NodeAug: Semi-Supervised Node Classification with Data Augmentation

Yiwei Wang
National University of Singapore
Singapore
wangyw_seu@foxmail.com

Wei Wang
National University of Singapore
Singapore
wangwei@comp.nus.edu.sg

Yuxuan Liang
National University of Singapore
Singapore
yuxliang@outlook.com

Yujun Cai
Nanyang Technological University
Singapore
yujun001@e.ntu.edu.sg

Juncheng Liu
National University of Singapore
Singapore
juncheng.liu@u.nus.edu

Bryan Hooi
National University of Singapore
Singapore
bhooi@comp.nus.edu.sg

ABSTRACT

By using Data Augmentation (DA), we present a new method to enhance Graph Convolutional Networks (GCNs), that are the state-of-the-art models for semi-supervised node classification. DA for graph data remains under-explored. Due to the connections built by edges, DA for different nodes influence each other and lead to undesired results, such as uncontrollable DA magnitudes and changes of ground-truth labels. To address this issue, we present the **NodeAug** (Node-Parallel Augmentation) scheme, that creates a 'parallel universe' for each node to conduct DA, to block the undesired effects from other nodes. NodeAug regularizes the model prediction of every node (including unlabeled) to be invariant with respect to changes induced by Data Augmentation (DA), so as to improve the effectiveness. To augment the input features from different aspects, we propose three DA strategies by modifying both node attributes and the graph structure. In addition, we introduce the subgraph mini-batch training for the efficient implementation of NodeAug. The approach takes the subgraph corresponding to the receptive fields of a batch of nodes as the input per iteration, rather than the whole graph that the prior full-batch training takes. Empirically, NodeAug yields significant gains for strong GCN models on the Cora, Citeseer, Pubmed, and two co-authorship networks, with a more efficient training process thanks to the proposed subgraph mini-batch training approach.

CCS CONCEPTS

• **Computing methodologies → Semi-supervised learning settings**; **Regularization**; *Neural networks.*

KEYWORDS

graph convolutional networks, data augmentation, graph mining, semi-supervised learning

ACM Reference Format:
Yiwei Wang, Wei Wang, Yuxuan Liang, Yujun Cai, Juncheng Liu, and Bryan Hooi. 2020. NodeAug: Semi-Supervised Node Classification with Data Augmentation. In *Proceedings of the 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '20), August 23–27, 2020, Virtual Event, CA, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3394486.3403063

1 INTRODUCTION

Semi-Supervised node classification is a fundamental task on graph data, which aims to classify the nodes in an (attributed) graph given the class labels of a few nodes [19]. For this task, Graph Convolutional Networks (GCNs) have achieved state-of-the-art performance [30]. Typically, GCNs predict the classes via the 'message-passing' mechanism, i.e., they aggregate the semantic representations between each node and its neighbors at each layer to generate the final-layer predictions. Thus, the prediction of a node relies on the attributes of other nodes and the graph structure in addition to its own attributes, all of which act as its feature.

In the general semi-supervised learning (SSL) problem, effectively using unlabelled data is essential [21], since labeled samples are scarce while unlabeled data are typically present in massive quantity and easier to obtain. In contrast, GCNs are trained only over the predictions of the labeled nodes by minimizing the supervised classification loss, but the predictions of the unlabeled nodes do not contribute to the training. This leads to the question: how can we effectively incorporate unlabeled data into GCN models?

Hence, in this work, we regularize the predictions of all nodes to be invariant with respect to the changes induced by Data Augmentation (DA), through minimizing the classification divergence between the original nodes and the augmented ones, a.k.a., consistency training [31]. Intuitively, DA makes changes to the input data in ways that should have relatively trivial effects on the final node classification, based on our human knowledge, but diverse effects to the input features. If we enforce the model predictions to be invariant with respect to these changes, we embed human knowledge on the labels of especially the unlabeled nodes into GCN models, which potentially yields the performance gains. However, to date, the DA techniques for graph data remains under-explored.

Data Augmentation has been applied successfully to the tasks on image data, e.g., image classification [31]. A significant difference between graph and image data is that nodes are connected, while images are isolated. There are various DA techniques for image classification that augment an image by changing its own attributes, e.g. permuting the RGB values, without affecting (the features and labels of) other images. However, modifying the attributes of a node or removing/adding an edge, can influence many other nodes. For example, a node close to the change is affected seriously to have its ground-truth label changed, and distant nodes keep their features unaltered. Thus, under a straightforward implementation of DA, it is difficult to ensure that the labels for each node are not altered significantly, while their features are augmented effectively. But this is the target of an effective DA technique [31].

To coordinate DA for different nodes, the central idea of this paper is to make them happen in '*parallel universes*', i.e., the modifications made for augmenting one node do not happen when augmenting another one. In other words, one can modify the whole graph for augmenting a node, and each node has an separate augmented graph (see Fig. 1). We encapsulate this idea in a novel framework, called **NodeAug** (Node-Parallel Augmentation), that augments different nodes individually and separately. To complement NodeAug, we propose three novel DA strategies considering the graphical factors, such as node degrees and attribute weights. We take a node-centric hierarchical view on the graph, i.e., we take the node to be augmented as the center and place other nodes on different levels based on their distances to the center. To augment the central node, we replace its attributes with others in its neighborhood and add/remove edges connecting them. We set the probability of these actions sensitive to their levels because the closer nodes/edges tend to be more influential for the central node.

NodeAug is a general module that can be applied for training popular GCN models, such as GCN [17], GAT [26], and LGCN [26]. A fact of these models is that the subgraph contributing to the prediction of a node, a.k.a., its receptive field, is typically much smaller than the whole graph [23]. Based on this fact, we propose a subgraph mini-batch method for efficient training. For example, in each augmented graph, since the consistency loss is computed only on the augmented node, we extract the subgraph corresponding to its receptive field as the input. Our method takes the subgraphs corresponding to a batch of nodes as the input at each training step. As a result, it has better generalization [16] and incurs fewer memory costs than the existing full-batch training method [17], which processes the whole graph at each iteration. The advantage is dominant when dealing with large-scale graphs, as illustrated empirically.

We evaluate NodeAug on the semi-supervised node classification task using the Cora, Citeseer, Pubmed [19], Coauthor-CS, and Coauthor-Physics [24] datasets. Quantitatively, we observe the improvements in accuracy for various GCN models. NodeAug improves the strong LGCN [13] and Graph U-Net [12] models by a significant margin.

Our contributions are as follows:

(1) **NodeAug**: We propose a novel methodology for the GCN consistency training using Data Augmentation, named NodeAug. NodeAug conducts DA for different nodes individually and separately, to coordinate DA for different nodes. Moreover, we propose three DA techniques on the graph data to complement NodeAug, by changing both the node attributes and the graph structure.

(2) **Subgraph Mini-Batch Training**: We take the receptive field subgraphs of a batch of nodes as the input at each training step, so as to use NodeAug efficiently. It induces better generalization [16] and reduces resource costs compared with the prior full-batch training.

(3) **Effectiveness**: Our experimental results show that NodeAug, with our DA techniques, yields significant gains for semi-supervised node classification over the strong GCN models.

2 RELATED WORK

Consistency Training in Semi-Supervised Learning. Several works have explored consistency training for SSL, showing it to work well on many benchmarks [31], [6]. Generally speaking, they inject noise into the data and regularize the model to make its predictions consistent with respect to the noise. VAT [21] defines the noise by approximating the direction of change in the input space that the model is most sensitive to. More recently, some works show that Data Augmentation (DA) is an effective approach for injecting the noise [18]. ICT [27] and MixMatch [1] employ Mixup [33] on top of augmentations. UDA [31] achieves state-of-the-art performance on multiple SSL tasks including image classification.

