Neural Processing Letters
https://doi.org/10.1007/s11063-022-10750-8

How Does Bayesian Noisy Self-Supervision Defend Graph Convolutional Networks?

Jun Zhuang[1] · **Mohammad Al Hasan**[1]

Accepted: 17 January 2022

Abstract
In recent years, it has been shown that, compared to other contemporary machine learning models, graph convolutional networks (GCNs) achieve superior performance on the node classification task. However, two potential issues threaten the robustness of GCNs, label scarcity and adversarial attacks. .Intensive studies aim to strengthen the robustness of GCNs from three perspectives, the self-supervision-based method, the adversarial-based method, and the detection-based method. Yet, all of the above-mentioned methods can barely handle both issues simultaneously. In this paper, we hypothesize noisy supervision as a kind of self-supervised learning method and then propose a novel Bayesian graph noisy self-supervision model, namely GraphNS, to address both issues. Extensive experiments demonstrate that GraphNS can significantly enhance node classification against both label scarcity and adversarial attacks. This enhancement proves to be generalized over four classic GCNs and is superior to the competing methods across six public graph datasets.

Keywords Defense of graph convolutional networks · Node classification · Bayesian inference · Noisy Supervision

1 Introduction

By Generalizing convolutional operation to graph-structured data, Graph Convolutional Networks (GCNs) achieve superior performance on the node classification task [7,24,48]. However, two potential issues threaten the robustness of GCNs, label scarcity and adversarial attacks. On the one hand, lacking annotated labels may undermine the training of robust GCNs [16]. On the other hand, GCNs may be vulnerable to adversarial attacks [3], i.e. small well-designed perturbations may lead to dramatic degradation in a GCN's performance.

To improve the robustness of GCNs on node classification, intensive studies have been pursued to defend GCNs. Such efforts can be categorized into three groups: self-

✉ Jun Zhuang
junz@iu.edu

Mohammad Al Hasan
alhasan@iupui.edu

[1] Indiana University-Purdue University Indianapolis, Indianapolis, IN, USA

Published online: 08 February 2022

supervision-based method [15,46,52,54], adversarial-based method [5,9,21], and detection-based method [45,56,58]. The adversarial-based methods improve the robustness of GCNs by training with adversarial samples, whereas the detection-based methods detect the attacker nodes or edges and then alleviate the negative impact by removing them. However, neither of them can handle both the label scarcity and adversarial attacks issues simultaneously. An increased number of recent studies that focus on GCN's robustness are employing a kind of self-supervision-based method, namely, pre-training [11,18], to overcome the label scarcity. This approach constructs a pretext task to help GCNs learn transferable graph representation through the pre-training stage. Nevertheless, it is still a challenge to design a more generalized pretext task that can be beneficial to the downstream tasks. Self-training [37] is an extended form of pre-training. It assigns self-generated pseudo-labels to highly confident unlabeled nodes and then adds up these to the labeled nodes for the next iteration. Nonetheless, the accuracy of such pseudo-labels may suffer serious degradation when the pre-trained GCN is under perturbation.

Learning graph representation with the noisy label is a potential solution to mitigate both aforementioned issues, label scarcity and adversarial attacks [6]. In this paper, we argue that the annotated noisy labels assigned to the vertices could be regarded as a kind of self-information for each node. Thus, noisy supervision can be generalized as noisy self-supervision, a subset of self-supervised learning methods. In this paper, our interest is to find answers to the following questions:

Q1: Can a GCN-based node classifier benefit from noisy self-supervision on a label-scarce graph?
Q2: Can noisy self-supervision defend a GCN-based node classifier from adversarial attacks?

To effectively answer the above-mentioned questions, we propose a novel Bayesian **Graph** **N**oisy **S**elf-supervision model, named as **GraphNS**, which improves the robustness of a GCN-based node classifier. In the beginning, GraphNS trains the node classifier with noisy labels on the train graph. For each node, our model returns a multinomial distribution over the label set, which is updated by using a conditional label transition matrix. GraphNS then infers the label for each node by sampling from the updated multinomial distribution. The inference expects that inferred labels would match with the corresponding latent labels. In each iteration, the conditional label transition matrix is dynamically updated by replacing the predicted labels with the inferred labels from Gibbs sampling, and the node classifier is retrained. With each subsequent iterations, the prediction from the node classifier increasingly aligns with the latent labels, making the node classification more robust. Note that, throughout the whole process, the latent labels are unknown, yet GraphNS employs Bayesian inference to approximate the latent labels over conditional label transition. This is made possible by considering that the conditional label transition vector (a multinomial distribution) of each of the K labels follows a Dirichlet prior, and these vectors are iteratively updated using the Bayesian framework. The above design of GraphNS provides an affirmative answer to the first question, as it shows that GraphNS can utilize self-information, i.e., annotated noisy labels of the nodes, to build a robust node classifier (without knowing the ground-truth latent labels) and thereby can be used for node prediction in a label-scarce graph. For a likewise answer to the second question, in the experiment, we will show that GraphNS can repair the prediction of a node classifier when the graph is under adversarial attacks.

The contribution of this paper can be summarized as follows:

- We first generalize noisy supervision as a subset of self-supervised learning methods. This generalization offers an innovative path towards the defense of GCNs.

- We propose a new Bayesian graph noisy self-supervision model, namely GraphNS, to improve the robustness of the node classifier on graph data. To the best of our knowledge, our work is the first model that adapts Bayesian inference with noisy self-supervision on graph data and significantly improves the robustness of GCNs against both label scarcity and adversarial attacks.
- Extensive experiments demonstrate that GraphNS can enhance node classification under both non-attacked and attacked environments. This enhancement proves to be generalized over four classic GCNs and is superior to the competing methods across six public graph datasets.

2 Related Work

2.1 Graph Convolutional Networks

Graph convolutional networks (GCNs) generalize conventional convolutional neural networks (CNNs) to graph data and achieve great success in the recent few years [47,48]. Bruna et al. [2] first generalize convolutions on graph data from the perspective of both spatial method and spectral method. However, the eigendecomposition of the graph Laplacian matrix leads to high complexity on a large graph. To improve efficiency, [4] introduce K-order Chebyshev polynomial to approximate spectral filter. [24] limit the graph convolution to 1-order polynomial and achieve state-of-the-art performance with high efficiency. Wu et al. [44] further simplify the graph convolution by successively removing nonlinearities and collapsing weight between consecutive layers without a negative impact on accuracy. Different from the aforementioned spectral methods, [14] propose a spatial model that aggregates features from fixed-size local neighbors of the current node. Veličković et al. [41] leverage a masked self-attention strategy to aggregate neighborhoods' information in both transductive and inductive manners. All aforesaid models improve the performance from the view of the model structure.

2.2 Defense of GCNs on Node Classification

Intensive studies about the defense of GCNs mainly focus on node classification task [12,22,25,39]. Inspired by [38], we divide most existing defending methods into three main categories, the self-supervised-based method, which utilizes self-information to help enhance the robustness of GCNs [20,32,40,51], the adversarial-based method, which improves the robustness of GCNs by training with generated adversarial samples [42,49,55], and the detection-based method, which aims to mitigate the negative impact of attacks by detecting and removing potential attacker nodes or edges [45,56,58]. Within the first category, pre-training [19,29,34] is a popular approach to mitigate the label scarcity. The approach constructs a pretext task to help GCNs learn transferable graph representation and then fine-tunes with the targeted task. Self-training [37] is an extended form of pre-training. It assigns pseudo-labels to highly confident unlabeled nodes and then adds up these to the labeled nodes for the next iteration. Nonetheless, all of the above-mentioned defending methods can barely handle both label scarcity and adversarial attack simultaneously.

2.3 Learning with Noisy Label

In the past few years, an increasing number of studies learns the deep learning networks with noisy labels [13,28,31]. Sukhbaatar et al. [36] introduce an extra noise transition matrix to adjust the network's output by noisy supervision. Subsequent improvement from [26] considers noise, not only conditioning on the input image but also conditioning on the human annotation. [50] propose a dynamic label regression framework that improves the prediction by embedding the noise transition into Dirichlet-distributed space. Those works achieve great improvement in conventional CNNs. From the perspective of GCNs, [27] present a loss correction approach to handle the graph noisy label. [59] employ GCNs as a label noise cleaner to acquire clean labels. Both works attempt to filter noises and then acquire cleaner labels. Inspired by [50], we propose an innovative Bayesian graph noisy self-supervision model to improve the robustness of the node classifier on graph data without utilizing adversarial samples or identifying perturbators.

3 Methodology

In this section, we introduce the methodology of our proposed model, Bayesian graph noisy self-supervision model, GraphNS, as follows. Section 3.1 introduces the notation and preliminary background. Section 3.2 theoretically analyzes the Bayesian noisy self-supervision. Section 3.3 explains our algorithm and analyzes its time complexity.

