing graphs. ConDgen (Yang et al., 2019) exploits the GCN encoder to handle the invariant permutation for conditional structure generation. Besides, some methods have been proposed to apply the graph generative models in various applications such as graph matching (Simonovsky & Komodakis, 2018), and 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.

Concatenation-style Design. In this work, we use concatenation to concatenate the original features and the different generated features at each training iteration to enrich the neighborhood information through local augmentation. Concatenation-style design is a general technique that many works employ such as GAT (Veličković et al., 2018) and SIGN (Rossi et al., 2020). SIGN focuses on the scalable training of GNN models on large graph and augments the feature by multi-hop information through powers of adjacency.

6. Conclusion

We propose local augmentation, a new technique that exploits the generative model to learn the conditional distribution of the central node's neighbors' features given the central node's features. We feed the generated feature matrix from a well-trained generative model to some modified backbone GNN models to enhance their 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. One future work is to 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.

Code and Data Availability

Code and data are available at `https://github.com/SongtaoLiu0823/LAGNN`.

Acknowledgements

We thank all the reviewers for their helpful comments and suggestions, the support of Tencent AI Lab and the helpful discussion from Yulai Cong. Specially, Songtao Liu is also thankful for the encouragement from Hao Yin. Liu and Wu were supported partially by an NSF grant CNS-1652790 and a seed grant from Penn State Center for Security Research and Education (CSRE).

References

Abu-El-Haija, S., Perozzi, B., Kapoor, A., Alipourfard, N., Lerman, K., Harutyunyan, H., Ver Steeg, G., and Galstyan, A. Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing. In *International Conference on Machine Learning*, 2019.

Berthelot, D., Carlini, N., Goodfellow, I., Papernot, N., Oliver, A., and Raffel, C. Mixmatch: A holistic approach to semi-supervised learning. *arXiv preprint arXiv:1905.02249*, 2019.

Bhatia, K., Dahiya, K., Jain, H., Kar, P., Mittal, A., Prabhu, Y., and Varma, M. The extreme classification repository: Multi-label datasets and code, 2016. URL http://manikvarma.org/downloads/XC/XMLRepository.html.

Chen, D., Lin, Y., Li, W., Li, P., Zhou, J., and Sun, X. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020a.

Chen, M., Wei, Z., Huang, Z., Ding, B., and Li, Y. Simple and deep graph convolutional networks. In *International Conference on Machine Learning*, 2020b.

Chiang, W.-L., Liu, X., Si, S., Li, Y., Bengio, S., and Hsieh, C.-J. 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.

Cubuk, E. D., Zoph, B., Mane, D., Vasudevan, V., and Le, Q. V. Autoaugment: Learning augmentation strategies from data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019.

Dai, H., Li, C., Coley, C. W., Dai, B., and Song, L. Retrosynthesis prediction with conditional graph logic network. In *Advances in Neural Information Processing Systems*, 2019.

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

Deng, C., Zhao, Z., Wang, Y., Zhang, Z., and Feng, Z. Graphzoom: A multi-level spectral approach for accurate and scalable graph embedding. In *International Conference on Learning Representations*, 2020.

Deng, Z., Dong, Y., and Zhu, J. Batch virtual adversarial training for graph convolutional networks. *arXiv preprint arXiv:1902.09192*, 2019.

Dong, H., Fu, Y., Hwang, S. J., Sigal, L., and Xue, X. Learning the compositional domains for generalized zero-shot learning. *Computer Vision and Image Understanding*, pp. 103454, 2022.

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

Fang, C., Dong, H., and Zhang, T. Mathematical models of overparameterized neural networks. *Proceedings of the IEEE*, 109(5):683–703, 2021.

Feng, F., He, X., Tang, J., and Chua, T.-S. Graph adversarial training: Dynamically regularizing based on graph structure. *IEEE Transactions on Knowledge and Data Engineering*, 2019.

Feng, W., Zhang, J., Dong, Y., Han, Y., Luan, H., Xu, Q., Yang, Q., Kharlamov, E., and Tang, J. Graph random neural network for semi-supervised learning on graphs. In *Advances in Neural Information Processing Systems*, 2020.

Fey, M. and Lenssen, J. E. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.

Gao, H. and Ji, S. Graph u-nets. In *International Conference on Machine Learning*, 2019.

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

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. Neural message passing for quantum chemistry. In *International Conference on Machine Learning*, 2017.

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative adversarial networks. In *Advances in Neural Information Processing Systems*, 2014.

Grover, A. and Leskovec, J. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016.

Guo, S., Lin, Y., Feng, N., Song, C., and Wan, H. Attention based spatial-temporal graph convolutional networks for traffic flow forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2019.

Hamilton, W. L., Ying, R., and Leskovec, J. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, 2017.

Hassani, K. and Khasahmadi, A. H. Contrastive multi-view representation learning on graphs. In *International Conference on Machine Learning*, 2020.

Hausman, D. M. and Woodward, J. Independence, invariance and the causal markov condition. *The British journal for the philosophy of science*, 1999.