VAT holds the cluster assumption [5]. Based on this, VAT regularizes the model to be locally smooth on each sample, i.e., predictions remain invariant to the injected noise within a ball of a given radius measured by, for example, ℓ_2-norm. Different from VAT, UDA uses DA techniques to inject noise, such as rotating the images and replacing the document attributes. These actions adjust the input features drastically if measured by the ℓ_2-norm, but does not change their labels based on a human oracle. The miscellaneous modifications made by the DA techniques embed the label information on the unlabeled data into the trained model, which accounts for the superior performance of UDA.

As for semi-supervised node classification, some works have used consistency training to enhance GCN models [9]. BVAT [7] and GraphVAT [11] extend VAT to GCN models. Since removing/adding edges is binary and hard to measure by ℓ_2-norm, GraphVAT injects the noise only to the node attributes. The authors of [28] follow MixMatch to propose GraphMix, which saves the additional gradient computation for adversarial perturbations in BVAT and GraphVAT. It takes a Multi-Layer Linear Perceptron as the cousin network, and its DA technique is only to perturb the node attributes randomly. In contrast, our approach incorporates DA techniques, as UDA does, and does not require additional architecture like the cousin network of GraphMix. Furthermore, we design multiple advanced DA techniques that change both the node attributes and graph structure to augment the input effectively.

Data Augmentation. Data Augmentation plays a central role in training deep learning models. It only operates on the input data, without changing the model architecture, but improves the performance significantly. For example, in image classification, DA strategies such as horizontal flips, random erasing [34], Hide-and-Seek [25], and Cutout [8] have been shown to improve performance. Considering the similarities of the convolutional operations over the image and graph data, we propose two strategies to remove the edges, that deletes partial information, and add the edges, that

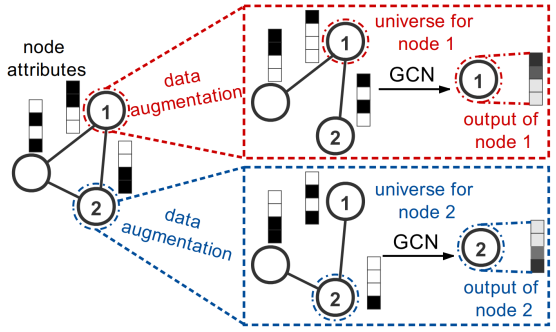


Figure 1: *NodeAug augments different nodes individually and separately.* NodeAug assumes the DA for a node can modify the whole graph. DA for different nodes happen in 'parallel universes' (node 1 in red and node 2 in blue). In each universe, we only consider the predictions of the augmented node, for example, node 1 in the red universe.

changes the convolutional orders. The authors of UDA [31] propose TF-IDF based word replacement strategies for augmenting the keyword features of documents. We compare the node attributes to the document keywords, both of which can be seen as (weighted) bag-of-words data. From this vantage, we propose the DA strategy of replacing the node attributes, additionally considering the structural information in the graph data.

3 METHODOLOGY

In this section, we propose NodeAug (Node-Parallel Augmentation) as the methodology for improving semi-supervised node classification using Data Augmentation. A NodeAug module accepts a GCN model as the input, e.g. GCN, GAT, LGCN, etc., and trains it with DA techniques to improve its inference performance. A NodeAug module consists of two main components: (i) A *'parallel universes' DA scheme* to conduct DA for different nodes individually and separately. (ii) A *consistency training scheme* to minimize the classification divergence between the predictions of the original nodes and augmented ones.

To complement NodeAug, we propose three DA techniques for augmenting a node by modifying the graph structure and node attributes. Moreover, for the sake of efficiency, we propose a subgraph mini-batch training method, taking the subgraphs corresponding to the receptive fields of a batch of nodes as the input. Details are discussed next.

3.1 Consistency Training with Data Augmentation on Graph Data

We define a graph as $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ denotes the set of nodes, and $\mathcal{E}$ is the set of edges. Typically, GCNs give the conditional output distribution of node i as follows:

$$p\Big(y\Big|\mathcal{G}_i = (\mathcal{V}_i, \mathcal{E}_i), \{\mathbf{x}_j\}_{j\in\mathcal{V}_i}, \Theta\Big), \tag{1}$$

where y is the class, $\mathbf{x}_j$ denotes the attributes of node j, Θ is the parameter set. The node set $\mathcal{V}_i$ and the edge set $\mathcal{E}_i$ constitute the subgraph $\mathcal{G}_i$, which corresponds to the receptive field of node i. Note that $\mathcal{G}_i$ is typically much smaller than $\mathcal{G}$, i.e., $|\mathcal{V}_i| \ll |\mathcal{V}|, |\mathcal{E}_i| \ll |\mathcal{E}|$. For example, $\mathcal{G}_i$ is the two-hop neighborhood of node i in a 2-layer GCN with row-normalization [29]. $\mathcal{G}_i = (\mathcal{V}_i, \mathcal{E}_i)$ and $\{\mathbf{x}_j\}_{j\in\mathcal{V}_i}$ contribute to the prediction of node i, as illustrated in Eq. (1), which can be seen as the features for classifying node i.

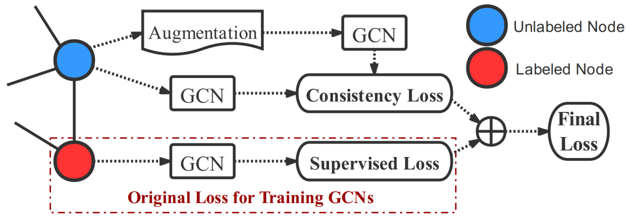


Figure 2: *NodeAug conducts consistency training on all the nodes.* We add the consistency loss $\mathcal{L}_C$ in (2) computed on both labeled (red) and unlabeled blue nodes, to the supervised classification loss $\mathcal{L}_S$ utilized by the existing GCN models.

We obtain two insights for DA from (1):

- For augmenting node i, it would be more effective if we modify $\mathcal{G}_i = (\mathcal{V}_i, \mathcal{E}_i)$ and $\{\mathbf{x}_j\}_{j\in\mathcal{V}_i}$ than altering $\mathbf{x}_i$, since $\mathbf{x}_i$ is only part of the input features for node i.
- To augment node i, the modifications to $\{\mathbf{x}_j\}_{j\in\mathcal{V}_i}, \mathcal{G}_i$ influence the input features of other nodes, and vice versa.

Following the first point, during data augmentation, we modify the attributes of other nodes and the graph structure instead of the node itself, so as to augment the input feature of node i effectively. To avoid the case in the second point, we propose the 'parallel universes' DA scheme: we conduct the DA for different nodes individually and separately (see Fig. 1). Each node has its own separate augmented graph. This ensures that DA for different nodes does not influence each other, and the DA magnitude for each node is under control. This scheme is named as NodeAug, short for Node-Parallel Augmentation.

Assuming some DA actions have taken place on graph $\mathcal{G}$ for node i, we have the augmented $\hat{\mathcal{G}}$ and node attributes $\{\hat{\mathbf{x}}_j, j \in \mathcal{V}\}$. Then, we denote the subgraph corresponding to its receptive field as $\hat{\mathcal{G}}_i$. Valid DA changes the input features but does not alter the labels. We embed this knowledge into the model by minimizing the consistency loss as follows:

$$\mathcal{L}_C = \frac{1}{|\mathcal{V}|}\sum_{i\in\mathcal{V}} \mathcal{D}_{\mathrm{KL}}\Big(p\Big(y\Big|\mathcal{G}_i, \{\mathbf{x}_j\}_{j\in\mathcal{V}_i}, \tilde{\Theta}\Big) \Big\| p\Big(y\Big|\hat{\mathcal{G}}_i, \{\hat{\mathbf{x}}_j\}_{j\in\hat{\mathcal{V}}_i}, \Theta\Big)\Big), \tag{2}$$

where $\mathcal{D}_{\mathrm{KL}}(\cdot\|\cdot)$ is the Kullback-Leibler divergence [15] measuring the divergence between two distributions. $\tilde{\Theta}$ is a fixed copy of the current parameter Θ, indicating that the gradient is not propagated through $\tilde{\Theta}$, as suggested by [21].

