Defending Graph Convolutional Networks against Dynamic Graph Perturbations via Bayesian Self-Supervision

Jun Zhuang, Mohammad Al Hasan
Indiana University-Purdue University Indianapolis, IN, USA
junz@iu.edu, alhasan@iupui.edu

Abstract
In recent years, plentiful evidence illustrates that Graph Convolutional Networks (GCNs) achieve extraordinary accomplishments on the node classification task. However, GCNs may be vulnerable to adversarial attacks on label-scarce dynamic graphs. Many existing works aim to strengthen the robustness of GCNs; for instance, adversarial training is used to shield GCNs against malicious perturbations. However, these works fail on dynamic graphs for which label scarcity is a pressing issue. To overcome label scarcity, self-training attempts to iteratively assign pseudo-labels to highly confident unlabeled nodes but such attempts may suffer serious degradation under dynamic graph perturbations. In this paper, we generalize noisy supervision as a kind of self-supervised learning method and then propose a novel Bayesian self-supervision model, namely GraphSS, to address the issue. Extensive experiments demonstrate that GraphSS can not only affirmatively alert the perturbations on dynamic graphs but also effectively recover the prediction of a node classifier when the graph is under such perturbations. These two advantages prove to be generalized over three classic GCNs across five public graph datasets.

Introduction
Graph Convolutional Networks (GCNs) have been widely ascertained to achieve extraordinary accomplishments on various node classification tasks on large graphs (Kipf and Welling 2016; Du et al. 2017; Bruna et al. 2013; Defferrard, Bresson, and Vandergheynst 2016; Wu et al. 2019a; Hamilton, Ying, and Leskovec 2017). For example, GCN-based node classification can be used for clustering users into different groups based on their interests and behavior in online social networks (OSNs), such as Facebook and Twitter. To do so, unlabeled nodes are annotated so that a supervised node classification model can be built, which can subsequently be used for advertisement, product recommendation, etc. However, GCNs may be vulnerable to adversarial attacks on label-scarce dynamic graphs. The label-scarce issue arises in a dynamic graph as new nodes appear, which are not annotated; lacking annotated labels undermine the training of robust GCNs (Hu et al. 2019a). Besides, temporal changes of the graph structure make GCNs vulnerable to adversarial attacks (Dai et al. 2018; Du et al. 2018) on any nodes at any time. We denote such adversarial attacks as Dynamic Graph Perturbations (DGP). Our objective in this work is to safeguard GCNs’ performance in presence of Dynamic Graph Perturbations.

Extensive researches have been pursued to improve the robustness of GCNs on node classification tasks (Sun et al. 2018). Such efforts can be categorized into four categories: adversarial learning methods (Jin and Zhang 2019; Deng, Dong, and Zhu 2019; Feng et al. 2019; Wang et al. 2019; Zhang and Ma 2020; Liu et al. 2018), self-supervised learning methods (You et al. 2020a; Hassan and Khasahmadi 2020; Sun, Lin, and Zhu 2020), topological denoising methods (Rong et al. 2019; Luo et al. 2021), and mechanism designing methods (Zhang and Zitnik 2020; Geisler, Zügner, and Günnemann 2020; Chen et al. 2021; Liu et al. 2021; Jin et al. 2021). The adversarial learning methods improve the robustness of GCNs by training with adversarial samples. However, these methods can barely train a robust model under the circumstance of extensive label scarcity. To overcome label scarcity, a kind of self-supervised learning method, namely pre-training (Galke, Vagliano, and Scherp 2019; Hu et al. 2019b, 2020; Qiu et al. 2020; Shang et al. 2019), is introduced, which 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 generalized pretext task that can be beneficial to arbitrary downstream tasks. Topological denoising methods prune suspicious edges on the graph before feeding this graph into GCNs. Nonetheless, such methods are not efficient on dynamic graphs as the graph structure is always changing over time. Mechanism designing methods mainly propose a message-passing mechanism, a.k.a. graph convolutional operator or node aggregator, to better classify the nodes. Such a mechanism sometimes highly depends on heuristics explorations.