Houlsby, N., Huszár, F., Ghahramani, Z., and Lengyel, M. Bayesian active learning for classification and preference learning. *arXiv preprint arXiv:1112.5745*, 2011.

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

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

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

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

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

Kong, K., Li, G., Ding, M., Wu, Z., Zhu, C., Ghanem, B., Taylor, G., and Goldstein, T. Flag: Adversarial data augmentation for graph neural networks. *arXiv preprint arXiv:2010.09891*, 2020.

Li, G., Müller, M., Ghanem, B., and Koltun, V. Training graph neural networks with 1000 layers. In *International Conference on Machine Learning*, 2021.

Li, Q., Han, Z., and Wu, X.-M. Deeper insights into graph convolutional networks for semi-supervised learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018.

Liu, Q., Allamanis, M., Brockschmidt, M., and Gaunt, A. L. Constrained graph variational autoencoders for molecule design. In *Advances in Neural Information Processing Systems*, 2018.

Luo, Y., Chen, A., Yan, K., and Tian, L. Distilling self-knowledge from contrastive links to classify graph nodes without passing messages. *arXiv preprint arXiv:2106.08541*, 2021.

Nielsen, C. and Okoniewski, M. M. Gan data augmentation through active learning inspired sample acquisition. In *CVPR Workshops*, 2019.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, 2019.

Perozzi, B., Al-Rfou, R., and Skiena, S. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2014.

Rong, Y., Huang, W., Xu, T., and Huang, J. Dropedge: Towards deep graph convolutional networks on node classification. In *International Conference on Learning Representation*, 2020.

Rossi, E., Frasca, F., Chamberlain, B., Eynard, D., Bronstein, M., and Monti, F. Sign: Scalable inception graph neural networks. *arXiv preprint arXiv:2004.11198*, 2020.

Şahin, G. G. and Steedman, M. Data augmentation via dependency tree morphing for low-resource languages. *arXiv preprint arXiv:1903.09460*, 2019.

Sajjadi, M., Javanmardi, M., and Tasdizen, T. Regularization with stochastic transformations and perturbations for deep semi-supervised learning. *Advances in Neural Information Processing Systems*, 2016.

Salha, G., Hennequin, R., and Vazirgiannis, M. Keep it simple: Graph autoencoders without graph convolutional networks. *arXiv preprint arXiv:1910.00942*, 2019.

Samanta, B., De, A., Ganguly, N., and Gomez-Rodriguez, M. Designing random graph models using variational autoencoders with applications to chemical design. *arXiv preprint arXiv:1802.05283*, 2018.

Samuli, L. and Timo, A. Temporal ensembling for semi-supervised learning. In *International Conference on Learning Representations*, 2017.

Sen, P., Namata, G., Bilgic, M., Getoor, L., Galligher, B., and Eliassi-Rad, T. Collective classification in network data. *AI magazine*, 2008.

Shi, C., Xu, M., Guo, H., Zhang, M., and Tang, J. A graph to graphs framework for retrosynthesis prediction. In *International Conference on Machine Learning*, 2020.

Shorten, C. and Khoshgoftaar, T. M. A survey on image data augmentation for deep learning. *Journal of Big Data*, 2019.

Simonovsky, M. and Komodakis, N. Graphvae: Towards generation of small graphs using variational autoencoders. In *International Conference on Artificial Neural Networks*, 2018.

Sohn, K., Lee, H., and Yan, X. Learning structured output representation using deep conditional generative models. In *Advances in Neural Information Processing Systems*, 2015.

Sun, F.-Y., Hoffmann, J., Verma, V., and Tang, J. Infograph: Unsupervised and semi-supervised graph-level representation learning via mutual information maximization. In *International Conference on Learning Representations*, 2020.

Szklarczyk, D., Gable, A. L., Lyon, D., Junge, A., Wyder, S., Huerta-Cepas, J., Simonovic, M., Doncheva, N. T., Morris, J. H., Bork, P., et al. String v11: protein–protein association networks with increased coverage, supporting functional discovery in genome-wide experimental datasets. *Nucleic acids research*, 47(D1):D607–D613, 2019.

Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J., and Mei, Q. Line: Large-scale information network embedding. In *Proceedings of the 24th international conference on world wide web*, 2015.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. Graph attention networks. In *International Conference on Learning Representations*, 2018.

Velickovic, P., Fedus, W., Hamilton, W. L., Liò, P., Bengio, Y., and Hjelm, R. D. Deep graph infomax. In *International Conference on Learning Representations*, 2019.

Verma, V., Kawaguchi, K., Lamb, A., Kannala, J., Bengio, Y., and Lopez-Paz, D. Interpolation consistency training for semi-supervised learning. *arXiv preprint arXiv:1903.03825*, 2019.

Wang, H., Zhou, C., Chen, X., Wu, J., Pan, S., and Wang, J. Graph stochastic neural networks for semi-supervised learning. In *Advances in Neural Information Processing Systems*, 2020a.