In addition to $\mathcal{L}_C$, we follow the supervised classification loss $\mathcal{L}_S$ utilized by prior GCN models, which typically contains the element of cross-entropy function $\mathcal{H}(\cdot,\cdot)$ [14]:

$$\frac{1}{|\mathcal{V}_L|}\sum_{i\in\mathcal{V}_L} \mathcal{H}\Big(\mathbf{y}_i^*, p\Big(y\Big|\mathcal{G}_i = (\mathcal{V}_i, \mathcal{E}_i), \{\mathbf{x}_j\}_{j\in\mathcal{V}_i}, \Theta\Big)\Big), \tag{3}$$

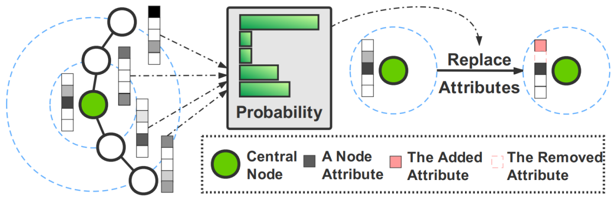


Figure 3: *We replace attributes to augment the central node.* The central node is in green. NodeAug computes the attribute distribution from the 2-hop neighborhood corpus (excluding itself). After some central node attributes are removed, we sample attributes from the neighborhood to take the place (pink).

where $\mathcal{V}_L$ is the set of labeled nodes, and $\mathbf{y}^*$ is the one-hot label of node i.

Overall, to make GCN models not only classify the few labeled nodes correctly, but also classify both the labeled and unlabeled nodes consistently before and after augmentation, we minimize the loss as follows:

$$\mathcal{L} = \mathcal{L}_S + \alpha \mathcal{L}_C, \tag{4}$$

as shown in Fig. 2. We introduce the hyper-parameter α to balance $\mathcal{L}_C$ and $\mathcal{L}_S$, which is set to 1 by default.

Discussion The consistency loss $\mathcal{L}_C$ defined in Eq. (2) is similar to that used in UDA [31]. The major difference is that UDA does not have the subgraphs $\mathcal{G}_i$ and $\hat{\mathcal{G}}_i$ as inputs, because the samples that UDA classify, e.g., images, are not connected as the nodes in a graph. In principle, the input features of different data samples considered by UDA do not have overlaps, while those of the nodes have. Thus, to conduct consistency training for GCNs on the graph data, we propose NodeAug to separate the augmentations for different nodes, in order to coordinate DA for different nodes and simplify the design of the graph DA strategies.

Compared with the prior consistency training methods on graphs, GraphVAT [11] and GraphMix [28], we mainly make three advancements in NodeAug. First, we design NodeAug inspired by UDA with our proposed 'parallel-universe' DA scheme. Note that UDA is more effective than VAT and MixMatch (as discussed in Sec. 2), which are the bases of GraphVAT and MixMatch respectively. Second, NodeAug incurs neither the additional computation for adversarial perturbations in GraphVAT nor the extra cousin network in GraphMix. This makes NodeAug more succinct and efficient. Last but not least, GraphVAT and GraphMix inject noise into the node attributes. In contrast, NodeAug offers additional flexibility allowing more advanced DA techniques, that operate not only on the node attributes but also the graph structure, which is introduced in the next subsection in detail.

3.2 Data Augmentation Strategies

We propose three DA strategies for the graph data: 1). replacing attributes; 2). removing edges; 3). adding edges. For the node to be augmented, the first strategy replaces its attributes with other attributes that are potentially relevant to it, while the latter two change the paths over which GCN models aggregate the attributes from other nodes to it. The strategies manipulate the input features of the node to be augmented in different ways, while not changing its label with moderate magnitudes.

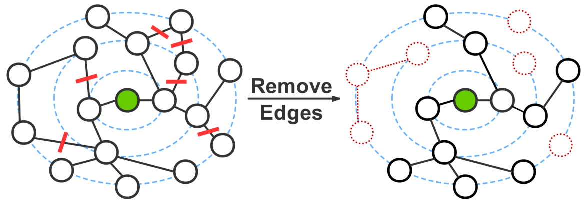


Figure 4: *We remove edges to augment the central node.* The augmented node is in green. The edges and nodes with the paths to the central node removed are in red. We remove all the edges with the removing probability p_{e-rem} (defined in Eq. (8)) higher than 0.5 for the visualization.

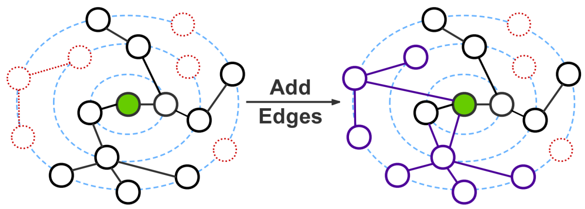


Figure 5: *We add edges to augment the central node.* The augmented node is in green. The edges and nodes with the level changed are in purple. We add all the edges with the adding-ing probability p_{e-add} (defined in Eq. (9)) higher than 0.1 for the visualization.

For the node to be augmented, we assume closer nodes/edges tend to be more influential. We take a 'node-centric' hierarchical view on the graph, i.e., the node to be augmented is at the center, and we place other nodes onto different levels based on their distance to the center. For example in Fig. 3, the node to be augmented is colored green, and the circles denote the levels. The nodes directly connected to the green node are on the inner circle (level 1), while the 2-hop ones are on the outer circle (level 2), etc. For the ease of expression, we denote the *smaller* level index as the *higher* one, e.g., the nodes on level 1 are higher than those on level 2. Moreover, we define the level of an edge as the level of the node on its lower end, e.g., the edge connecting the central node and another on level 1 is on level 1. To simplify the DA strategies, we only consider modifications at levels higher than level **3**. The reason is that the number of the considered nodes/edges grows exponentially as the level index increases, but the nodes/edges lower than level 3 do not contribute much to the prediction of the central node.

The search space for the hyper-parameters of DA magnitudes is large if we set the parameters separately for different DA techniques. Hence, for the convenience of setting the hyper-parameters, we use a unified hyper-parameter p to control the magnitude of all the three DA strategies. A higher value of p implies a larger magnitude. Next, we introduce our DA strategies in detail.

Replace Attributes Node attributes can be thought of as a weighted bag of words, where a higher weight indicates larger importance. To generate both diverse and valid samples, we replace the uninformative attributes of the central node with the attributes possibly relevant to it. Suppose the minimum, maximum and average attribute weights of the central node are $w_{\min}$, $w_{\max}$ and w_{ave} respectively. We set the probability of having an attribute with weight w to be removed as

$$p_{a-rem} = \min\left(p\frac{w_{\max} - w}{w_{\max} - w_{\text{ave}}}, 1\right), \tag{5}$$

where p, as defined above, is the hyper-parameter for controlling the augmentation magnitude. With a larger p, the probability p_{a-rem} increases for all the attributes, i.e., they are more likely to be removed (larger DA magnitude). $w_{\max} - w_{\text{ave}}$ acts as a normalization term, so that p_{a-rem} for every attribute belongs to the range between 0 and $p\frac{w_{\max}-w_{\min}}{w_{\max}-w_{\text{ave}}}$. $(w_{\max} - w)$ shows that p_{a-rem} is a locally linear function negatively related to the attribute weight w. p_{a-rem} increases as w decreases, because we tend to remove the unimportant attributes. The function $\min(\cdot, 1)$ truncates p_{a-rem} to be a valid probability value between 0 and 1.