Self-training (Sun, Zhu, and Lin 2019) is an extended form of pre-training. It assigns self-generated pseudo-labels to highly confident unlabeled nodes and then adds these nodes to the set of labeled nodes for the next training iteration. Nonetheless, accurate assignment of such pseudo-labels may suffer serious degradation when the graph is undergoing dynamic graph perturbations. Learning graph representation with noisy labels is a potential solution to miti-
gate this issue (Zhuang and Hasan 2020, 2022). In this paper, we assume that noisy labels include both manually-annotated labels and auto-generated labels. We argue that such noisy labels assigned to the vertices could be regarded as a kind of self-information for each node, which can help GCN to retain its performance in presence of DGP. In fact, usage of noisy labels is a kind of defense, which is equivalent to recovering the original label distribution of a node on a perturbed graph, for which the node’s label distribution has undergone a change due to perturbation. This intuition inspires us to propose a Bayesian self-supervision model, namely GraphSS\(^1\), which recovers the original label distribution iteratively, to improve the robustness of a GCN-based node classifier against dynamic graph perturbations. First, GraphSS utilizes self-information, i.e., noisy labels of the nodes, to build a robust node classifier (without knowing the ground-truth latent labels), helping the node classification in a label-scarce graph. Most importantly, GraphSS can recover the prediction of a node classifier by iterative label transition with auto-generated labels when the graph is undergoing dynamic graph perturbations.

We summarize the contributions of this paper as follows:  
• We generalize noisy supervision as a subset of self-supervised learning methods and propose a new Bayesian self-supervision model, namely GraphSS, to improve the robustness of the node classifier on the dynamic graph. To the best of our knowledge, our work is the first model that adapts Bayesian inference with self-supervision and significantly improves the robustness of GCNs against perturbations on label-scarce dynamic graphs.  
• Extensive experiments demonstrate that GraphSS can 1) effectively recover the prediction of a node classifier against dynamic graph perturbations, and 2) affirmatively alert such perturbations on dynamic graphs. These two advantages prove to be generalized over three classic GCNs across five public graph datasets.

Methodology

In this section, we introduce the methodology of our proposed model, Bayesian self-supervision model, GraphSS, as follows. We first introduce the notation and preliminary background and then theoretically analyze the Bayesian self-supervision. Besides, we explain our algorithm and analyze its time complexity.

Notations and Preliminaries. Given an undirected attributed graph \( G = (V, E) \), where \( V = \{v_1, v_2, ..., v_N\} \) denotes the set of vertices, \( N \) is the number of vertices in \( G \), and \( E \subseteq V \times V \) denotes the set of edges between vertices. We denote \( A \in \mathbb{R}^{N \times N} \) as the symmetric adjacency matrix and \( X \in \mathbb{R}^{N \times d} \) 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 as latent labels of the vertices \( Z \in \mathbb{R}^{N \times 1} \), are unobserved. We argue that manual annotation is a potential solution to this problem but human annotation unavoidably brings into noises (Misra et al. 2016). Another potential solution is to use a trained node classifier to label the vertices with pseudo-labels. This solution also yields noisy labels. We use \( Y \in \mathbb{R}^{N \times 1} \) to denote the noisy labels, which include manual-annotated labels \( Y_m \) and auto-generated labels \( Y_a \). 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 \( Y \) and \( Z \) 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 (Kipf and Welling 2016) is our preferred GCNs’ variant. The layer-wise propagation of this GCN is presented as follows:

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

In Equation (1), \( \tilde{A} = A + I_N, \tilde{D} = D + I_N \), where \( I_N \) is the identity matrix and \( D_{i,i} = \sum_j A_{i,j} \) is the diagonal degree matrix. \( H^{(l)} \in \mathbb{R}^{N \times d} \) is the nodes hidden representation in the \( l \)-th layer, where \( H^{(0)} = X. W^{(l)} \) 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 \( z_n \) as the latent label of node \( v_n \) and \( y_n \) as the corresponding noisy label (\( y_{nk} \) is a one-hot vector representation of \( y_n \)). In our setting, the GCN can be trained by the loss function \( L \):

\[
L = -\frac{1}{N} \sum_{n=1}^{N} L(y_n, f_{b}(v_n)) = -\frac{1}{NK} \sum_{n=1}^{N} \sum_{k=1}^{K} y_{nk} \ln f_{b}(v_{nk}) \tag{2}
\]