3.1 Notations and Preliminaries

Given an undirected attributed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{v_1, v_2, \ldots, v_N\}$ denotes the set of vertices, N is the number of vertices in $\mathcal{G}$, and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ denotes the set of edges between vertices. We denote $\mathbf{A} \in \mathbb{R}^{N \times N}$ as symmetric adjacency matrix and $\mathbf{X} \in \mathbb{R}^{N \times d}$ as the feature matrix, where d is the number of features for each vertex. We define the label-scarce graph as an extreme case in which all ground-truth labels, hereby referred to as **latent labels** of the vertices $\mathbf{Z} \in \mathbb{R}^{N \times 1}$, are unobserved. We argue that manual annotation is a potential solution to this problem but human annotation unavoidably brings into noises [26]. Another potential solution is to use a secondary or weak classifier to label the vertices with pseudo-labels. This solution also yields noisy labels; furthermore, such a solution is vulnerable to adversarial attacks. We use $\mathbf{Y} \in \mathbb{R}^{N \times 1}$ to denote the manual-annotated (or auto-generated) **noisy labels**, which are observed for all nodes (train and test). Our task is to defend GCN-based node classification when its noisy labels (observed) deviate from its latent labels (unobserved). However, we assume that the entries of both $\mathbf{Y}$ and $\mathbf{Z}$ take values from the same closed category set. Below, we first discuss the variant of graph convolutional networks (GCNs) that we consider for our task.

The most representative GCN proposed by [24] is our preferred GCNs' variant. The layer-wise propagation of this GCN is presented as follows:

$$\mathbf{H}^{(l+1)} = \sigma\left(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{H}^{(l)}\mathbf{W}^{(l)}\right) \tag{1}$$

In Equation (1), $\tilde{\mathbf{A}} = \mathbf{A} + I_N, \tilde{\mathbf{D}} = \mathbf{D} + I_N$, where I_N is the identity matrix and $\mathbf{D}_{i,i} = \sum_j \mathbf{A}_{i,j}$ is the diagonal degree matrix. $\mathbf{H}^{(l)} \in \mathbb{R}^{N \times d}$ is the nodes hidden representation in the l-th

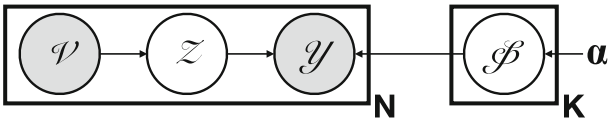

Fig. 1 The diagram of Bayesian noisy self-supervision ($\mathcal{V}$, $\mathbf{Z}$ and $\mathbf{Y}$ denote the nodes, the latent labels, and the noisy labels, respectively. ϕ denotes the conditional label transition matrix. α denotes the Dirichlet parameter. N and K denote the number of nodes and classes, respectively.)

layer, where $\mathbf{H}^{(0)} = \mathbf{X}$. $\mathbf{W}^{(l)}$ is the weight matrix in the l-th layer. $\sigma(\cdot)$ denotes a non-linear activation function, such as ReLU.

In the K-class node classification task, we denote $\mathbf{z}_n$ as the latent label of node v_n and $\mathbf{y}_n$ as the corresponding noisy label ($\mathbf{y}_{nk}$ is a one-hot vector representation of $\mathbf{y}_n$). In our setting, the GCN can be trained by the following loss function $\mathcal{L}$:

$$\mathcal{L} = -\frac{1}{N}\sum_{n=1}^{N}\mathcal{L}(\mathbf{y}_n, f_\theta(v_n)) = -\frac{1}{NK}\sum_{n=1}^{N}\sum_{k=1}^{K}\mathbf{y}_{nk}\ln f_\theta(v_{nk}) \tag{2}$$

where $f_\theta(\cdot) = softmax(\mathbf{H}^{(l)})$ is the prediction of node classifier parameterized by θ.

Our idea to build a robust GCN-based node classifier is to use label transition, in which a transition matrix of size $K \times K$ is learned, which reflects a mapping from $\mathbf{Y}$ to $\mathbf{Z}$. We represent such a matrix by ϕ (the same term is also used to denote a label to label mapping function). Under the presence of ϕ, the loss function $\mathcal{L}$ in Equation (2) can be rewritten as follows:

$$\mathcal{L} = -\frac{1}{N}\sum_{n=1}^{N}\mathcal{L}(\mathbf{y}_n, \phi^{-1} \circ f_\theta(v_n)) \tag{3}$$

However, learning accurate ϕ and f_θ is a difficult task as the latent labels of the nodes are not available. So, in our method, the ϕ and f_θ are iteratively updated to approximate the perfect one by using the Bayesian framework.

3.2 Bayesian Noisy Self-supervision

As shown in the plate diagram of Bayesian noisy self-supervision (Fig. 1), the unobserved latent labels ($\mathbf{Z}$) depend on the node features, whereas the observed noisy labels ($\mathbf{Y}$) depend on both $\mathbf{Z}$ and the conditional label transition matrix, ϕ, modeled by K multinomial distributions with a Dirichlet prior parameterized by α.

The latent label of node v_n, $\mathbf{z}_n \sim P(\cdot \mid v_n)$, where $P(\cdot \mid v_n)$ is a $Categorical$ distribution modeled by the node classifier $f_\theta(v_n)$. The noisy label $\mathbf{y}_n \sim P\left(\cdot \mid \phi_{\mathbf{z}_n}\right)$, where $\phi_{\mathbf{z}_n}$ is the parameter of $Categorical$ distribution $P\left(\cdot \mid \phi_{\mathbf{z}_n}\right)$. The conditional label transition matrix $\phi = [\phi_1, \phi_2, , \phi_K]^T \in \mathbb{R}^{K\times K}$ consists of K transition vectors. The k-th transition vector $\phi_k \sim Dirichlet(\alpha)$, where α is the parameter of $Dirichlet$ distribution. The goal of our label transition is to obtain an updated $P(\cdot \mid v_n)$ by using Bayesian noisy self-supervision so that the **inferred label** of given node sampled from this updated distribution is identical to the latent label of that node as much as possible.

According to Fig. 1, the dependency of latent labels can be formulated as follows:

$$P(\mathbf{Z} \mid \mathcal{V}, \mathbf{Y}; \alpha) = P(\mathbf{Z} \mid \mathcal{V}, \mathbf{Y}, \phi)\, P(\phi; \alpha) \tag{4}$$

where the posterior of $\mathbf{Z}$ which is conditioned on the nodes $\mathcal{V}$, the noisy labels $\mathbf{Y}$, and the Dirichlet parameter α.

The posterior of $\mathbf{Z}$ can be deduced by Bayes' theorem as follows:

$$\begin{aligned} P(\mathbf{Z} \mid \mathcal{V}, \mathbf{Y}; \alpha) &= \int_{\phi} \prod_{k=1}^{K} P(\phi_k; \alpha) \prod_{n=1}^{N} P(\mathbf{z}_n \mid v_n, \mathbf{y}_n, \phi)\, d\phi \\ &= \int_{\phi} \prod_{k=1}^{K} P(\phi_k; \alpha) \prod_{n=1}^{N} \frac{P(\mathbf{z}_n \mid v_n)\, P(\mathbf{y}_n \mid \mathbf{z}_n, \phi)}{P(\mathbf{y}_n \mid v_n)} d\phi \end{aligned} \tag{5}$$

We assume the Dirichlet distribution is symmetric, i.e., the parameter vector α has the same value to all elements. Thus, Equation (5) can be further deduced as follows:

$$\begin{aligned} P(\mathbf{Z} \mid \mathcal{V}, \mathbf{Y}; \alpha) &= \int_{\phi} \prod_{k=1}^{K} \frac{\Gamma\left(\sum_{k'}^{K} \alpha_{k'}\right)}{\prod_{k'}^{K} \Gamma(\alpha_{k'})} \prod_{k'}^{K} \phi_{kk'}^{\alpha_{k'}-1} \prod_{n=1}^{N} \frac{P(\mathbf{z}_n \mid v_n)}{P(\mathbf{y}_n \mid v_n)} \prod_{n=1}^{N} \phi_{\mathbf{z}_n \mathbf{y}_n} d\phi \\ &= \prod_{n=1}^{N} \frac{P(\mathbf{z}_n \mid v_n)}{P(\mathbf{y}_n \mid v_n)} \int_{\phi} \prod_{k=1}^{K} \frac{\Gamma\left(\sum_{k'}^{K} \alpha_{k'}\right)}{\prod_{k'}^{K} \Gamma(\alpha_{k'})} \prod_{k'}^{K} \phi_{kk'}^{\alpha_{k'}-1} \prod_{n=1}^{N} \phi_{\mathbf{z}_n \mathbf{y}_n} d\phi \end{aligned} \tag{6}$$

where the term $\prod_{n=1}^{N} \frac{P(\mathbf{z}_n \mid v_n)}{P(\mathbf{y}_n \mid v_n)}$ is constant w.r.t. ϕ, and hence we take it out of the integration. We simplify this term as CST in the following deduction.