Wang, K., Shen, Z., Huang, C., Wu, C.-H., Dong, Y., and Kanakia, A. Microsoft academic graph: When experts are not enough. *Quantitative Science Studies*, 1(1):396–413, 2020b.

Wang, M., Zheng, D., Ye, Z., Gan, Q., Li, M., Song, X., Zhou, J., Ma, C., Yu, L., Gai, Y., Xiao, T., He, T., Karypis, G., Li, J., and Zhang, Z. Deep graph library: A graph-centric, highly-performant package for graph neural networks. *arXiv preprint arXiv:1909.01315*, 2019.

Wang, Y., Wang, W., Liang, Y., Cai, Y., Liu, J., and Hooi, B. Nodeaug: Semi-supervised node classification with data augmentation. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020c.

Wijesinghe, A. and Wang, Q. A new perspective on "how graph neural networks go beyond weisfeiler-lehman?". In *International Conference on Learning Representations*, 2022.

Wu, Z., Ramsundar, B., Feinberg, E. N., Gomes, J., Geniesse, C., Pappu, A. S., Leswing, K., and Pande, V. Moleculenet: a benchmark for molecular machine learning. *Chemical science*, 9(2):513–530, 2018.

Xia, M., Kong, X., Anastasopoulos, A., and Neubig, G. Generalized data augmentation for low-resource translation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019.

Xu, D., Ruan, C., Motwani, K., Korpeoglu, E., Kumar, S., and Achan, K. Generative graph convolutional network for growing graphs. In *International Conference on Acoustics, Speech and Signal Processing*, 2019a.

Xu, K., Li, C., Tian, Y., Sonobe, T., Kawarabayashi, K.-i., and Jegelka, S. Representation learning on graphs with jumping knowledge networks. In *International Conference on Machine Learning*, 2018.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019b.

Yang, C., Zhuang, P., Shi, W., Luu, A., and Li, P. Conditional structure generation through graph variational generative adversarial nets. In *Advances in Neural Information Processing Systems*, 2019.

Yang, Z., Cohen, W., and Salakhudinov, R. Revisiting semi-supervised learning with graph embeddings. In *International Conference on Machine Learning*, 2016.

Ying, R., He, R., Chen, K., Eksombatchai, P., Hamilton, W. L., and Leskovec, J. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018.

You, Y., Chen, T., Sui, Y., Chen, T., Wang, Z., and Shen, Y. Graph contrastive learning with augmentations. *Advances in Neural Information Processing Systems*, 2020.

Zeng, H., Zhang, M., Xia, Y., Srivastava, A., Malevich, A., Kannan, R., Prasanna, V., Jin, L., and Chen, R. Decoupling the depth and scope of graph neural networks. In *Advances in Neural Information Processing Systems*, 2021.

Zhang, M. and Li, P. Nested graph neural networks. In *Advances in Neural Information Processing Systems*, 2021.

Zhao, A., Balakrishnan, G., Durand, F., Guttag, J. V., and Dalca, A. V. Data augmentation using learned transformations for one-shot medical image segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019.

Zhao, T., Liu, Y., Neves, L., Woodford, O., Jiang, M., and Shah, N. Data augmentation for graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021.

Zhu, D., Dai, X.-Y., and Chen, J. Pre-train and learn: Preserve global information for graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020.

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

A. Proofs

We give more details of the derivation of the generator ELBO (Eq. (2)) as follows:

$$\begin{aligned}
\log p_\theta(\boldsymbol{X}_u|\boldsymbol{X}_v) &= \int q_\phi(\mathbf{z}|\boldsymbol{X}_u,\boldsymbol{X}_v)\log p_\theta(\boldsymbol{X}_v|\boldsymbol{X}_v)\mathrm{d}\mathbf{z} \\
&= \int q_\phi(\mathbf{z}|\boldsymbol{X}_u,\boldsymbol{X}_v)\log \frac{p_\theta(\boldsymbol{X}_u,\boldsymbol{X}_v)}{p_\theta(\boldsymbol{X}_v)}\mathrm{d}\mathbf{z} \\
&= \int q_\phi(\mathbf{z}|\boldsymbol{X}_u,\boldsymbol{X}_v)\log \frac{p_\theta(\boldsymbol{X}_u,\boldsymbol{X}_v)p_\theta(\boldsymbol{X}_v,\boldsymbol{X}_v,\mathbf{z})}{p_\theta(\boldsymbol{X}_v)p_\theta(\boldsymbol{X}_v,\boldsymbol{X}_v,\mathbf{z})}\mathrm{d}\mathbf{z} \\
&= \int q_\phi(\mathbf{z}|\boldsymbol{X}_v,\boldsymbol{X}_v)\log \frac{p_\theta(\boldsymbol{X}_u,\boldsymbol{X}_v,\mathbf{z})}{p_\theta(\boldsymbol{X}_v)}\frac{1}{\frac{p_\theta(\boldsymbol{X}_u,\boldsymbol{X}_v,\mathbf{z})}{p_\theta(\boldsymbol{X}_u,\boldsymbol{X}_v)}}\mathrm{d}\mathbf{z} \\
&= \int q_\phi(\mathbf{z}|\boldsymbol{X}_u,\boldsymbol{X}_v)\log \frac{p_\theta(\boldsymbol{X}_u,\mathbf{z}|\boldsymbol{X}_v)}{p_\theta(\mathbf{z}|\boldsymbol{X}_u,\boldsymbol{X}_v)}\mathrm{d}\mathbf{z} \\
&= \int q_\phi(\mathbf{z}|\boldsymbol{X}_u,\boldsymbol{X}_v)\log \frac{p_\theta(\boldsymbol{X}_u,\mathbf{z}|\boldsymbol{X}_v)}{p_\theta(\mathbf{z}|\boldsymbol{X}_u,\boldsymbol{X}_v)}\frac{q_\phi(\mathbf{z}|\boldsymbol{X}_u,\boldsymbol{X}_v)}{q_\phi(\mathbf{z}|\boldsymbol{X}_u,\boldsymbol{X}_v)}\mathrm{d}\mathbf{z} \\
&= \int q_\phi(\mathbf{z}|\boldsymbol{X}_u,\boldsymbol{X}_v)\left(\log \frac{p_\theta(\boldsymbol{X}_u,\mathbf{z}|\boldsymbol{X}_v)}{q_\phi(\mathbf{z}|\boldsymbol{X}_u,\boldsymbol{X}_v)} + \log\frac{q_\phi(\mathbf{z}|\boldsymbol{X}_u,\boldsymbol{X}_v)}{p_\theta(\mathbf{z}|\boldsymbol{X}_u,\boldsymbol{X}_v)}\right)\mathrm{d}\mathbf{z} \\
&= \int q_\phi(\mathbf{z}|\boldsymbol{X}_u,\boldsymbol{X}_v)\log \frac{p_\theta(\boldsymbol{X}_u,\mathbf{z}|\boldsymbol{X}_v)}{q_\phi(\mathbf{z}|\boldsymbol{X}_u,\boldsymbol{X}_v)}\mathrm{d}\mathbf{z} + KL(q_\phi(\mathbf{z}|\boldsymbol{X}_u,\boldsymbol{X}_v)||p_\theta(\mathbf{z}|\boldsymbol{X}_u,\boldsymbol{X}_v)) \\
&\geq \int q_\phi(\mathbf{z}|\boldsymbol{X}_u,\boldsymbol{X}_v)\log \frac{p_\theta(\boldsymbol{X}_u,\mathbf{z}|\boldsymbol{X}_v)}{q_\phi(\mathbf{z}|\boldsymbol{X}_u,\boldsymbol{X}_v)}\mathrm{d}\mathbf{z}
\end{aligned}$$

$$\begin{aligned}
L_{ELBO} &= \int q_\phi(\mathbf{z}|\boldsymbol{X}_u,\boldsymbol{X}_v)\log \frac{p_\theta(\boldsymbol{X}_u,\mathbf{z}|\boldsymbol{X}_v)}{q_\phi(\mathbf{z}|\boldsymbol{X}_u,\boldsymbol{X}_v)}\mathrm{d}\mathbf{z} \\
&= \int q_\phi(\mathbf{z}|\boldsymbol{X}_u,\boldsymbol{X}_v)\log \frac{p_\theta(\boldsymbol{X}_u,\boldsymbol{X}_v,\mathbf{z})}{q_\phi(\mathbf{z}|\boldsymbol{X}_u,\boldsymbol{X}_v)p_\theta(\boldsymbol{X}_v)}\mathrm{d}\mathbf{z} \\
&= \int q_\phi(\mathbf{z}|\boldsymbol{X}_u,\boldsymbol{X}_v)\log \frac{p_\theta(\boldsymbol{X}_u|\boldsymbol{X}_v,\mathbf{z})p_\theta(\boldsymbol{X}_v,\mathbf{z})}{q_\phi(\mathbf{z}|\boldsymbol{X}_u,\boldsymbol{X}_v)p_\theta(\boldsymbol{X}_v)}\mathrm{d}\mathbf{z} \\
&= \int q_\phi(\mathbf{z}|\boldsymbol{X}_u,\boldsymbol{X}_v)\log \frac{p_\theta(\boldsymbol{X}_u|\boldsymbol{X}_v,\mathbf{z})p_\theta(\mathbf{z}|\boldsymbol{X}_v)}{q_\phi(\mathbf{z}|\boldsymbol{X}_u,\boldsymbol{X}_v)}\mathrm{d}\mathbf{z} \\
&= \int q_\phi(\mathbf{z}|\boldsymbol{X}_u,\boldsymbol{X}_v)\log \frac{p_\theta(\mathbf{z}|\boldsymbol{X}_v)}{q_\phi(\mathbf{z}|\boldsymbol{X}_u,\boldsymbol{X}_v)}\mathrm{d}\mathbf{z} + \int q_\phi(\mathbf{z}|\boldsymbol{X}_u,\boldsymbol{X}_v)\log p_\theta(\boldsymbol{X}_u|\boldsymbol{X}_v,\mathbf{z})\mathrm{d}\mathbf{z} \\
&= -KL(q_\phi(\mathbf{z}|\boldsymbol{X}_u,\boldsymbol{X}_v)||p_\theta(\mathbf{z}|\boldsymbol{X}_v)) + \int q_\phi(\mathbf{z}|\boldsymbol{X}_u,\boldsymbol{X}_v)\log p_\theta(\boldsymbol{X}_u|\boldsymbol{X}_v,\mathbf{z})\mathrm{d}\mathbf{z} \\
&= -KL(q_\phi(\mathbf{z}|\boldsymbol{X}_u,\boldsymbol{X}_v)\|p_\theta(\mathbf{z}|\boldsymbol{X}_v)) + \frac{1}{L}\sum_{l=1}^{L}\log p_\theta(\boldsymbol{X}_u|\boldsymbol{X}_v,\mathbf{z}^{(l)})
\end{aligned}$$