After removing an attribute, we sample another one to take its place. The sampling is done on the neighborhood including the nodes on levels 1 and 2, denoted as $\mathcal{V}_{2-hop}$, because we assume that the nodes in $\mathcal{V}_{2-hop}$ are more likely to have relevant attributes than the others distant to the center. We compute the score s of an attribute by the sum of its weights of the nodes in $\mathcal{V}_{2-hop}$. Denote the set of all the attributes as $\mathcal{X}$, and the minimum and average scores in $\mathcal{X}$ as $s_{\min}$ and s_{ave} respectively. Then we set the probability of sampling an attribute of score s as

$$p_{a-sam} = \frac{s - s_{\min}}{|\mathcal{X}|\,(s_{\text{ave}} - s_{\min})}, \tag{6}$$

where $|\mathcal{X}| \cdot (s_{\text{ave}} - s_{\min})$ is a normalization term to ensure that the sum of the probabilities of sampling different attributes in $\mathcal{X}$ is 1. From $s - s_{\min}$, we know that p_{a-sam} is a linear function of s, and larger s infers a larger p_{a-sam}. This is justified because a higher s implies that the attribute is more popular in the neighborhood $\mathcal{V}_{2-hop}$, and thus we are more likely to sample it. This strategy is visualized in Fig. 3. We set the weight of the newly added attribute to that of the removed one.

Remove Edges When removing edges, we aim to retain important edges and remove the uninformative ones. We evaluate the importance of an edge by the node it connects on the lower end, since the edges propagate the attributes from the low-level nodes to the higher ones and finally to the central node. Removing an edge deletes a path for the node on its lower end to transmit its attributes to the center. Suppose the degree of the node on its lower end is d_{low}. We define the score of an edge as

$$s_e = \log(d_{low}), \tag{7}$$

because a node of a larger degree tends to be more influential. For example, a famous person in a social network tends to have numerous followers. We use the function $\log(\cdot)$ but not a linear one because the node degrees in a graph can vary across orders of magnitudes, such as in a scale-free network [10], while the node degree does not indicate the importance as significantly as the attribute weight, on which we apply the linear function directly as shown in Eq. (5). Suppose the maximum and average edge scores on level l are $s_{e-\max}^{(l)}$ and $s_{e-\text{ave}}^{(l)}$ respectively. We set the probability of removing the edge with score s_e on level l to

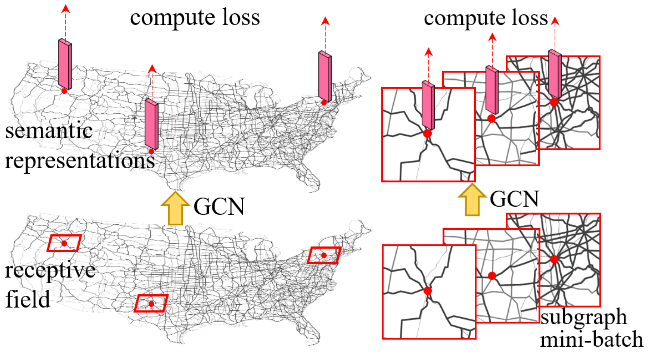


Figure 6: (*left*) The full-batch training method takes the whole graph as the input at each training step, although only the final-layer semantic representations of a few nodes contribute to the loss computation, of which the receptive fields are much smaller than the whole graph. (*right*) Our subgraph mini-batch approach extracts the subgraphs corresponding to the receptive fields of the considered nodes, and batches the subgraphs as the input at each training step.

$$p_{e-rem} = \min\left(pl\frac{s_{e-\max}^{(l)} - s_e}{s_{e-\max}^{(l)} - s_{e-\text{ave}}^{(l)}}, 1\right), \tag{8}$$

An example of this strategy with $p = 0.3$ is illustrated in Fig. 4. Similar to p_{a-rem} defined in Eq. (5), p_{e-rem} is a truncated linear function negatively related to s_e. The difference is that in Eq. (8), the level index l influences p_{e-rem}. The distant edges (larger l) tend to be less informative, so we are more likely to remove it (higher p_{e-rem}).

Add Edges We add edges between the central node and some on levels 2 and 3, which are not connected to the central one directly, but potentially have important information for predicting the central node. For example, in a citation network, paper A cites B because it uses a method M introduced in B. However, M is not the main contribution of B, and it is paper C cited by B that proposes M. Then, adding an edge between A and C augments the input features of A without changing its label. Similar to the edge score s_e defined in Eq. (7), we define the score for a node of degree d as $s_n = \log(d)$. Denote the minimum, maximum and average node scores on level l as $s_{n-\min}^{(l)}$, $s_{n-\max}^{(l)}$, and $s_{n-\text{ave}}^{(l)}$ respectively. We set the probability of adding an edge between the center and the node with score s_n on level l to

$$p_{e-add} = \min\left(\frac{p}{l}\frac{s_n - s_{n-\min}^{(l)}}{s_{n-\text{ave}}^{(l)} - s_{n-\min}^{(l)}}, 1\right), \tag{9}$$

A larger p incurs the higher probability p_{e-add} for all the nodes on levels 2 and 3, i.e., larger DA magnitude. $(s_{n-\text{ave}}^{(l)} - s_{n-\min}^{(l)})$ acts as a normalization term, so that p_{e-add} falls into the range between 0 and $\frac{p}{l}\frac{s_{n-\max}^{(l)} - s_{n-\min}}{s_{n-\text{ave}}^{(l)} - s_{n-\min}^{(l)}}$. $s_n - s_{n-\min}^{(l)}$ shows that p_{e-add} is locally a linear function for the weight w. p_{a-rem} increases as s_n increases, because those important nodes tend to be connected directly. The function $\min(\cdot, 1)$ ensures p_{e-add} is a valid probability value between 0 and 1. The distant nodes (larger l) tend to be less important, so we are less likely to build connections for it (smaller p_{e-add}). An example of this strategy with $p = 0.3$ is illustrated in Fig. 5.

Discussion We compare our DA strategies on graph data to those for images and texts, which have shown effective performance. **Cutout** [8] is a simple yet effective DA approach for augmenting images, which randomly masks out some regions of the input. Removing edges, as shown in Fig. 4, is similar to Cutout, in the sense that it blocks some nodes from propagating their attributes to the central one. **Shearing** and **Resizing** an image [2] changes the orders in which the neural networks do convolutions. This is similar to adding edges for the central node, in the sense that the added edges change the structures of graph convolutions. **TF-IDF based word replacement** [31] replace the uninformative keywords of a document with the keywords of other documents. It is similar to our attribute replacement approach, in the sense that both of them augment bag-of-word features, while ours additionally takes the graph structural information into consideration.

3.3 Subgraph Mini-Batch Training

Most prior works on semi-supervised node classification feed the whole graph to GCN models for training, a.k.a., full-batch training. It computes the supervised loss for all the labeled nodes at each step, but causes excessive resource requirements when dealing with large-scale graphs, which limits the practical applications of GCNs. With NodeAug, since each node has a separate augmented graph, processing all the augmented graphs as a full-batch aggravates the limitation further.

In this work, we propose a subgraph mini-batch training method to address the resource problems for large graphs. From Eq. (2), we know that to classify node i, we only need to take the subgraph $\mathcal{G}_i$ corresponding to the receptive field of node i as the input, which is generally much smaller than the whole graph. At each training step, only the predictions of the nodes in the batch contribute to the loss computation, e.g., an augmented node in each augmented graph for $\mathcal{L}_C$. Thus, extracting the subgraphs for them to form a batch as the inputs can significantly reduce the resource requirements during training, as visualized in Fig. 6. The extraction step can be completed efficiently with existing algorithms. For instance, with the models of GCN, GAT, LGCN, etc., the receptive field of a node is its neighborhood within a constant number of hops, which can be obtained by breadth-first search [20]. This ensures that our subgraph mini-batch method is convenient to be implemented on miscellaneous platforms [4].