According to the conjugation property between the Multinomial distribution and the Dirichlet distribution, Equation (6) can be deduced as follows:

$$\begin{aligned} P(\mathbf{Z} \mid \mathcal{V}, \mathbf{Y}; \alpha) &= CST \int_{\phi} \prod_{k=1}^{K} \frac{\Gamma\left(\sum_{k'}^{K} \alpha_{k'}\right)}{\prod_{k'}^{K} \Gamma(\alpha_{k'})} \prod_{k'}^{K} \phi_{kk'}^{\mathbf{C}_{kk'}+\alpha_{k'}-1} d\phi \\ &= CST \prod_{k=1}^{K} \frac{\Gamma\left(\sum_{k'}^{K} \alpha_{k'}\right)}{\prod_{k'}^{K} \Gamma(\alpha_{k'})} \prod_{k=1}^{K} \frac{\prod_{k'}^{K} \Gamma(\alpha_{k'} + \mathbf{C}_{kk'})}{\Gamma\left(\sum_{k'}^{K} (\alpha_{k'} + \mathbf{C}_{kk'})\right)} \end{aligned} \tag{7}$$

Here we denote the confusion matrix between the node prediction and the noisy labels as $\mathbf{C}$, where $\sum_{k}^{K} \sum_{k'}^{K} \mathbf{C}_{kk'} = N$. The term $\prod_{n=1}^{N} \phi_{\mathbf{z}_n \mathbf{y}_n}$ is expressed as $\prod_{k}^{K} \prod_{k'}^{K} \phi_{kk'}^{\mathbf{C}_{kk'}}$ so that we can integrate the terms based on the aforementioned conjugation property.

Unfortunately, Eq. (7) can not directly be employed to infer the label. Instead, we apply Gibbs sampling here to approximate our goal. According to Gibbs sampling, for each time we sample $\mathbf{z}_n$ by fixing n-th dimension in order to satisfy the detailed balance condition on the assumption of Markov chain [1]. Combined with Equation (7) and the recurrence relation of Γ function, $\Gamma(n+1) = n\Gamma(n)$, we sample a sequence of $\mathbf{z}_n$ as follows:

$$\begin{aligned} P\left(\mathbf{z}_n \mid \mathbf{Z}^{\neg \mathbf{z}_n}, \mathcal{V}, \mathbf{Y}; \alpha\right) &= \frac{P(\mathbf{Z} \mid \mathcal{V}, \mathbf{Y}; \alpha)}{P\left(\mathbf{Z}^{\neg \mathbf{z}_n} \mid \mathcal{V}, \mathbf{Y}; \alpha\right)} \\ &= \frac{P(\mathbf{z}_n \mid v_n)}{P(\mathbf{y}_n \mid v_n)} \frac{\alpha_{\mathbf{y}_n} + \mathbf{C}_{\mathbf{z}_n \mathbf{y}_n}^{\neg \mathbf{z}_n}}{\sum_{k'}^{K} \left(\alpha_{k'} + \mathbf{C}_{\mathbf{z}_n k'}^{\neg \mathbf{z}_n}\right)} \\ &\propto \overline{P}(\mathbf{z}_n \mid v_n) \frac{\alpha_{\mathbf{y}_n} + \mathbf{C}_{\mathbf{z}_n \mathbf{y}_n}^{\neg \mathbf{z}_n}}{\sum_{k'}^{K} \left(\alpha_{k'} + \mathbf{C}_{\mathbf{z}_n k'}^{\neg \mathbf{z}_n}\right)} \end{aligned} \tag{8}$$

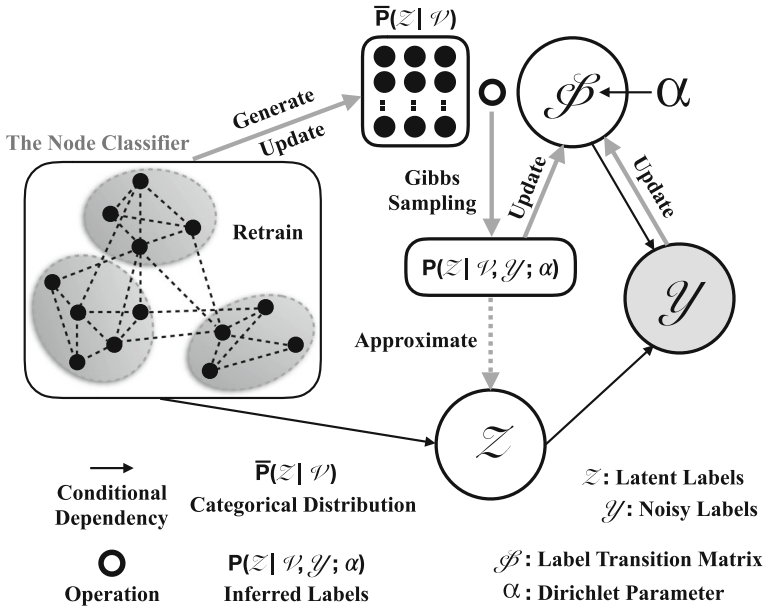

Fig. 2 The workflow of Bayesian graph noisy self-supervision (GraphNS) (The latent labels represent the ground-truth labels, which cannot be observed (White color). The noisy labels are manually annotated, which can be observed (Gray color).)

where we denote $\mathbf{Z}^{\neg \mathbf{z}_n}$ as the subset of $\mathbf{Z}$ that removes statistic $\mathbf{z}_n$. In the last row of Eq. (8), the first term $\overline{P}(\mathbf{z}_n \mid v_n)$ is a categorical distribution of labels for the node v_n modeled by f_θ. We use the term $\overline{P}(\mathbf{Z} \mid \mathcal{V}) \in \mathbb{R}^{N \times K}$ to denote the same over all the nodes. Note that $P(\mathbf{Z} \mid \mathcal{V}) \in \mathbb{R}^{N \times 1}$ in this paper denotes the predicted labels. Whereas the second term represents the conditional label transition which is obtained from the posterior of the multinomial distribution corresponding to label transition from $\mathbf{y}_n$ to $\mathbf{z}_n$. We use Eq. (8) to sample the inferred label, $\mathbf{z}_n$, which becomes the node v_n's label for retraining f_θ. Also, ϕ is updated through Bayesian inference in each iteration. Such process is repeated for a given number of epochs with the expectation that subsequent inferred label can approximate to the latent label.

3.3 GraphNS Algorithm and Pseudo-code

The total process of GraphNS is displayed in Fig. 2. Given an undirected attribute graph, GraphNS classifies the nodes and generates categorical distribution $\overline{P}(\mathbf{Z} \mid \mathcal{V}) \in \mathbb{R}^{N \times K}$ at first. After that, GraphNS applies Gibbs sampling to sample the inferred labels $P(\mathbf{Z} \mid \mathcal{V}, \mathbf{Y}; \alpha) \in \mathbb{R}^{N \times 1}$ and updates the label transition matrix ϕ parameterized by α. The information of $\mathcal{V}$ is represented by both $\mathbf{A}$ and $\mathbf{X}$. Simultaneously, the node classifier is iteratively re-trained to update $\overline{P}(\mathbf{Z} \mid \mathcal{V})$. The inference will ultimately converge, approximating the inferred labels $P(\mathbf{Z} \mid \mathcal{V}, \mathbf{Y}; \alpha)$ to the latent labels $\mathbf{Z}$ as close as possible. In brief, the goal of GraphNS is to sample the inferred labels by supervising the categorical distribution based on dynamic conditional label transition and ultimately approximates the inferred labels to the latent labels as close as possible.