B. Pretraining details

B.1. Framework

We build CVAE based on MLP. The encode and decoder are two-layer MLP where each layer has 256 hidden units for all the graph datasets. For a node v and its neighbors $\mathcal{N}_v$, we extract neighboring-paris ($\boldsymbol{X}_v$, $\boldsymbol{X}_u$) as input for CVAE during the training stage, where $u \in \mathcal{N}_v$. In the inference stage, we extract a latent variable $\boldsymbol{z}$ from $\mathcal{N}(\mathbf{0}, \boldsymbol{I})$ and the center node v's feature vector $\boldsymbol{X}_v$ as input for the decoder of CVAE. Thus, we can obtain the generated feature vector $\overline{\boldsymbol{X}}_v$. The detail of CVAE is illustrated in Fig. 2.

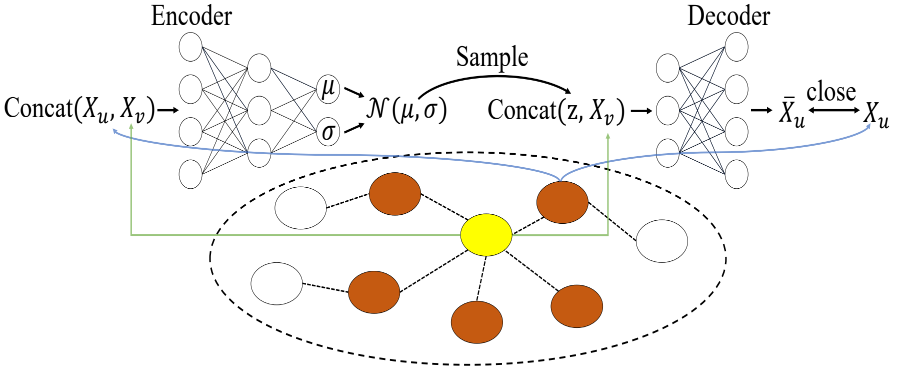


Figure 2. A schematic depiction of CVAE. The yellow and brown circles on the graph correspond to the center node and its neighbors respectively. We extract their feature vectors as input for CVAE.

Algorithm 2 The framework to obtain the Generator Q_Φ with active learning trick on Cora, Citeseer, and Pubmed

```
Input: Adjacency matrix $\boldsymbol{A}$, feature matrix $\boldsymbol{X}$
Output: Generator $Q_\Phi$
Initialize $U$=-inf, $Q_\Phi$, and $Q'_\Phi$
for $i = 1$ to the number of pre-training epochs do
    Update the parameters of generator $Q_\Phi$
    Generate feature matrix $\overline{\boldsymbol{X}}$ using $Q_\Phi$
    Compute $U(\overline{\boldsymbol{X}})$ using Eq.(10).
    if $U(\overline{\boldsymbol{X}}) > U$ then
        $U = U(\overline{\boldsymbol{X}})$
        $Q'_\Phi = Q_\Phi$
    end if
end for
$Q_\Phi = Q'_\Phi$
Return: Generator $Q_\Phi$
```

B.2. Active Learning Trick

We introduce a trick for the pre-training of CVAE on Cora, Citerseer, and Pubmed. Since the generator may generate some features from the long tail of the distribution. This critical question makes the generation inefficient. Inspired by Nielsen & Okoniewski (2019), we introduce active learning to address this issue. During active learning, 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) with the approximation from the Monte Carlo (MC) dropout samples, which is defined as follows:

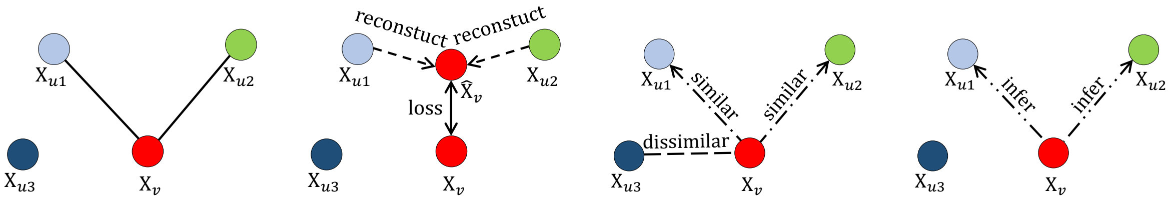
$$U(\overline{\boldsymbol{X}}) \approx H\left[\frac{1}{N}\sum_{n=1}^{N} P\left(\boldsymbol{Y}|\overline{\boldsymbol{X}}, \boldsymbol{\omega}_n\right)\right] - \frac{1}{N}\sum_{n=1}^{N} H\left[P\left(\boldsymbol{Y}|\overline{\boldsymbol{X}}, \boldsymbol{\omega}_n\right)\right], \tag{10}$$

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 BALD 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 CVAE training procedure is summarized in Algorithm 2.

C. Connection to Existing works

C.1. Connection to EP-B and GraphSAGE

We discuss how our proposed model distinguishes from the classical graph embedding models. EP-B (García-Durán & Niepert, 2017) and GraphSAGE (Hamilton et al., 2017) rely on reconstruction loss function between the center node and its neighbors' embeddings. EP-B aims to minimize their defined reconstruction error and make the attribute vector representation reconstructed by the neighbor node from message passing mechanism is as close as possible to the original attribute vector.



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

Figure 3. (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 neighbors.

Table 9. Data statistics

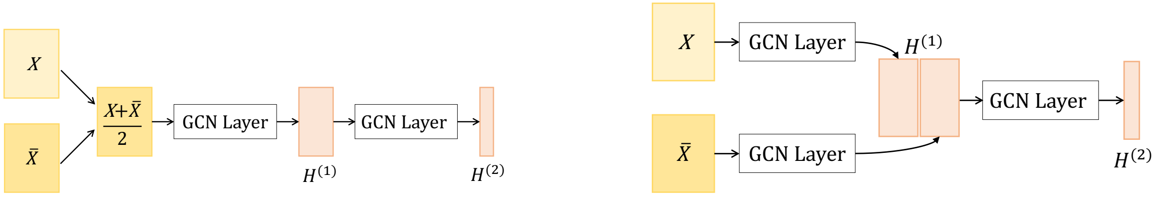
Category	Name	#Graphs	Average #Nodes	Average #Edges	#Features	#Classes	Split Ratio	Task Type	Metric
Node `citation-`	`cora`	1	2,708	5,429	1,433	7	8.5/30.5/61	Multi-class class.	Accuracy
	`citeseer`	1	3,327	4,732	3,703	6	7.4/30.9/61.7	Multi-class class.	Accuracy
	`pubmed`	1	19,717	44,338	500	3	3.8/32.1/64.1	Multi-class class.	Accuracy
Node `ogbn-`	`products`	1	2,449,029	61,859,140	100	47	8/2/90	Multi-class class.	Accuracy
	`proteins`	1	132,534	39,561,252	8	2	65/16/19	Binary class.	ROC-AUC
	`arxiv`	1	169,343	1,166,243	128	40	54/18/28	Multi-class class.	Accuracy
Link `ogbl-`	`collab`	1	235,868	1,285,465	128	-	92/4/4	Link prediction	Hits@50
Graph `ogbg-`	`molhiv`	41,127	25.5	27.5	9	2	80/10/10	Binary class.	ROC-AUC
	`molpcba`	437,929	26.0	28.1	9	2	80/10/10	Binary class.	AP

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 their proposed objective function. The two graph embedding models build upon the assumption that nearby nodes share similar attributes. In contrast, our model does not rely on such assumption and generates more feature vectors of the connected neighbors from a well-learned distribution. A comparison between the reconstruction-based representation learning on graphs and our proposed framework is illustrated in Figure 3. And our local augmentation method is the third paradigm to exploit neighbors in a generative way.

D. Reproducibility

D.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 six large graph datasets:ogbn-products, ogbn-proteins, ogbn-arxiv, ogbl-collab, ogbg-molhiv, and ogbg-molpcba from OGB (Hu et al., 2020) for evaluation. The goal of OGB is to support and catalyze research in graph ML. Specifically, ogbn-products is an Amazon products co-purchasing network (Bhatia et al., 2016) originally developed by Chiang et al. (2019), where node features are dimensionality-reduced bag-of-words of the product descriptions. The ogbn-arxiv dataset is extracted from the Microsoft Academic Graph (MAG) (Wang et al., 2020b), where each node is an ARXIV paper and each directed edge indicates that one paper cites another one and its node representation is a 128-dimensional feature vector obtained by averaging the embeddings of words in its title and abstract. ognb-proteins dataset is a protein-protein association network (Szklarczyk et al., 2019), where nodes represent proteins, and edges indicate different types of biologically meaningful associations between proteins. We use edge embedding as node embedding as suggested in PyG (Fey & Lenssen, 2019) implementation. ogbl-collab is an author collaboration network (Wang et al., 2020b), where each node represents an author, edges indicate the collaboration between authors, and the node representation is a 128-dimensional feature vector, obtained by averaging the word embeddings of