Compared with full-batch training, our subgraph mini-batch approach holds advantages in its flexibility, generalization, and resource occupation. Ours takes the subgraphs of a flexible number of nodes as the input, while the full-batch method processes the whole graph at each step. In addition, with the mini-batch training, GCNs are likely to converge to better generalization performance [16]. Moreover, in terms of the space complexity, the memory occupation of our method is constant with the fixed batch size, while that of the full-batch technique grows with the graph size. For NodeAug, at each training step, we extract the receptive field subgraphs of a batch of the augmented nodes for the consistency loss $\mathcal{L}_C$, and those of the labeled nodes for the supervised loss $\mathcal{L}_S$. The former extraction is conducted on both the original and augmented graphs, while the latter one is done only on the original graph.

Table 1: Statistics of the utilized datasets

Dataset	# Nodes	# Edges	# Classes	# Attributes
Cora	2,708	5,429	7	1,433
Citeseer	3,327	4,732	6	3,703
Pubmed	19,717	44,338	3	500
Co-CS	18,333	81,894	67	6,805
Co-Phy	34,493	247,962	5	8,415

4 EXPERIMENTS

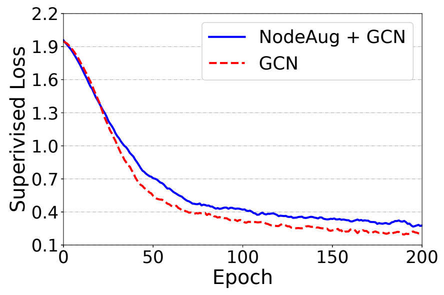
In this section, we first present the effectiveness improvements over various GCN architectures given by NodeAug. Next, we analyze the benefits in terms of memory and time costs due to our subgraph mini-batch method. Then, the losses, accuracy, and the final-layer representations of GCNs with NodeAug are visualized compared with GCN without NodeAug. Finally, we conduct ablation studies to show the influence of different components of NodeAug, as well as the sensitivity with respect to the hyper-parameters of NodeAug.

We use standard benchmark datasets: Cora, Citeseer, Pubmed [19], Coauthor-CS (short as Co-CS) and Coauthor-Physics (short as Co-Phy) [24] for evaluation. The former three are citation networks, and the latter two are co-author networks. Each of them contains an unweighted adjacency matrix and bag-of-words features. The statistics of these datasets are presented in Table 1. We take the popular GCN architectures of GCN [17], GAT [26], LGCN [13], and Graph U-Net [12] as the models to be enhanced by NodeAug. We additionally use the methods GMNN [22], GraphVAT [11], and GraphMix [28] as baselines for comparison.

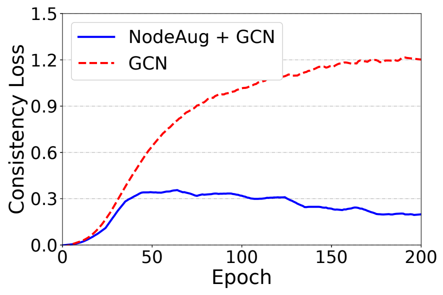
For the hyper-parameters of different GCN architectures, e.g., the number of layers, the number of hidden units, the optimizer, the learning rate, we set them as suggested by their authors. For the hyper-parameters of our NodeAug, We set $p = 0.3$ for the magnitude of our DA techniques, and $\alpha = 1$ for the weight of the consistency loss by default. At each training step, we randomly sample 5 nodes from the labeled ones for the supervised loss, and 64 from all the nodes for the consistency loss to form a batch.

4.1 Semi-Supervised Node Classification

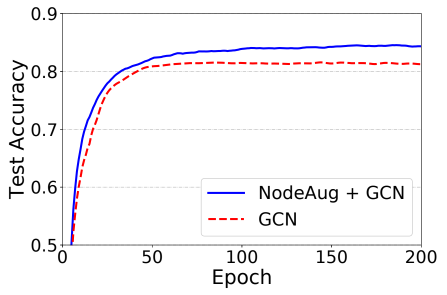
We conduct the experiments on different datasets with both standard splits and random splits. The datasets Citeseer, Cora, and Pubmed hold the standard splits [17], [32] that are widely utilized in different works, where 20 nodes per class form the labeled nodes in the training set, 500 nodes are used for the validation and 1000 nodes for testing. We present the results on the standard splits in Table 2. We report the mean accuracy and the standard derivations of 100 runs with random weight initialization. We observe that NodeAug improves the popular GCN models GCN, GAT, LGCN, and Graph U-Net by a significant margin in terms of the test accuracy. Specifically, on Cora, Citeseer, Pubmed, it enhances the strong



(a) Supervised Loss



(b) Consistency Loss



(c) Test Accuracy

Figure 7: The training curves of NodeAug and GCN on the standard split of Cora dataset.

Table 2: Results of node classification on the standard splits in terms of test accuracy (%). We report mean and standard derivations of 100 trails with random weight initialization.

Method	Cora	Citeseer	Pubmed
GCN [17]	81.5	70.3	79.0
GAT [26]	83.0	72.5	79.0
LGCN [13]	83.8	73.0	79.5
Graph U-Net [12]	84.4	73.2	79.6
GMNN [22]	83.7	73.1	81.8
GraphVAT [11]	82.9	73.8	79.5
GraphMix (GCN) [28]	83.9	74.5	81.0
GraphMix (GAT) [28]	83.3	73.1	81.1
NodeAug + GCN	84.3±0.5	74.9±0.5	81.5±0.5
NodeAug + GAT	84.8±0.2	75.1±0.4	81.6±0.6
NodeAug + LGCN	85.8±0.5	**75.5±0.5**	81.8±0.3
NodeAug + Graph U-Net	**86.2±0.5**	75.4±0.8	**82.1**±0.4

LGCN [13] and Graph U-Net [12] models by 2.4%, 2.9%, 2.8%, and 2.1%, 2.5%, 2.9% respectively. Meanwhile, it outperforms the current state-of-the-art methods by 2.1% (over Graph U-Net), 1.2% (over GraphMix with GCN), and 0.4% (over GMNN) on the three datasets respectively.

We highlight two points in this set of results. First, GraphVAT and GraphMix are both the consistency training approaches designed for GCNs. The authors of the former one gives the results for GCN, while the latter one presents those for both GCN and GAT. Over both GCN and GAT, NodeAug achieves higher improvements on the test accuracy than GraphVAT and GraphMix. The reasons are as follows. We design NodeAug inspired by UDA with our proposed 'parallel-universe' DA scheme, where UDA through its use of DA is more effective than VAT and MixMatch (as discussed in Sec. 2), which are the bases of GraphVAT and MixMatch respectively. Moreover, NodeAug modifies both the node attributes and the graph structure, while GraphVAT and GraphMix only inject noise to the node attributes. Second, GAT is thought of as an advanced variant of GCN. However, GraphMix does not improve GAT as effectively as GCN, while NodeAug does. This is explained by the cousin network in GraphMix: this extra structure may not fit the advanced GCN models well. Our NodeAug is succinct without the extra architecture, which helps NodeAug to enhance the advanced GCN models consistently.

The second set of experiments is on the random splits. We randomly select $K \in \{5, 10\}$ nodes per class to form the training set, and the same number of samples for the validation. All the remaining ones are put into the testing set. We make 20 random splits and conduct the experiments for 100 trials with random weight initialization for each split. We present the accuracy on the random splits in Table 3. Empirically, NodeAug outperforms the benchmark methods on enhancing the performance of GCN for different sizes of the training set. In principle, with less labeled nodes, i.e., smaller K, our method gives larger accuracy improvements, because NodeAug utilizes the massive unlabeled nodes with our advanced DA techniques for the consistency training.