Algorithm 1: GRAPH Noisy Self-supervision

Input: Graph $\mathcal{G}_{train}$ and $\mathcal{G}_{test}$, which contain corresponding symmetric adjacency matrix $\mathbf{A}$, feature matrix $\mathbf{X}$ and noisy labels $\mathbf{Y}$, Node classifier f_θ, The number of warm-up steps WS, The number of epochs for inference $Epochs$

1 Train f_θ by Equation (2) on $\mathcal{G}_{train}$;
2 Generate categorical distribution $\overline{P}(\mathbf{Z} \mid \mathcal{V})$ by f_θ;
3 Compute the warm-up label transition matrix ϕ' based on $\mathcal{G}_{train}$;
4 Define the inferred labels $P(\mathbf{Z} \mid \mathcal{V}, \mathbf{Y}; \alpha)$ and the dynamic label transition matrix ϕ based on $\mathcal{G}_{test}$;
5 **for** $step \leftarrow 1$ ***to*** $Epochs$ **do**
6 **if** $step < WS$ **then**
7 Sample $\mathbf{z}_n$ with warm-up ϕ' by Eq. (8);
8 **end**
9 **else**
10 Sample $\mathbf{z}_n$ with dynamic ϕ by Eq. (8);
11 **end**
12 Update dynamic ϕ, $P(\mathbf{Z} \mid \mathcal{V}, \mathbf{Y}; \alpha)$, and retrain f_θ;
13 **end**
14 **return** $P(\mathbf{Z} \mid \mathcal{V}, \mathbf{Y}; \alpha)$ *and Dynamic* ϕ*;*

The pseudo-code of GraphNS is shown in Algorithm (1). **Training:** GraphNS trains the node classifier f_θ on the train graph $\mathcal{G}_{train}$ at first (**line 1**) and then generates categorical distribution $\overline{P}(\mathbf{Z} \mid \mathcal{V})$ by f_θ (**line 2**). **Inference:** Before the inference, GraphNS first computes a warm-up label transition matrix ϕ' by using the prediction over the train graph (**line 3**) and then defines (creates empty spaces) the inferred labels $P(\mathbf{Z} \mid \mathcal{V}, \mathbf{Y}; \alpha)$ and the dynamic label transition matrix ϕ based on the test graph $\mathcal{G}_{test}$ (**line 4**). In the warm-up stage of the inference, GraphNS samples $\mathbf{z}_n$ with the warm-up label transition matrix ϕ' (**line 7**), which is built with the categorical distribution of f_θ and the noisy labels on the train graph. The categorical distributions of both the train graph and the test graph should have high similarity if both follow a similar distribution. Thus, the warm-up ϕ' is a keystone since subsequent inference largely depends on this distribution. After the warm-up stage, GraphNS samples $\mathbf{z}_n$ with the dynamic ϕ (**line 10**). This dynamic ϕ updates in every epoch with current sampled $\mathbf{z}_n$ and corresponding $\mathbf{y}_n \in \mathbf{Y}$. Simultaneously, the inferred labels $P(\mathbf{Z} \mid \mathcal{V}, \mathbf{Y}; \alpha)$ is also updated based on the before-mentioned $\mathbf{z}_n$ whereas the node classifier is iteratively retrained (**line 12**). The inference will ultimately converge, approximating the inferred labels to the latent labels as close as possible. Note that both the train graph and the test graph contain corresponding symmetric adjacency matrix $\mathbf{A}$, feature matrix $\mathbf{X}$, and noisy labels $\mathbf{Y}$. The categorical distributions of the test graph may change abruptly when this graph is under perturbation since the original distribution in this graph is being perturbed. In this case, GraphNS can also help recover the original categorical distribution by dynamic conditional label transition.

According to Algorithm (1), GraphNS applies Gibbs sampling via Eq. (8) inside the FOR loop. The time complexity of the sampling is $\mathcal{O}(N_{test} \times K + K^2)$ since element-wise multiplication only traverses the number of elements in matrices once, where N_{test} denotes the number of nodes in the test graph. In practice, the number of test nodes is far more than the number of classes, i.e., $N \gg K$. So, the time complexity of this sampling operation is approximately equal to $\mathcal{O}(N_{test})$. Hence, the time complexity of inference except the first training (**line 1**) is $\mathcal{O}(Epochs \times N_{test})$, where $Epochs$ is the number of epochs for inference.

Table 1 Statistics of six public datasets ($|\mathcal{V}|$, $|\mathcal{E}|$, $|F|$, and $|C|$ denote the number of nodes, edges, features, and classes, respectively. Avg.D denotes the average degree of test nodes.)

Dataset	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{E}\rvert$	$\lvert F\rvert$	$\lvert C\rvert$	Avg.D
Cora	2708	10,556	1433	7	3.85
Citeseer	3327	9228	3703	6	2.78
Pubmed	19,717	88,651	500	3	4.49
AMZcobuy	7650	287,326	745	8	32.32
Coauthor	18,333	327,576	6805	15	10.01
Reddit	232,965	114,615,892	602	41	487.48

4 Experiments

In this section, we present and analyze experimental results as follows. At first, we study how different variants of GCNs (spectral and spatial) node classifiers benefit from noisy self-supervision, under different noise concentrations on the label-scarce graph. Furthermore, we perform investigations on how noisy self-supervision defends node classifiers against adversarial attacks. Finally, we analyze parameters and conduct an ablation study on our model.

4.1 Experimental Settings

Our proposed model is evaluated on six datasets in Table 1. **Cora**, **Citeseer**, and **Pubmed** are famous citation graph data [33]. **AMZcobuy** comes from the photo segment of the Amazon co-purchase graph [17,35], where nodes represent the products, whereas edges indicate that two products are frequently bought together. Product reviews are encoded by bag-of-words as the feature. The node label is the product category. **Coauthor** is co-authorship graphs of computer science based on the Microsoft Academic Graph from the KDD Cup 2016 challenge [1]. Each Node represents one author. Two nodes are connected if they are co-authors of one paper. The feature represents all paper keywords for this author. The node label is the most active research area for this author. **Reddit** dataset is constructed by connecting Reddit posts if the same user comments on both posts [14]. The node label is the community that this post belongs to. For each post, its feature vector includes the embedding of title and comments, score, and the number of comments.

For all six datasets, the percentage of train, validation, and test partition comprise 40%, 30%, and 30% of the nodes, respectively. In the training phase, all node classifiers converge within 200 epochs for training. Thus, we set the number of training epochs as 200. To generate noisy labels, we randomly replace the ground-truth label of a node with another label, chosen uniformly. To implement the adversarial attacks, we execute non-targeted node-level direct evasion attacks [61] on the edges and features ($L\&F$) of the victim nodes, whereas the trained node classifier remains unchanged. The attacker randomly selects 20% of the test nodes as victims. Similar to [53], the intensity of perturbation n_{pert} is set to be 2 (for Cora, Citeseer, and Pubmed), 5 (for AMZcobuy and Coauthor), and 10 (for Reddit), respectively. The ratio of n_{pert} between applying on links and applying on features is $1:10$. For example, the attacker applies 2 perturbations on links and 20 perturbations on features for $L\&F$. The noise ratio nr is 0.1 here. We run each experiment five times and present its mean and standard deviation.

[1] https://www.kdd.org/kdd-cup/view/kdd-cup-2016.

To evaluate the defending performance, we compare our proposed model against six popular defending methods and present the hyper-parameters below. For reproducibility, we maintain the same denotation for each competing method as the corresponding original paper. The competing models are trained by Adam optimizer with 200 epochs.

- **AdvTrain** [43,53] assigns pseudo labels to generated adversarial samples and then retrains the node classifier with both noisy labeled nodes and adversarial nodes. To implement this adversarial training method, we first generate the adversarial samples by $L\&F$. The number of adversarial samples is equal to 10% of victim nodes. After that, we add up these generated adversarial samples into the training set and then retrain the node classifier. Finally, we evaluate the defending result of adversarial training by implementing $L\&F$ on the victim nodes. The approach of adversarial training can be formulated as follows:

$$\begin{aligned} \mathcal{L}_{noisy} &= \mathcal{L}(\mathbf{Y}_{noisy},\, f_\theta(\tilde{\mathbf{A}}, \mathbf{X})), \\ \mathcal{L}_{adv} &= \mathcal{L}(\mathbf{Y}_{pseudo},\, f_\theta(\mathbf{A}', \mathbf{X}')), \\ \theta^* &= \arg\min_\theta \left(\mathcal{L}_{noisy} + \mathcal{L}_{adv}\right), \end{aligned} \tag{9}$$

  where $\mathbf{Y}_{noisy}$, $\mathbf{Y}_{pseudo}$ are the corresponding labels to compute loss functions, and $\mathbf{A}'$, $\mathbf{X}'$ are perturbed adjacency, feature matrices generated by the attacking algorithm.
- *G*NN-Jaccard [45] preprocesses the graph by eliminating suspicious connections, whose Jaccard similarity of node's features is smaller than a given threshold. The similarity threshold for dropping edges is 0.01. The number of hidden units is 16. The dropout rate is 0.5.
- *G*NN-SVD [8] proposes another preprocessing approach with low-rank approximation on the perturbed graph to mitigate the negative effects from high-rank attacks, such as Nettack [61]. The number of singular values and vectors is 15. The number of hidden units is 16. The dropout rate is 0.5.
- *G*RAND [10] proposes random propagation and consistency regularization strategies to address the issues of over-smoothing and non-robustness of GCNs. We follow the same procedure to tune the hyper-parameters. The optimal hyper-parameters of GRAND in this paper are reported in Table 2.
- **SelfGNN** [23] introduces a contrastive self-supervised approach for GCNs that leverages Batch Normalization and requires no explicit negative sampling. The number of units for each layer is 512 and 128, respectively. The dropout rate is 0.2. The learning rate is 1e-4. For the GAT model, the number of heads for each layer is 8 and 1, respectively.
- *G*MNN [30] models the joint label distribution with a conditional random field, which can be effectively trained with the variational EM algorithm. We apply RMSprop optimizer on Cora and Citeseer with learning rate lr = 0.05 and alternatively use Adam optimizer on the rest datasets with learning rate lr = 0.01. The decay rate is 5e-4. The number of hidden units is 16. The input dropout rate is 0.5. For the rest of the hyper-parameters, we follow the default settings in the paper [30].