(a) Average-style LAGCN

(b) Concatenation-style LAGCN

Figure 4. Average-style and Concatenation-style LAGCN architectures. The difference between the two architectures is that the concatenation-style LAGCN has an additional convolutional layer for $\overline{\boldsymbol{X}}$ and it uses a concatenation operation to mix the hidden representations.

papers that are published by the authors. The ogbg-molhiv and ogbg-molpcba datasets are two molecular property prediction datasets adopted from Wu et al. (2018), where each graph represents a molecule. ogbn-arxiv, ogbn-proteins, ogbn-products are for node classification, ogbl-collab is for link prediction, and ogbg-molhiv and ogbg-molpcba are for graph prediction. All the dataset statistics are summarized in Table 9.

D.2. Implementation Details

We use Pytorch (Paszke et al., 2019), PyG (Fey & Lenssen, 2019), and DGL (Wang et al., 2019) to implement LA-GNNs. The codes of LA-GCN, LA-GAT, LA-GCNII, LA-GRAND on Cora, Citeseer, and Pubmed are implemented referring to Pytorch implementation of GCN[2] (Kipf & Welling, 2017), PyG implementation of GAT[3] (Veličković et al., 2018), Pytorch implementation of GCNII[4] (Chen et al., 2020b), and Pytorch implementation of GRAND[5] (Feng et al., 2020) respectively. The codes of LA-GCN, LA-GRAND, and LA-GIN on OGB datasets are implemented referring to official OGB implementation[6]. The codes of LA-GAT on ogbn-products and ogbn-arxiv are implemented referring to official OGB implementation[7] and DGL implementation[8]. All the experiments in this work are conducted on a single NVIDIA Tesla V100 with 32GB memory size. The software that we use for experiments are Python 3.6.8, pytorch 1.9.0, pytorch-cluster 1.5.9, pytorch-scatter 2.0.9, pytorch-sparse 0.6.12, pyg 2.0.3, ogb 1.3.2, dgl 0.7.2, numpy 1.19.2, torchvision 0.10.0, CUDA 10.2.89, and CUDNN 7.6.5.

D.3. Hyperparameter Details

As we have discussed in Sec. 3.3, we provide two design for our LAGNNs - average and concatenation. For concatenation-style design, LA-GNNs introduce an additional aggregation over our generated feature matrix $\overline{\boldsymbol{X}}$ before concatenation. The details of two style design architectures of LA-GCN can be found in Figure 4. Specifically, we use 3 additional generated feature matrix in the first layer for the Pubmed dataset. More details about hyparatemeters can be found in Table 10 and 11.

[2] https://github.com/tkipf/pygcn

[3] https://github.com/pyg-team/pytorch_geometric/blob/master/examples/gat.py

[4] https://github.com/chennnM/GCNII

[5] https://github.com/THUDM/GRAND

[6] https://github.com/snap-stanford/ogb/blob/master/examples

[7] https://github.com/pyg-team/pytorch_geometric/blob/master/examples/ogbn_products_gat.py

[8] https://github.com/dmlc/dgl/blob/master/examples/pytorch/ogb/ogbn-arxiv/gat.py

Table 10. The hyper-parameters for each backbone on three citation datasets.