4.2 Subgraph Mini-Batch Training

We evaluate the performance of our subgraph mini-batch training method using the Co-Phy dataset, which is the largest one among the 5 datasets. We follow the experimental setting of the random splits with the number of labeled samples per class $K = 5$, as introduced in the last subsection. The methods we evaluate include: GCN with the full-batch training, GCN with NodeAug using our subgraph mini-batch training (NodeAug$_{sub}$), and GCN with NodeAug using the full-batch training (NodeAug$_{full}$). NodeAug$_{full}$ puts all the augmented graphs and the original graph into a single batch in each training step to compute the loss. The average numbers of nodes and edges processed at each training step are reported in Table 4, where the running time is the training time until convergence using a single 1080Ti GPU. We notice that our subgraph mini-batch method significantly reduces the number of the edges and the nodes processed at each iteration. Compared with GCN, the data volume that NodeAug$_{sub}$ processes at each training step is smaller, and the running time that it consumes is shorter, since only the data within the receptive fields are processed, out of which the data do not contribute to the loss computation. Because NodeAug$_{full}$ needs to process too much data per iteration for us to feed them into the GPU, its running time and test accuracy are unavailable (denoted as N.A.). In terms of effects on the generalization of the subgraph mini-batch method, we cannot evaluate it from Table 4, since the accuracy from NodeAug$_{full}$ is absent. But we apply the subgraph

Table 3: Results of node classification with the random splits of the varied number K of labeled examples per class, in terms of test accuracy (%). We report mean and standard derivations of 2000 trials on 20 splits with random weight initialization.

Method	Cora		Citeseer		Pubmed		Co-CS		Co-Phy	
	$K = 5$	$K = 10$	$K = 5$	$K = 10$	$K = 5$	$K = 10$	$K = 5$	$K = 10$	$K = 5$	$K = 10$
GCN [17]	65.9±3.5	72.3±2.2	55.8±4.1	64.8±2.9	64.9±4.5	75.2±1.1	72.1±3.2	83.7±3.1	73.3±4.2	85.2±3.4
GraphVAT [11]	68.6±4.2	75.7±3.7	56.9±5.3	66.2±2.7	66.4±3.7	75.9±2.6	73.1±5.3	84.2±2.8	74.5±5.7	85.7±4.2
GraphMix (GCN) [28]	71.2±6.1	79.2±2.3	58.8±4.3	71.0±1.7	67.1±4.4	76.6±3.7	74.9±3.9	86.3±3.4	76.1±3.3	87.1±1.8
NodeAug + GCN	**74.5±2.9**	**81.1±1.4**	**61.9±3.1**	**71.8±2.5**	**69.3±3.5**	**77.4±1.7**	**76.2±2.6**	**86.9±1.2**	**77.6±3.8**	**87.7±2.6**

Table 4: Results of node classification with the random splits of 5 labeled examples per class on the *Co-Phy* dataset. The methods GCN and NodeAug (with GCN) are compared. For the fair comparison, we apply the full-batch training method, employed by GCN, to NodeAug, denoted as NodeAug$_{full}$. Accordingly, NodeAug with the subgraph mini-batch training is NodeAug$_{sub}$. We report mean values of 2000 trials on 20 splits with random weight initialization.

Method	# Nodes	# Edges	Time	Accuracy (%)
GCN	34,493	247,962	373s	73.3
NodeAug$_{full}$	1.2×10^9	8.6×10^9	N.A.	N.A.
NodeAug$_{sub}$	**4912**	**6862**	**39s**	**77.6**

mini-batch training for the original GCN and show the results in Table 5, which is analyzed later.

4.3 Visualization of NodeAug

We study the effects of NodeAug on GCN during training. We depict the supervised loss, consistency loss, and test accuracy at each training epoch in Fig. 7 on the standard splits of Cora dataset. As we can see, the supervised loss on the labeled nodes still converges to a low value under NodeAug, although NodeAug spends extra efforts on minimizing the consistency loss on all the nodes.

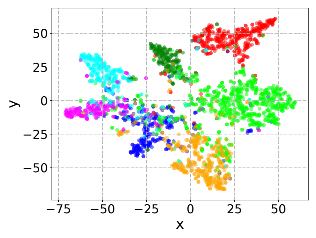
In addition, we observe that the consistency loss for both NodeAug and GCN grows at the first stage of training, because initially, the classifier predicts different nodes nearly randomly, inducing small divergence of classification results with respect to our DA actions. After training for more iterations, the gap of the consistency losses between NodeAug and GCN becomes large. NodeAug effectively inhibits the increase of the inconsistent behaviors of the classifier given the data augmentation, and thus converges to better test accuracy.

In Fig. 8, we visualize the final-layer representations of NodeAug and GCN via t-SNE [3]. NodeAug makes the learned representations for the nodes in the same class more concentrated than GCN, which potentially improves the classification performance of GCN.

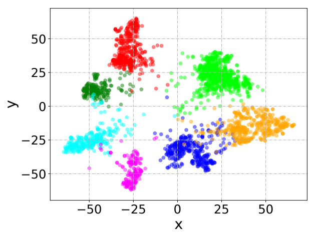
4.4 Ablation Study

We evaluate the contributions of different components of NodeAug. We apply different techniques of NodeAug incrementally on GCN. The results are presented in Table 5. The second row (+ Subgraph Training) means that we apply our subgraph mini-batch training method to GCN without the consistency training, i.e., we extract the subgraphs of a batch of labeled nodes as the input at each training iteration. Although we do not use consistency training on GCN, the mini-batch training method makes GCN converge to better generalization, and thus achieves improvements on the test accuracy. With the consistency training and the advanced DA techniques, the performance of GCN is enhanced further. Among different DA techniques, it is shown empirically that adding edges can lead to the largest benefits. In addition, from the third row for the Attribute Replace DA technique, we know that only changing the attributes of the node is not as effective as the strategy that additionally augments the graph structure. This agrees with our first insight obtained from Eq. (1).

Finally, we investigate the sensitivity of NodeAug to the hyper-parameters: p to control the DA magnitudes, and α to adjust the weight of consistency loss. The result is visualized in Fig. 9. We alter p among $\{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$ and α among $\{0.1, 1, 10, 10^2, 10^3\}$. As we can see, the accuracy of NodeAug with GCN is relatively smooth when parameters are within certain



(a) GCN

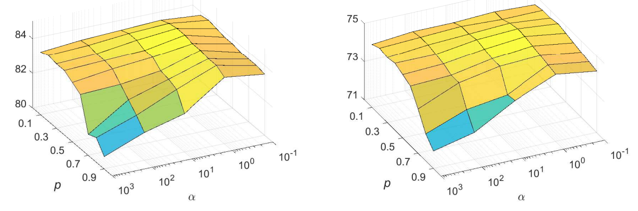


(b) NodeAug + GCN

Figure 8: The final-layer representations of the nodes in the Cora dataset (visualized by t-SNE [3]). Colors denote the ground-truth class labels.

Table 5: Effects of different components of NodeAug evaluated on the standard split of the Cora dataset.

Technique	Accuracy (%)	Δ	Cumu Δ
GCN	81.5	0	0
+ Subgraph Training	81.7	+0.2	+0.2
+ Replace Attr.	82.3	+0.6	+0.8
+ Remove Edges	83.2	+0.9	+1.7
+ Add Edges	**84.3**	**+1.1**	**+2.8**



(a) `Cora` **(b) `Citeseer`**

Figure 9: The test accuracy (z-axis) of NodeAug with GCN on the standard splits of the datasets `Cora` and `Citeseer` with different hyper-parameters p and α.

ranges. However, extremely large values of p and α results in low performances on both datasets, which should be avoided in practice. Moreover, increasing p from 0.1 to 0.3 improves the accuracy of both datasets, demonstrating that diverse changes given by our DA techniques can improve the performance of GCN.