Extra details about reproducibility are described in **Appendix**.

4.2 Node Classification Using GraphNS

Can node classifiers benefit from noisy self-supervision on the label-scarce graph? In this experiment, we study how node classifiers benefit from noisy self-supervision over two groups of popular methods, spectral methods (GCN [24], SGC [44]) and spatial methods

Table 2 Hyper-parameters of GRAND in this paper

Hyper-parameters	Cora	Citeseer	Pubmed	AMZcobuy	Coauthor	Reddit
DropNode probability	0.5	0.5	0.5	0.5	0.5	0.6
Propagation step	8	2	5	5	5	5
Data augmentation times	4	2	4	3	3	4
CR loss coefficient	1.0	0.7	1.0	0.9	0.9	1.0
Sharpening temperature	0.5	0.3	0.2	0.4	0.4	0.5
Learning rate	0.01	0.01	0.2	0.2	0.2	0.2
Early stopping patience	200	200	100	100	100	100
Hidden layer size	32	32	32	32	32	32
L2 weight decay rate	5e-4	5e-4	5e-4	5e-4	5e-4	5e-4
Dropout rate in input layer	0.5	0.0	0.6	0.6	0.6	0.6
Dropout rate in hidden layer	0.5	0.2	0.8	0.5	0.5	0.6

(GraphSAGE [14], GAT [41]), across six public datasets. As presented in Table 3, we compare the test accuracy of these node classifiers between the original performance **Orig.** and the performance after noisy self-supervision **NS** under different noise ratios $nr = [0.5, 0.3, 0.1]$, where the value of noise ratio indicates the fraction of vertices, whose label has been replaced.

As seen in Table 3, as expected, the accuracy increases as the noise ratio decreases. According to the result, all node classifiers benefit from GraphNS, often substantially when the noise ratio is small. However, node classifiers may sometimes be weakened when $(1 - nr)$ is much lower than the original accuracy in some cases. For example, GraphNS is weakened by using the noisy labels ($nr = 0.5$) on Citeseer. We argue that GraphNS iteratively updates the predicted labels with the inferred labels, whereas the inferred labels are obtained from Gibbs sampling based on both the updated multinomial distribution and the noisy labels. However, the accuracy of the inferred labels may get worse if $(1 - nr)$ is much lower than that of the predicted labels. We affirmatively observe that GraphNS can stably achieve superior improvement when $(1 - nr)$ is not lower than the original accuracy. We also observe that GraphSAGE achieves the best classification performance on Cora, Citeseer, and Pubmed whereas GCN achieves that on AMZcobuy, Coauthor, and Reddit. We argue that the difference is caused by the degree of nodes. Compared to GCN, which applies 1-order polynomial on layer-wise graph convolution, GraphSAGE aggregates features from fixed-size local neighbors of the current node. Such aggregation may lose more information when the degree of nodes increases. That is to say, GCN can outperform GraphSAGE when the graph is denser (higher degrees of nodes).

4.3 GraphNS Defends Node Classifiers

Can noisy self-supervision defend node classifiers from adversarial attacks? We investigate how GraphNS defends node classifiers against Nettack [61]. In this investigation, we compare the defending performance of GraphNS against the competing methods and visualize the defending results on Cora, Citeseer, and Pubmed as examples.

In Table 4, we highlight the best performance for each dataset. All node classifiers originally have comparable performance. After the adversarial attacks, the results explicitly state that GraphNS achieves robust defending results on top of all node classifiers across six public

Table 3 Investigation about how node classifiers f_θ benefit from GraphNS under different noise ratios nr (**Orig.** and **NS** denote the accuracy before/after noisy self-supervision, respectively.)

Test Acc (%)	**nr**	Cora		Citeseer		Pubmed		AMZcobuy		Coauthor		Reddit	
		Orig.	NS	Orig.	NS	Orig.	NS	Orig.	NS	Orig.	NS	Orig.	NS
GCN	0.5	75.65	76.26	56.46	58.55	76.64	81.46	90.72	89.15	91.35	73.09	92.58	87.72
	0.3	77.86	87.09	66.17	76.78	80.54	86.43	92.11	95.16	92.25	94.09	92.83	95.01
	0.1	83.03	93.85	69.57	93.09	82.64	94.29	**93.12**	**97.56**	**92.29**	**97.73**	**93.41**	**98.14**
SGC	0.5	78.35	82.78	62.26	59.46	77.84	84.75	85.75	87.76	89.91	69.67	88.76	82.74
	0.3	83.39	88.32	69.37	76.67	82.30	87.53	87.23	93.68	90.47	94.98	89.97	91.82
	0.1	84.87	94.47	70.47	93.49	83.74	93.85	90.20	96.56	91.36	97.41	91.03	96.59
GraphSAGE	0.5	78.60	79.71	62.06	58.56	79.90	86.53	90.28	75.47	90.71	82.62	92.53	93.81
	0.3	81.55	88.68	70.67	76.68	84.11	89.66	91.76	94.86	91.76	82.80	92.78	95.94
	0.1	**85.85**	**94.71**	**73.07**	**93.79**	**85.09**	**96.04**	92.51	97.55	92.28	96.78	92.99	97.48
GAT	0.5	77.12	74.78	61.86	65.26	78.18	83.65	89.59	70.94	90.80	78.82	92.44	93.67
	0.3	80.81	81.67	69.67	81.88	81.32	86.10	92.37	94.68	91.56	94.74	92.57	94.15
	0.1	85.36	94.07	70.67	92.59	82.22	93.93	93.02	97.50	92.01	96.95	93.12	97.61

Table 4 Comparison of the defending results (test accuracy) between our model, GraphNS, and the competing methods (Orig./Attack denote the test accuracy before/after the non-targeted attacks ($L\&F$).)

	Test Acc (%)	Cora	Citeseer	Pubmed	AMZcobuy	Coauthor	Reddit
GCN	Orig.	81.69(±2.87)	70.52(±0.85)	81.93(±0.41)	92.34(±1.06)	91.84(±0.90)	93.72(±0.68)
	Attack	23.46(±1.32)	36.35(±3.54)	57.06(±3.27)	45.25(±3.15)	36.77(±3.21)	49.27(±4.28)
	AdvTrain	44.49(±3.13)	49.22(±1.96)	60.40(±3.11)	58.97(±2.34)	61.44(±1.60)	66.41(±2.30)
	GNN-Jaccard	73.12(±2.40)	69.17(±2.05)	79.32(±1.43)	80.64(±2.49)	67.41(±1.71)	75.35(±1.89)
	GNN-SVD	66.52(±1.91)	64.28(±2.80)	72.54(±1.31)	76.79(±1.68)	63.45(±1.09)	70.63(±3.30)
	GRAND	28.95(±3.86)	61.72(±5.44)	75.39(±1.81)	52.84(±12.08)	64.79(±7.99)	66.99(±6.48)
	SelfGNN	68.95(±1.64)	55.72(±1.77)	69.58(±2.23)	63.25(±1.84)	62.46(±1.37)	78.01(±1.78)
	GMNN	68.39(±3.86)	65.45(±4.04)	81.49(±1.44)	81.59(±3.21)	69.19(±1.46)	77.59(±2.87)
	GraphNS	**81.55(±2.24)**	**70.54(±1.36)**	**91.62(±1.99)**	**88.82(±2.07)**	**72.07(±2.46)**	**86.87(±2.86)**
SGC	Orig.	83.75(±2.59)	69.85(±0.82)	84.75(±1.38)	91.00(±1.23)	90.98(±1.23)	91.76(±0.49)
	Attack	56.99(±1.27)	45.23(±7.69)	4.24(±1.20)	43.00(±5.42)	43.70(±4.98)	40.09(±2.40)
	AdvTrain	64.99(±3.42)	63.42(±2.85)	46.13(±8.54)	61.40(±4.12)	71.32(±4.76)	64.56(±3.46)
	GNN-Jaccard	80.52(±2.49)	78.27(±2.94)	63.86(±2.82)	77.72(±2.15)	75.16(±3.60)	76.21(±1.92)
	GNN-SVD	61.42(±3.08)	69.91(±2.85)	58.05(±2.98)	73.57(±2.46)	68.94(±3.48)	70.19(±2.88)
	GRAND	28.75(±3.90)	69.20(±1.17)	71.64(±7.22)	50.07(±7.23)	65.81(±11.22)	66.21(±6.72)
	SelfGNN	64.77(±1.28)	51.33(±1.81)	67.34(±2.11)	56.15(±2.37)	66.03(±1.65)	78.98(±1.54)
	GMNN	75.42(±2.63)	80.64(±2.95)	68.86(±1.39)	78.46(±1.72)	76.86(±2.64)	78.37(±3.86)
	GraphNS	**84.55(±3.78)**	**86.22(±4.22)**	**71.86(±2.99)**	**86.46(±2.83)**	**83.13(±3.79)**	**83.99(±1.96)**