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

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 \( Y \) to \( Z \). We represent such a matrix by \( \phi \) (the same term is also used to denote a label to label mapping function). Under the presence of \( \phi \), the loss function \( L \) in Equation (2) can be rewritten as follows:

\[
L = -\frac{1}{N} \sum_{n=1}^{N} L(y_n, \phi^{-1} \circ f_{b}(v_n)) \tag{3}
\]

However, learning accurate \( \phi \) and \( f_{b} \) is a difficult task as the latent labels of the nodes are not available. So, in our method, the \( \phi \) and \( f_{b} \) are iteratively updated to approximate the perfect one by using the Bayesian framework.

Figure 1: The diagram of Bayesian self-supervision (\( V, Z \) and \( Y \) denote the nodes, the latent labels, and the noisy labels, respectively. \( \phi \) denotes the conditional label transition matrix. \( \alpha \) denotes the Dirichlet parameter. \( N \) and \( K \) denote the number of nodes and classes, respectively.)

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

The latent label of node \( v_n \), \( z_n \) \( \sim \) \( P(\cdot \mid v_n) \), where \( P(\cdot \mid v_n) \) is a Categorical distribution modeled by the node classifier \( \phi(v_n) \). The noisy label \( y_n \) \( \sim \) \( P(\cdot \mid \phi(v_n)) \), where \( \phi(v_n) \) is the parameter of Categorical distribution \( P(\cdot \mid \phi(z_n)) \). The conditional label transition matrix \( \mathbf{\phi} = [\phi_1, \phi_2, \ldots, \phi_K]^{T} \in \mathbb{R}^{K \times K} \) consists of \( K \) transition vectors. The \( k \)-th transition vector \( \phi_k \sim \text{Dirichlet}(\alpha) \), where \( \alpha \) 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 self-supervision, so that the inferred label of a given node sampled from this updated distribution is identical to the latent label of that node as much as possible.

According to the Figure 1, we can employ Bayes’ theorem to deduce the posterior of a given node sampled from this updated distribution is identical to the latent label of that node as much as possible.

\[
P(Z \mid V, Y; \alpha) = P(\phi(v_n), \alpha) P(Z \mid V, Y, \phi)
\]

According to the conjugation property between the Multinomial distribution and the Dirichlet distribution, we can continue to deduce the Equation (4) as follows:

\[
P(Z \mid V, Y; \alpha) = \prod_{n=1}^{N} P(z_n \mid v_n) \prod_{k=1}^{K} \sum_{z} \phi_k^{z_n} \prod_{n=1}^{N} \phi_n y_n d\phi
\]

\[
P(Z \mid V, Y; \alpha) = \prod_{n=1}^{N} P(z_n \mid v_n) \prod_{k=1}^{K} \sum_{z} \phi_k^{z_n} \prod_{n=1}^{N} \phi_n y_n d\phi
\]

\[
= \prod_{n=1}^{N} P(z_n \mid v_n) \prod_{k=1}^{K} \sum_{z} \phi_k^{z_n} \prod_{n=1}^{N} \phi_n y_n d\phi
\]

\[
= \prod_{n=1}^{N} P(z_n \mid v_n) \prod_{k=1}^{K} \sum_{z} \phi_k^{z_n} \prod_{n=1}^{N} \phi_n y_n d\phi
\]

\[
= \prod_{n=1}^{N} P(z_n \mid v_n) \prod_{k=1}^{K} \sum_{z} \phi_k^{z_n} \prod_{n=1}^{N} \phi_n y_n d\phi
\]

\[
= \prod_{n=1}^{N} P(z_n \mid v_n) \prod_{k=1}^{K} \sum_{z} \phi_k^{z_n} \prod_{n=1}^{N} \phi_n y_n d\phi
\]

Here we denote the confusion matrix between the node prediction and the noisy labels as \( \mathbf{C} \), where \( \sum_{k=1}^{K} \phi_k = N \). The term \( \prod_{n=1}^{N} \phi_n y_n \) is expressed as \( \prod_{k=1}^{K} \prod_{n=1}^{N} \phi_k \), so that we can integrate the terms based on the aforementioned conjugation property.