5 CONCLUSION

In this paper, we study the problem of exploring Data Augmentation techniques to strengthen the GCN models by consistency training. We propose a novel methodology named NodeAug, which conducts DA for different nodes individually and separately, to coordinate DA for different nodes. NodeAug is a generic framework that can use any DA strategies to enhance GCN models. In our work, considering the distances between DA actions and the node to be augmented, we propose three DA techniques on the graph data by adjusting both the node attributes and the graph structure. To use NodeAug efficiently, we take the subgraphs corresponding to the receptive fields of a batch of nodes as the input at each training step, inducing better effectiveness and efficiency compared with the prior full-batch training method. Our experimental results show that NodeAug, with our DA techniques, yields significant gains for semi-supervised node classification. Future work could introduce other advanced DA strategies to enhance our approach.

ACKNOWLEDGMENTS

This paper is supported by NUS ODPRT Grant R252-000-A81-133 and Singapore Ministry of Education Academic Research Fund Tier 3 under MOEs official grant number MOE2017-T3-1-007.

REFERENCES

[1] David Berthelot, Nicholas Carlini, Ian Goodfellow, Nicolas Papernot, Avital Oliver, and Colin A Raffel. 2019. Mixmatch: A holistic approach to semi-supervised learning. In *Advances in Neural Information Processing Systems*. 5050–5060.
[2] Marcus D Bloice, Christof Stocker, and Andreas Holzinger. 2017. Augmentor: An Image Augmentation Library for Machine Learning. *arXiv preprint arXiv:1708.04680* (2017).
[3] Andreas Buja, Dianne Cook, and Deborah F Swayne. 1996. Interactive high-dimensional data visualization. *Journal of computational and graphical statistics* 5, 1 (1996), 78–99.
[4] Aydin Buluç and Kamesh Madduri. 2011. Parallel breadth-first search on distributed memory systems. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–12.
[5] Olivier Chapelle and Alexander Zien. 2005. Semi-supervised classification by low density separation.. In *AISTATS*, Vol. 2005. Citeseer, 57–64.
[6] Kevin Clark, Minh-Thang Luong, Christopher D Manning, and Quoc V Le. 2018. Semi-supervised sequence modeling with cross-view training. *arXiv preprint arXiv:1809.08370* (2018).
[7] Zhijie Deng, Yinpeng Dong, and Jun Zhu. 2019. Batch virtual adversarial training for graph convolutional networks. *arXiv preprint arXiv:1902.09192* (2019).
[8] Terrance DeVries and Graham W Taylor. 2017. Improved Regularization of Convolutional Neural Networks with Cutout. *arXiv preprint arXiv:1708.04552* (2017).
[9] Ming Ding, Jie Tang, and Jie Zhang. 2018. Semi-supervised learning on graphs with generative adversarial nets. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. 913–922.
[10] Holger Ebel, Lutz-Ingo Mielsch, and Stefan Bornholdt. 2002. Scale-free topology of e-mail networks. *Physical review E* 66, 3 (2002), 035103.
[11] Fuli Feng, Xiangnan He, Jie Tang, and Tat-Seng Chua. 2019. Graph adversarial training: Dynamically regularizing based on graph structure. *IEEE Transactions on Knowledge and Data Engineering* (2019).
[12] Hongyang Gao and Shuiwang Ji. 2019. Graph U-Nets. *arXiv preprint arXiv:1905.05178* (2019).
[13] Hongyang Gao, Zhengyang Wang, and Shuiwang Ji. 2018. Large-Scale Learnable Graph Convolutional Networks. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 1416–1424.
[14] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep learning*. MIT press.
[15] James M Joyce. 2011. Kullback-leibler divergence. *International encyclopedia of statistical science* (2011), 720–722.
[16] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. 2016. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836* (2016).
[17] Thomas N Kipf and Max Welling. 2016. Semi-supervised Classification With Graph Convolutional Networks. *arXiv preprint arXiv:1609.02907* (2016).
[18] Samuli Laine and Timo Aila. 2016. Temporal ensembling for semi-supervised learning. *arXiv preprint arXiv:1610.02242* (2016).
[19] Ben London and Lise Getoor. 2014. Collective Classification of Network Data. *Data Classification: Algorithms and Applications* 399 (2014).
[20] Lijuan Luo, Martin Wong, and Wen-mei Hwu. 2010. An effective GPU implementation of breadth-first search. In *Design Automation Conference*. IEEE, 52–55.
[21] Takeru Miyato, Shin-ichi Maeda, Masanori Koyama, and Shin Ishii. 2018. Virtual adversarial training: a regularization method for supervised and semi-supervised learning. *IEEE transactions on pattern analysis and machine intelligence* 41, 8 (2018), 1979–1993.
[22] Meng Qu, Yoshua Bengio, and Jian Tang. 2019. Gmnn: Graph markov neural networks. *arXiv preprint arXiv:1905.06214* (2019).
[23] Pei Quan, Yong Shi, Minglong Lei, Jiaxu Leng, Tianlin Zhang, and Lingfeng Niu. 2019. A Brief Review of Receptive Fields in Graph Convolutional Networks. In *IEEE/WIC/ACM International Conference on Web Intelligence-Volume 24800*. ACM, 106–110.
[24] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. 2018. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868* (2018).
[25] Krishna Kumar Singh, Hao Yu, Aron Sarmasi, Gautam Pradeep, and Yong Jae Lee. 2018. Hide-and-Seek: A Data Augmentation Technique for Weakly-Supervised Localization and Beyond. *arXiv preprint arXiv:1811.02545* (2018).
[26] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph Attention Networks. *arXiv preprint arXiv:1710.10903* (2017).
[27] Vikas Verma, Alex Lamb, Juho Kannala, Yoshua Bengio, and David Lopez-Paz. 2019. Interpolation consistency training for semi-supervised learning. *arXiv preprint arXiv:1903.03825* (2019).
[28] Vikas Verma, Meng Qu, Alex Lamb, Yoshua Bengio, Juho Kannala, and Jian Tang. 2019. GraphMix: Regularized Training of Graph Neural Networks for Semi-Supervised Learning. *arXiv preprint arXiv:1909.11715* (2019).
[29] Xiaoyun Wang, Joe Eaton, Cho-Jui Hsieh, and Felix Wu. 2018. Attack Graph Convolutional Networks by Adding Fake Nodes. *arXiv preprint arXiv:1810.10751* (2018).
[30] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S Yu. 2019. A Comprehensive Survey on Graph Neural Networks. *arXiv preprint arXiv:1901.00596* (2019).
[31] Qizhe Xie, Zihang Dai, Eduard Hovy, Minh-Thang Luong, and Quoc V Le. 2019. Unsupervised Data Augmentation. *arXiv preprint arXiv:1904.12848* (2019).
[32] Zhilin Yang, William W Cohen, and Ruslan Salakhutdinov. 2016. Revisiting semi-supervised learning with graph embeddings. *arXiv preprint arXiv:1603.08861* (2016).
[33] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. 2017. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412* (2017).
[34] Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. 2017. Random erasing data augmentation. *arXiv preprint arXiv:1708.04896* (2017).

A MORE DETAILS ABOUT EXPERIMENTS FOR REPRODUCIBILITY

In this section, we describe more detailed settings about the experiments to help in reproducibility.

A.1 Datasets and Software Versions

We download the Cora, Citeseer, and Pubmed datasets from the website[1], and the Coauthor-CS and Coauthor-Physics datasets from the website[2]. The latter two datasets are based on the Microsoft Academic Graph from the KDD Cup 2016 challenge, although the original link of the datasets is currently invalid[3]. For all the datasets, we follow the transductive setting, i.e., we use the attributes of all the nodes and the labels of only the nodes in the training set for the training.