Table 4 continued

	Test Acc (%)	Cora	Citeseer	Pubmed	AMZcobuy	Coauthor	Reddit
GraphSAGE	Orig.	84.15(±1.16)	72.36(±1.09)	83.05(±3.17)	93.04(±0.79)	91.47(±0.84)	93.12(±0.65)
	Attack	44.03(±7.55)	28.98(±5.19)	3.95(±1.44)	42.57(±6.32)	43.83(±5.02)	51.25(±2.61)
	AdvTrain	48.43(±7.71)	44.49(±2.94)	45.74(±7.74)	56.49(±6.12)	73.56(±3.69)	67.27(±3.48)
	GNN-Jaccard	79.96(±2.57)	76.19(±2.95)	62.25(±2.65)	76.72(±3.15)	74.22(±2.29)	75.62(±2.96)
	GNN-SVD	61.03(±3.28)	63.90(±2.52)	59.73(±2.90)	72.42(±3.31)	68.69(±2.07)	69.61(±2.08)
	GRAND	28.55(±3.57)	68.54(±3.06)	66.39(±10.10)	43.67(±4.18)	63.74(±8.29)	68.51(±3.02)
	SelfGNN	65.43(±1.59)	49.88(±1.96)	64.95(±1.51)	52.12(±1.44)	69.28(±1.47)	80.89(±1.96)
	GMNN	75.10(±2.98)	75.20(±1.88)	66.35(±1.91)	76.17(±1.92)	75.57(±1.40)	77.99(±2.13)
	GraphNS	**84.94(±1.62)**	**79.17(±2.13)**	**69.79(±1.52)**	**85.33(±3.69)**	**84.66(±2.38)**	**85.19(±2.17)**
GAT	Orig.	85.75(±2.60)	72.70(±1.55)	82.48(±2.80)	92.42(±0.56)	91.98(±0.53)	92.29(±0.41)
	Attack	44.45(±2.38)	29.81(±4.11)	5.93(±1.62)	42.96(±3.59)	44.64(±3.61)	52.17(±2.64)
	AdvTrain	65.32(±1.74)	43.86(±3.03)	42.09(±3.15)	53.37(±2.54)	72.23(±3.31)	68.07(±2.38)
	GNN-Jaccard	72.36(±2.39)	72.36(±2.05)	84.73(±1.39)	81.06(±2.82)	78.87(±2.86)	79.73(±3.55)
	GNN-SVD	70.78(±3.79)	60.31(±3.25)	79.43(±4.29)	76.13(±2.83)	71.61(±2.71)	72.63(±2.25)
	GRAND	32.02(±3.62)	61.04(±3.01)	48.52(±4.23)	44.60(±1.20)	62.78(±7.68)	65.67(±4.13)
	SelfGNN	65.37(±1.92)	64.03(±1.68)	70.99(±2.32)	74.87(±2.22)	78.23(±2.24)	81.57(±2.19)
	GMNN	70.58(±3.53)	62.71(±1.07)	84.34(±1.63)	73.94(±2.01)	77.01(±2.20)	79.34(±2.72)
	GraphNS	**86.21(±3.17)**	**76.23(±2.94)**	**87.54(±0.38)**	**85.66(±1.83)**	**85.26(±2.18)**	**86.08(±1.87)**

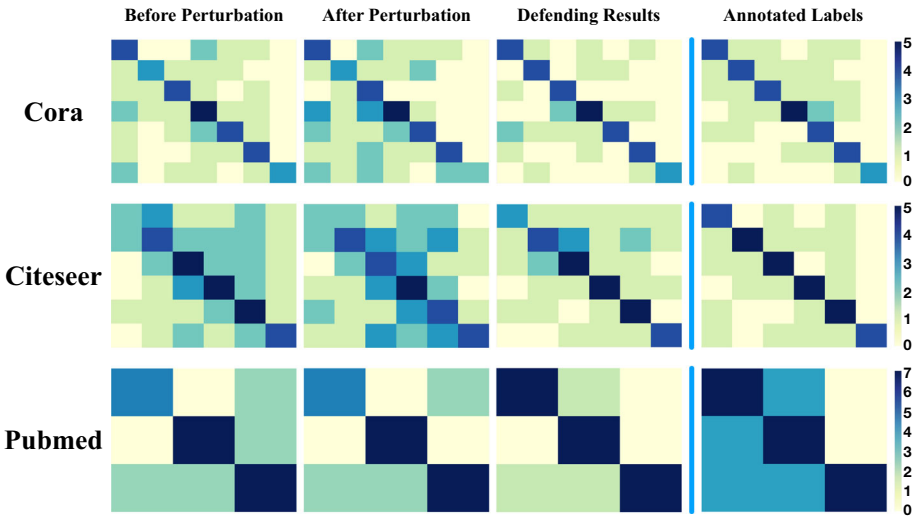

Fig. 3 The confusion matrices (heatmap) of the defending performances on Cora, Citeseer and Pubmed for GraphNS (nr=0.1) (We apply log-scale to the confusion matrix for fine-grained visualization.)

datasets and also gains superior defense against the competing methods. Such defense recovers the performance after perturbations and even exceeds the original performance in some cases. We argue that edges of citation graphs may not be always coherent, that is probably the reason that in the citation graphs GraphSS sometimes outperforms the original performance. The extent of recovery may sometimes be influenced by the attacking consequence, i.e., GraphNS can retrieve higher accuracy when the node classifier undergoes less degradation. Furthermore, we have several observations on the competing methods. 1) AdvTrain and GRAND are two fragile defending methods. For AdvTrain, more serious perturbations lead to a much weaker recovery, which reveals that simply using adversarial samples is insufficient to obtain a robust classifier. For GRAND, its performance is worse (under Nettack [61]) than what was reported in [10] (under Metattack [60]). One of the reasons for this is that GRAND assumes that the network satisfies homophily property, and if the input network does not satisfy such, it may not work well with GRAND. Nettack simultaneously alters the graph structures and features, which may make the network not satisfy homophily property leading to poor performance by GRAND. We also notice that GRAND is sensitive to the parameter's setting as we spent substantial time tuning the optimal parameters on different datasets. On the contrary, GraphNS does not assume such network property, and it is also easy to tune. 2) GNN-SVD presents lower performance compared to GNN-Jaccard across all datasets because GNN-SVD is designed for targeted attacks rather than non-targeted attacks. 3) SelfGNN achieves stable performance with a relatively lower standard deviation across all datasets. We argue that leveraging Batch Normalization and feature augmentation may help recover the accuracy from adversarial perturbations. 4) Both GMNN [30] and Bayesian-GCNN [57] aim to solve semi-supervised classification problems via Bayesian approaches. We select GMNN as the competing method and examine its defending performance. The results successfully verify its capacity of defense.

We also visualize the defending performances for GraphNS (on top of GCN) on Cora, Citeseer, and Pubmed as examples in Fig. 3. The first three columns present the predictions before perturbation, after perturbation, and our defending results by GraphNS. The last

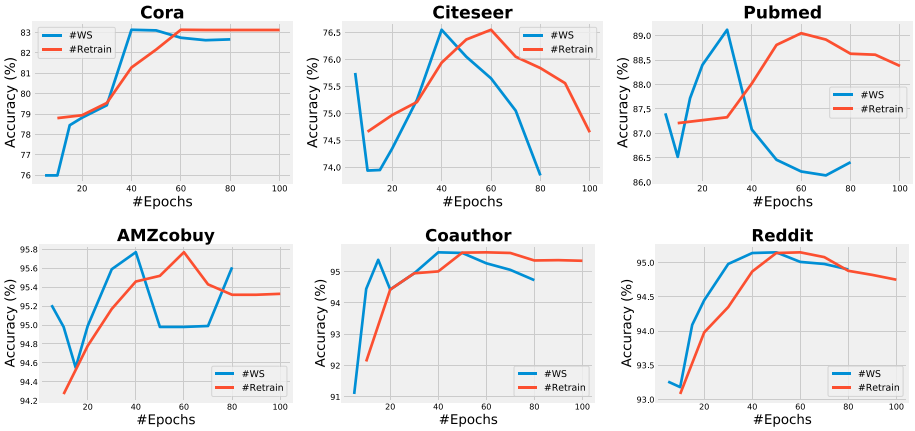

Fig. 4 Analysis of the number of warm-up steps WS (blue curve) and the number of retraining epochs $Retrain$ (red curve) for GraphNS inference (nr=0.3)

column shows the annotated noisy labels, where $nr = 0.1$. The visualization clarifies that GraphNS can recover the prediction as close as that before perturbation.