Dataset	Backbone	Architecture	Additional aggregation	Hyper-parameters
Cora	LAGCN	Concatenation	1	epochs: 2000, lr: 0.01, weight-decay: 5e-4, hidden: 8, layers: 2, dropout: 0.5, S: 4, consistency training: True, β: 1.0 T: 0.5
	LAGAT	Concatenation	1	epochs: 1000, lr: 0.01, weight-decay: 5e-4, hidden: 8, layers: 2, heads: [4, 1], dropout: 0.6, alpha: 0.2, S: 4, consistency training: True, β: 1.0 T: 0.5
	LAGCNII	Concatenation	1	lr: 0.01, L_{2_c}: 0.01, L_{2_d}: 5e-4, layers: 64, hidden: 32, α_l: 0.1, λ: 0.5, dropout: 0.6, S: 4, consistency training: False, early stopping patience: 200
	LAGRAND	-	-	lr: 0.01, weight-decay: 5e-4, input dropout rate: 0.5, hidden dropout rate: 0.5, dropNode probability: 0.5, propagation step: 8, hidden: 32, S: 4, consistency training: True, β: 1.0, T: 0.5, early stopping patience: 200, batch normalization: False
Citeseer	LAGCN	Concatenation	1	epochs: 2000, lr: 0.01, weight-decay: 5e-4, hidden: 8, layers: 2, dropout: 0.5, S: 4, consistency training: True, β: 1.0 T: 0.5
	LAGAT	Concatenation	1	epochs: 1000, lr: 0.01, weight-decay: 5e-4, hidden: 8, layers: 2, heads: [4, 1], dropout: 0.6, alpha: 0.2, S: 4, consistency training: True, β: 1.0 T: 0.5
	LAGCNII	Concatenation	1	lr: 0.01, L_{2_c}: 0.01, L_{2_d}: 5e-4, layers: 32, hidden: 128, α_l: 0.1, λ: 0.6, dropout: 0.7, S: 4, consistency training: True, β: 1.0, T: 0.5, early stopping patience: 200
	LAGRAND	-	-	lr: 0.01, weight-decay: 5e-4, input dropout rate: 0.2, hidden dropout rate: 0.1, dropNode probability: 0, propagation step: 2, hidden: 32, S: 4, consistency training: True, β: 0.7, T: 0.2, early stopping patience: 200, batch normalization: False
Pubmed	LAGCN	Concatenation	3	epochs: 300, lr: 0.02, weight-decay: 5e-4, hidden: 4, layers: 2, dropout: 0.5, S: 4, consistency training: True, β: 1.0 T: 0.5
	LAGAT	Concatenation	3	epochs: 1000, lr: 0.01, weight-decay: 5e-4, hidden: 8, layers: 2, heads: [2, 1], dropout: 0.6, alpha: 0.2, S: 4, consistency training: True, β: 1.0 T: 0.5
	LAGCNII	Concatenation	3	lr: 0.01, L_{2_c}: 5e-4, L_{2_d}: 5e-4, layers: 16, hidden: 64, α_l: 0.1, λ: 0.4, dropout: 0.5, S: 4, consistency training: True, β: 1.0, T: 0.5, early stopping patience: 200
	LAGRAND	-	-	lr: 0.2, weight-decay: 5e-4, input dropout rate: 0.7, hidden dropout rate: 0.8, dropNode probability: 0.7, propagation step: 5, hidden: 32, S: 4, consistency training: True, β: 1.2, T: 0.2, early stopping patience: 200, batch normalization: True

Table 11. The hyper-parameters for each backbone on OGB datasets.

Dataset	Backbone	Architecture	Additional aggregation	Hyper-parameters
ogbn-arxiv	LAGCN	Concatenation	1	epochs: 500, lr: 0.01, hidden: 128, layers: 3, dropout: 0.5, S: 2, consistency training: True, β: 1.0 T: 0.5
	LAGAT	Concatenation	1	epochs: 2000, lr: 0.01, n-hidden: 125, n-layers: 3, n-heads: 3, dropout: 0.75, mask-rate: 0.5, no-attn-dst: True, use-norm: True, use-labels: True, input-drop: 0, attn-drop: 0, edge-drop: 0, wd: 0, S: 2, consistency training: False
	LASAGE	Concatenation	1	epochs: 500, lr: 0.01, hidden: 128, layers: 3, dropout: 0.5, S: 4, consistency training: True, β: 1.0 T: 0.5
ogbn-proteins	LAGCN	Concatenation	1	epochs: 1000, lr: 0.01, hidden: 128, layers: 3, dropout: 0.0, S: 2, consistency training: False
	LASAGE	Concatenation	1	epochs: 1000, lr: 0.01, hidden: 128, layers: 3, dropout: 0.0, S: 2, consistency training: False
ogbn-products	LAGCN	Average	-	epochs: 300, lr: 0.01, hidden: 256, layers: 3, dropout: 0.5, S: 1, consistency training: False
	LAGAT	Average	-	epochs: 100, lr: 0.001, hidden: 128, layers: 3, heads: 4, S: 2, consistency training: False
	LASAGE	Average	-	epochs: 20, lr: 0.001, hidden: 128, layers: 3, dropout: 0.5, heads: 4, S: 2, consistency training: True, β: 1.0 T: 0.5
ogbl-collab	LAGCN	Concatenation	1	epochs: 400, lr: 0.001, hidden: 128, layers: 3, dropout: 0.0, S: 1, consistency training: False
	LASAGE	Concatenation	1	epochs: 400, lr: 0.001, hidden: 128, layers: 3, dropout: 0.0, S: 1, consistency training: False
ogbg-molhiv	LAGCN	Average	-	epochs: 100, lr: 0.001, hidden: 300, layers: 5, dropout: 0.0, S: 1, consistency training: False, batch size: 32
	LAGIN	Average	-	epochs: 100, lr: 0.001, hidden: 300, layers: 5, dropout: 0.0, S: 1, consistency training: False, batch size: 32
ogbg-molpcba	LAGCN	Average	-	epochs: 100, lr: 0.001, hidden: 300, layers: 5, dropout: 0.0, S: 1, consistency training: False, batch size: 32
	LAGIN	Average	-	epochs: 100, lr: 0.001, hidden: 300, layers: 5, dropout: 0.0, S: 1, consistency training: False, batch size: 32