Regarding software versions, we install CUDA 10.0 and cuDNN 7.0. TensorFlow 1.12.0 and PyTorch 1.0.0 with Python 3.6.0 are used. Note that all the experiments are running a Linux Server with the Intel(R) Xeon(R) E5-1650 v4 @ 3.60GHz CPU, and the GeForce GTX 1080 Ti GPU.

A.2 Settings of the Baseline

GCN. When implementing GCN as the benchmark and implementing it with NodeAug, we utilize the early stopping training strategy: stop optimization if the validation loss is larger than the mean of validation losses of the last 10 epochs. We set the maximum number of epochs as 200. The number of layers is 2. We apply the dropout on the semantic representations of the first layer. The dropout ratio is 0.5. The number of hidden units is 16. The ℓ_2 regularization weight is 5×10^{-4}. The Adam SGD optimizer is used. The Glorot initialization is used for all the variables. The row normalization is applied to the node attributes before being fed into the model. For the GCN as a benchmark, we use the full-batch training method, with the learning rate of 0.01. For the GCN with NodeAug, we use our mini-batch training, with the learning rate being linear scaled to the batch size, given that the learning rate of the full batch is 0.01.

GAT. When implementing GAT as the benchmark and implementing it with NodeAug, we utilize the early stopping training strategy: stop optimization if neither the validation loss nor the validation accuracy improves for 100 epochs. We set the maximum number of epochs as 100000. The number of layers is 2. We apply the dropout on the inputs of both layers. The dropout ratio is 0.6. The number of attention heads is 8, while the number of features per head is 8. The ℓ_2 regularization weight is 5×10^{-4} for Cora and Citeseer, and 1×10^{-4} for the other datasets. The Adam SGD optimizer is used. The Glorot initialization is used for all the variables. The row normalization is applied to the node attributes before being fed into the model. For the GAT as a benchmark, we use the full-batch training method, with the learning rate of 0.005 for the Cora and Citeseer datasets, and 0.01 for the other datasets. For the GAT with NodeAug, we use our mini-batch training, with the learning rate being linear scaled to the batch size, given that the learning rate of the full batch is 0.005 for the Cora and Citeseer datasets, and 0.01 for the other datasets.

LGCN. When implementing LGCN as the benchmark and implementing it with NodeAug, we utilize the early stopping training strategy: stop optimization if the validation accuracy does not improve for 100 epochs. We set the maximum number of epochs as 1000. A GCN layer is used as the graph embedding layer. The dimension of the embedding output is 32. After that, 2 LGCL layers are stacked for Cora, and 1 LGCL layer is used for Citeseer, Pubmed. We apply the dropout on both input feature vectors and adjacency matrices in each layer with the dropout rates of 0.16 and 0.999. The ℓ_2 regularization weight is 5×10^{-4}. The Adam SGD optimizer is used. The Glorot initialization is used for all the variables. The row normalization is applied to the node attributes before being fed into the model. For the LGCN as a benchmark, we use the full-batch training method, with the learning rate of 0.1 for all the datasets. For the LGCN with NodeAug, we use our mini-batch training, with the learning rate being linear scaled to the batch size, given that the learning rate of the full batch is 0.1 for all the datasets.

Graph U-Net. When implementing Graph U-Net as the benchmark and implementing it with NodeAug, we utilize the early stopping training strategy: stop optimization if the validation accuracy does not improve for 100 epochs. We set the maximum number of epochs as 1000. Four graph pooling with GCN blocks is stacked. We apply the dropout on both adjacency matrix and feature matrices in each layer with the dropout rates of 0.8 and 0.08. The ℓ_2 regularization weight is 1×10^{-3}. The Adam SGD optimizer is used. The Glorot initialization is used for all the variables. The row normalization is applied to the node attributes before being fed into the model. The trick on adding self-loops to each node is used for the adjacency matrix. For Graph U-Net as a benchmark, we use the full-batch training method, with the learning rate of 0.1 for all the datasets. For the LGCN with NodeAug, we use our mini-batch training, with the learning rate being linear scaled to the batch size, given that the learning rate of the full batch is 0.1 for all the datasets.

GMNN. When implementing GMNN as the benchmark, we set the maximum number of epochs as 100. Two GCN layers are stacked with the number of hidden units to be 16. We apply the dropout on the network inputs with the dropout rates of 0.5. The ℓ_2 regularization weight is 1×10^{-3}. The RMSProp optimizer is used with weight decay as 5×10^{-4} The Glorot initialization is used for all the variables. The row normalization is applied to the node attributes before being fed into the model. We reweight each attribute of nodes to 1 if the original weight is greater than 0. We use the full-batch training method, with the learning rate of 0.05 for all the datasets.

GraphVAT. When implementing GraphVAT as the benchmark, we utilize the early stopping training strategy: stop optimization if the validation loss is larger than the mean of validation losses of the last 10 epochs. We set the maximum number of epochs as 200. The number of layers is 2. We apply the dropout on the semantic representations of the first layer. The dropout ratio is 0.5. The number of hidden units is 16. The ℓ_2 regularization weight is 5×10^{-4}. The Adam SGD optimizer is used. The Glorot initialization is used for all the variables. The row normalization is applied to the node attributes before being fed into the model. We use the full-batch training method, with a learning rate of 0.01. The hyper-parameter ϵ for the scale of virtual perturbations is searched in

[1]https://linqs.soe.ucsc.edu/data
[2]https://github.com/shchur/gnn-benchmark
[3]https://kddcup2016.azurewebsites.net/

[0.01, 0.05, 0.1, 0.5, 1]; α for the weight of the virtual adversarial regularizer is searched in [0.001, 0.005, 0.01, 0.05, 0.1, 0.5]; ξ for the scale to calculate approximation is in $[1 \times 10^{-6}, 1 \times 10^{-5}, 1 \times 10^{-4}]$.

GraphMix. When implementing GraphMix as the benchmark, we set the temperature T in sharpening as 0.1, and the number of random perturbations K as 10. We conduct minimal hyperparameter search for α for the Beta distribution used in Manifold Mixup training of the MLP, and the max-consistency coefficient γ which controls the trade-off between the supervised loss and the unsupervised loss. When implementing GraphMix with GCN, we utilize the early stopping training strategy: stop optimization if the validation loss is larger than the mean of validation losses of the last 10 epochs. We set the maximum number of epochs as 200. The number of layers is 2. We apply the dropout on the semantic representations of the first layer. The dropout ratio is 0.5. The number of hidden units is 16. The ℓ_2 regularization weight is 5×10^{-4}. The Adam SGD optimizer is used. The Glorot initialization is used for all the variables. When implementing GraphMix with GAT, we utilize the early stopping training strategy: stop optimization if neither the validation loss nor the validation accuracy improves for 100 epochs. We set the maximum number of epochs as 100000. The number of layers is 2. We apply the dropout on the inputs of both layers. The dropout ratio is 0.6. The number of attention heads is 8, while the number of features per head is 8. The ℓ_2 regularization weight is 5×10^{-4} for Cora and Citeseer, and 1×10^{-4} for the other datasets.

We refer to the following websites when implementing the above mentioned models:

(1) **GCN**: https://github.com/tkipf/gcn
(2) **GAT**: https://github.com/PetarV-/GAT
(3) **LGCN**: https://github.com/HongyangGao/LGCN
(4) **Graph U-Net**: https://github.com/HongyangGao/Graph-U-Nets
(5) **GraphVAT**: https://github.com/fulifeng/GraphAT
(6) **GraphMix**: https://github.com/vikasverma1077/GraphMix