4.4 Analysis of Parameters

In this experiment, we analyze how the number of warm-up steps WS and the number of retraining epochs $Retrain$ affects the accuracy. We select GCN as the node classifier and conduct this analysis on the validation set with nr=0.3. We fix the number of epochs for inference as 100 since our model can achieve the best performance with this range. We then apply grid search to look for the optimal values for the above-mentioned parameters, where $WS \in [5, 80]$ and $Retrain \in [20, 100]$. The results turn out that WS, $Retrain$ reach the optimum nearby 40, and 60, respectively. To display the trend clearly, we fix one parameter as its optimal value and present another one in a curve. For example, we fix $Retrain$ as 60 and investigate how the validation accuracy changes with different WS in the blue curve. To enlarge the difference, we display these curves separately in Fig. 4. We observe that both larger and smaller WS have a negative effect on accuracy. Larger WS means insufficient inference, whereas smaller WS implies inadequate epochs to build the dynamic label transition matrix ϕ. This property also applies to $Retrain$. Larger $Retrain$ is even harmful to the performance. In this study, we select the aforementioned optimal values for our model.

Besides the parameters, we also analyze the runtime of GraphNS inference. The second row of Table 5 presents the average runtime of inference with different WS. It's expected that our model has longer runtime on a larger graph. We observe that the standard deviation for each dataset is stable, which indicates that the runtime stays stable no matter how the WS changes. In addition, we also present the runtime per 100 nodes in the third row of Table 5. The result specifies that this unit runtime maintains stability as the size of the graph increases in most cases, which means that the size of the graph doesn't affect the speed of inference. However, the unit runtime does increase slightly as the graph has denser connections. We argue that the inference gets slower on the denser graph (higher degrees of nodes).

Table 5 Analysis of the average runtime and the unit runtime (per 100 nodes) for GraphNS inference with different WS (nr=0.3)

Runtime (s)	Cora	Citeseer	Pubmed	AMZcobuy	Coauthor	Reddit
Average	15.11(±0.96)	17.89(±0.85)	119.34(±3.85)	58.62(±4.44)	110.15(±2.89)	1794.90(±33.04)
Unit	0.558	0.5377	0.6053	0.7663	0.6008	0.7705

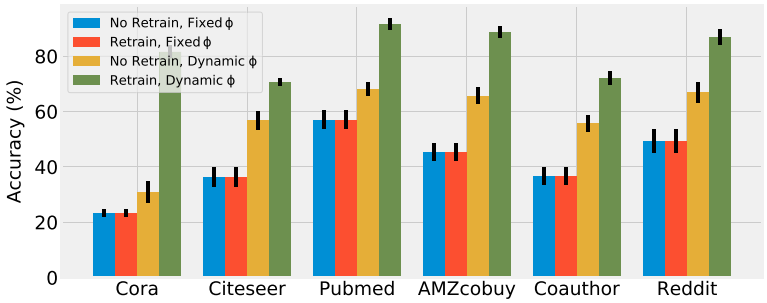

Fig. 5 Ablation study of GraphNS (on top of GCN) across six graph datasets ({Retrain, No Retrain} denote whether we retrain the node classifier in each iteration. {Fixed ϕ, Dynamic ϕ} denote whether we dynamically update the transition matrix ϕ in each iteration.)

4.5 Ablation Study

We conduct an ablation study on top of GCN to illustrate how the label transition matrix ϕ and retraining affect the defending performance of our model architecture under four scenarios. We follow the same procedure as our previous defense experiments and repeat the experiments five times. According to Fig. 5, we observe that the accuracy has no change without dynamically updating the transition matrix ϕ. In the last two scenarios (Dynamic ϕ), we notice that retraining the node classifier can help increase the accuracy further. This study reveals that dynamic transition matrix ϕ is a crucial component of GraphNS. Retraining can further promote the robustness of the node classifier with the help of dynamic ϕ.

4.6 Limitation, Future Probe, and Community Impact

The inference of GraphNS significantly depends on the warm-up transition matrix ϕ', which is built with the categorical distribution on the train graph. Such dependence indicates that GraphNS can only handle the evasion attacks at this point. In the future, our model can be improved to defend the poisoning attacks by iteratively updating the noisy labels if necessary. Besides, we could also use auto-generated labels as noisy labels to supervise the categorical distribution, which may be used in the label scarcity scenario for self-training purposes.

These potential directions could benefit the GCNs community. On the one hand, most existing defending methods don't consider the recovery from the point of node distribution. In this direction, there are still plenty of low-hanging fruits waiting for us. On the other hand, incorporating self-supervision into the defense of GCNs is a promising approach. Our model shows a new alternative path from this perspective.

5 Conclusion

In this paper, we first generalize noisy supervision as a subset of self-supervised learning methods. This generalization regards the annotated noisy label as one kind of self-information for each node. The robustness of the node classifier can be improved by such self-information. We then propose a new Bayesian graph noisy self-supervision model, namely GraphNS, to accomplish this improvement by supervising the categorical distribution of the latent labels based on dynamic conditional label transition, which follows the Dirichlet distribution. Extensive experiments demonstrate that GraphNS can enhance node classification against both label scarcity and adversarial attacks. This enhancement proves to be generalized over four classic GCNs and is superior to the competing methods across six public graph datasets.

A Implementation

A.1 Hardware and Software

All above-mentioned experiments are conducted on the server with the following configurations:

- Operating System: Ubuntu 18.04.5 LTS
- CPU: Intel(R) Xeon(R) Gold 6258R CPU @ 2.70 GHz
- GPU: NVIDIA Tesla V100 PCIe 16GB
- Software: Python 3.8, PyTorch 1.7.

A.2 Model Architecture and Hyper-parameters

The model architecture of GCNs and hyper-parameters are described in Table 6 and 7, respectively. We assume the Dirichlet distribution in this paper is symmetric and thus fix α as 1.0 (a.k.a. flat Dirichlet distribution).

Table 6 Model architecture of GCNs

Model	#Hops	Aggregator	#Heads	Activation	Dropout
GCN	×	×	×	ReLU	0.0
SGC	2	×	×	×	0.0
GraphSAGE	×	gcn	×	ReLU	0.0
GAT	×	×	3	ReLU	0.0

Table 7 Model hyper-parameters (#Hidden denotes the number of neurons in each hidden layer of GCNs.)

Hyper-parameters	Values
#Layers	2
#Hidden	200
Optimizer	Adam
Learning Rate	1×10^{-3}

References

1. Bishop CM (2006) Pattern recognition and machine learning. Springer, Berlin
2. Bruna J, Zaremba W, Szlam A, LeCun Y (2013) Spectral networks and locally connected networks on graphs. arXiv preprint arXiv:1312.6203
3. Dai H, Li H, Tian T, Huang X, Wang L, Zhu J, Song L (2018) Adversarial attack on graph structured data. arXiv preprint arXiv:1806.02371
4. Defferrard M, Bresson X, Vandergheynst P (2016) Convolutional neural networks on graphs with fast localized spectral filtering. In: Advances in neural information processing systems, pp 3844–3852
5. Deng Z, Dong Y, Zhu J (2019) Batch virtual adversarial training for graph convolutional networks. arXiv preprint arXiv:1902.09192
6. Du B, Xinyao T, Wang Z, Zhang L, Tao D (2018) Robust graph-based semisupervised learning for noisy labeled data via maximum correntropy criterion. IEEE Trans Cybern
7. Du J, Zhang S, Wu G, Moura JM, Kar S (2017) Topology adaptive graph convolutional networks. arXiv preprint arXiv:1710.10370
8. Entezari N, Al-Sayouri SA, Darvishzadeh A, Papalexakis EE (2020) All you need is low (rank) defending against adversarial attacks on graphs. In: Proceedings of the 13th International Conference on Web Search and Data Mining
9. Feng F, He X, Tang J, Chua TS (2019) Graph adversarial training: Dynamically regularizing based on graph structure. IEEE Trans Knowl Data Eng
10. Feng W, Zhang J, Dong Y, Han Y, Luan H, Xu Q, Yang Q, Kharlamov E, Tang J (2020) Graph random neural network for semi-supervised learning on graphs. In: NeurIPS'20
11. Galke L, Vagliano I, Scherp A (2019) Can graph neural networks go "online"? an analysis of pretraining and inference. arXiv preprint arXiv:1905.06018
12. Geisler S, Zügner D, Günnemann S (2020) Reliable graph neural networks via robust aggregation. Adv Neural Inf Process Syst, 33
13. Goldberger J, Ben-Reuven E (2017) Training deep neural-networks using a noise adaptation layer. International Conference on Learning Representations
14. Hamilton W, Ying Z, Leskovec J (2017) Inductive representation learning on large graphs. In: Advances in neural information processing systems, pp 1024–1034
15. Hassani K, Khasahmadi AH (2020) Contrastive multi-view representation learning on graphs. In: International conference on machine learning, PMLR, pp 4116–4126
16. Hu W, Liu B, Gomes J, Zitnik M, Liang P, Pande V, Leskovec J (2019a) Strategies for pre-training graph neural networks. In: International conference on learning representations
17. Hu W, Fey M, Zitnik M, Dong Y, Ren H, Liu B, Catasta M, Leskovec J (2020a) Open graph benchmark: Datasets for machine learning on graphs. arXiv preprint arXiv:2005.00687
18. Hu Z, Fan C, Chen T, Chang KW, Sun Y (2019b) Pre-training graph neural networks for generic structural feature extraction. arXiv preprint arXiv:1905.13728
19. Hu Z, Dong Y, Wang K, Chang KW, Sun Y (2020b) Gpt-gnn: Generative pre-training of graph neural networks. In: Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining, pp 1857–1867
20. Hwang D, Park J, Kwon S, Kim KM, Ha JW, Kim HJ (2020) Self-supervised auxiliary learning with meta-paths for heterogeneous graphs. arXiv preprint arXiv:2007.08294
21. Jin H, Zhang X (2019) Latent adversarial training of graph convolution networks. In: ICML workshop on learning and reasoning with graph-structured representations
22. Jin W, Ma Y, Liu X, Tang X, Wang S, Tang J (2020) Graph structure learning for robust graph neural networks. In: Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining, pp 66–74
23. Kefato ZT, Girdzijauskas S (2021) Self-supervised graph neural networks without explicit negative sampling. arXiv preprint arXiv:2103.14958
24. Kipf TN, Welling M (2016) Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907
25. Miller BA, Çamurcu M, Gomez AJ, Chan K, Eliassi-Rad T (2019) Improving robustness to attacks against vertex classification. In: MLG Workshop
26. Misra I, Lawrence Zitnick C, Mitchell M, Girshick R (2016) Seeing through the human reporting bias: Visual classifiers from noisy human-centric labels. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 2930–2939
27. NT H, Jin CJ, Murata T (2019) Learning graph neural networks with noisy labels. arXiv preprint arXiv:1905.01591

28. Patrini G, Rozza A, Krishna Menon A, Nock R, Qu L (2017) Making deep neural networks robust to label noise: A loss correction approach. In: Proceedings of the IEEE conference on computer vision and pattern recognition
29. Qiu J, Chen Q, Dong Y, Zhang J, Yang H, Ding M, Wang K, Tang J (2020) Gcc: Graph contrastive coding for graph neural network pre-training. In: Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining, pp 1150–1160
30. Qu M, Bengio Y, Tang J (2019) Gmnn: Graph markov neural networks. In: International conference on machine learning, PMLR, pp 5241–5250
31. Reed S, Lee H, Anguelov D, Szegedy C, Erhan D, Rabinovich A (2014) Training deep neural networks on noisy labels with bootstrapping. arXiv preprint arXiv:1412.6596
32. Rong Y, Bian Y, Xu T, Xie W, Wei Y, Huang W, Huang J (2020) Self-supervised graph transformer on large-scale molecular data. Adv Neural Inf Process Syst, 33
33. Sen P, Namata G, Bilgic M, Getoor L, Galligher B, Eliassi-Rad T (2008) Collective classification in network data. AI magazine
34. Shang J, Ma T, Xiao C, Sun J (2019) Pre-training of graph augmented transformers for medication recommendation. arXiv preprint arXiv:1906.00346
35. Shchur O, Mumme M, Bojchevski A, Günnemann S (2018) Pitfalls of graph neural network evaluation. Relational Representation Learning Workshop, NeurIPS
36. Sukhbaatar S, Bruna J, Paluri M, Bourdev L, Fergus R (2014) Training convolutional networks with noisy labels. arXiv preprint arXiv:1406.2080
37. Sun K, Zhu Z, Lin Z (2019) Multi-stage self-supervised learning for graph convolutional networks. arXiv preprint arXiv:1902.11038
38. Sun L, Dou Y, Yang C, Wang J, Yu PS, Li B (2018) Adversarial attack and defense on graph data: A survey. arXiv preprint arXiv:1812.10528
39. Tang X, Li Y, Sun Y, Yao H, Mitra P, Wang S (2020) Transferring robustness for graph neural network against poisoning attacks. In: Proceedings of the 13th international conference on web search and data mining
40. Tsitsulin A, Mottin D, Karras P, Bronstein A, Müller E (2018) Sgr: Self-supervised spectral graph representation learning. arXiv preprint arXiv:1811.06237
41. Veličković P, Cucurull G, Casanova A, Romero A, Liò P, Bengio Y (2018) Graph attention networks. In: International conference on learning representations
42. Wang S, Chen Z, Ni J, Yu X, Li Z, Chen H, Yu PS (2019a) Adversarial defense framework for graph neural network. arXiv preprint arXiv:1905.03679
43. Wang X, Liu X, Hsieh CJ (2019b) Graphdefense: Towards robust graph convolutional networks. arXiv preprint arXiv:1911.04429
44. Wu F, Zhang T, Souza Jr AHd, Fifty C, Yu T, Weinberger KQ (2019a) Simplifying graph convolutional networks. arXiv preprint arXiv:1902.07153
45. Wu H, Wang C, Tyshetskiy Y, Docherty A, Lu K, Zhu L (2019b) Adversarial examples on graph data: Deep insights into attack and defense. arXiv preprint arXiv:1903.01610
46. Xie Y, Xu Z, Zhang J, Wang Z, Ji S (2021) Self-supervised learning of graph neural networks: A unified review. arXiv preprint arXiv:2102.10757
47. Xu B, Shen H, Cao Q, Qiu Y, Cheng X (2019a) Graph wavelet neural network. arXiv preprint arXiv:1904.07785
48. Xu K, Hu W, Leskovec J, Jegelka S (2018) How powerful are graph neural networks? arXiv preprint arXiv:1810.00826
49. Xu K, Chen H, Liu S, Chen PY, Weng TW, Hong M, Lin X (2019b) Topology attack and defense for graph neural networks: an optimization perspective. arXiv preprint arXiv:1906.04214
50. Yao J, Wu H, Zhang Y, Tsang IW, Sun J (2019) Safeguarded dynamic label regression for noisy supervision. Proc AAAI Conf Artif Intell 33:9103–9110
51. Yasunaga M, Liang P (2020) Graph-based, self-supervised program repair from diagnostic feedback. In: International conference on machine learning, PMLR, pp 10799–10808
52. You Y, Chen T, Sui Y, Chen T, Wang Z, Shen Y (2020a) Graph contrastive learning with augmentations. Adv Neural Inf Process Syst, 33
53. You Y, Chen T, Wang Z, Shen Y (2020b) When does self-supervision help graph convolutional networks? In: International conference on machine learning, PMLR, pp 10871–10880
54. You Y, Chen T, Shen Y, Wang Z (2021) Graph contrastive learning automated. arXiv preprint arXiv:2106.07594
55. Zhang A, Ma J (2020) Defensevgae: Defending against adversarial attacks on graph data via a variational graph autoencoder. arXiv preprint arXiv:2006.08900

56. Zhang Y, Khan S, Coates M (2019a) Comparing and detecting adversarial attacks for graph deep learning. In: Proc. Representation Learning on Graphs and Manifolds Workshop, Int. Conf. Learning Representations, New Orleans, LA, USA
57. Zhang Y, Pal S, Coates M, Ustebay D (2019) Bayesian graph convolutional neural networks for semi-supervised classification. Proc AAAI Conf Artif Intell 33:5829–5836
58. Zheng C, Zong B, Cheng W, Song D, Ni J, Yu W, Chen H, Wang W (2020) Robust graph representation learning via neural sparsification. In: International conference on machine learning, PMLR, pp 11458–11468
59. Zhong JX, Li N, Kong W, Liu S, Li TH, Li G (2019) Graph convolutional label noise cleaner: Train a plug-and-play action classifier for anomaly detection. In: Proceedings of the IEEE conference on computer vision and pattern recognition
60. Zügner D, Günnemann S (2019) Adversarial attacks on graph neural networks via meta learning. arXiv preprint arXiv:1902.08412
61. Zügner D, Akbarnejad A, Günnemann S (2018) Adversarial attacks on neural networks for graph data. In: Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining, pp 2847–2856