Unfortunately, Equation (5) 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 \( z_n \) by fixing \( n \)-th dimension in order to satisfy the detailed balance condition on the assumption of Markov chain (Bishop 2006). Combined with Equation (5) and the recurrence relation of \( \Gamma \) function, \( \Gamma(n+1) = n\Gamma(n) \), we sample a sequence of \( z_n \) as follows:

\[
P(z_n \mid Z_{-n}, V, Y; \alpha) = \frac{P(Z \mid V, Y; \alpha)}{P(Z \mid V, Y; \alpha)} = \frac{P(z_n \mid v_n) \prod_{k=1}^{K} \sum_{z} \phi_k^{z_n} \prod_{n=1}^{N} \phi_n y_n d\phi}{P(y_n \mid v_n) \prod_{k=1}^{K} \sum_{z} \phi_k^{z_n} \prod_{n=1}^{N} \phi_n y_n d\phi}
\]

where we denote \( Z_{-n} \) as the subset of \( Z \) that removes statistic \( z_n \). In the last row of Equation (6), the first term \( P(z_n \mid v_n) \) is a categorical distribution of labels for the node \( v_n \) modeled by \( \phi \). We use the term \( P(Z \mid Y) \in \mathbb{R}^{N \times K} \) to denote the same over all the nodes. Note that \( P(Z \mid Y) \in \mathbb{R}^{N \times 1} \) in this paper denotes the predicted labels, a.k.a auto-generated labels \( Y \). To avoid confusion, we use the term \( Y \) to denote these labels in the rest of this paper.

GraphSS Algorithm and Pseudo-code. The total process of GraphSS is displayed in Figure 2. Given an undirected attribute graph, GraphSS classifies the nodes with manual-annotated labels \( Y_m \) at first and then generates both categorical distribution \( P(Z \mid V) \) and auto-generated labels \( Y_a \). After that, GraphSS applies Gibbs sampling to generate the inferred labels \( P(Z \mid V, Y_m; \alpha) \in \mathbb{R}^{N \times 1} \) and updates the label transition matrix \( \phi \) parameterized by \( \alpha \). The information of \( V \) is represented by both \( A \) and \( X \). For alert, GraphSS applies one-time label transition with \( Y_a \). For recovery, GraphSS iteratively re-trains the node classifier to update \( P(Z \mid V) \). The inference will ultimately converge, approximating \( P(Z \mid V, Y_m; \alpha) \) to \( Z \) as close as possible. In brief, the goal of GraphSS 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.

The pseudo-code of GraphSS is shown in Algorithm (1).

Training: GraphSS trains the node classifier \( \phi \) on the train graph \( G_{train} \) at first (line 1) with manual-annotated labels \( Y_m \) and then generates categorical distribution \( P(Z \mid V) \) and auto-generated labels \( Y_a \) by \( \phi \) (line 2). Inference: Before the inference, GraphSS first computes a warm-up label transition matrix \( \phi' \) by using the prediction over \( G_{train} \) (line 3) and then defines (creates empty spaces) the inferred labels \( P(Z \mid V, Y_a; \alpha) \) and the dynamic label transition matrix \( \phi \) based on the test graph \( G_{test} \) (line 4). In the warm-up stage
Algorithm 1: BAYESIAN Self-supervision

Input: Graph $G_{\text{train}}$ and $G_{\text{test}}$, which contain corresponding symmetric adjacency matrix $A$ and feature matrix $X$. Manual-annotated labels $Y_m$ in $G_{\text{train}}$. Node classifier $f_\theta$. The number of warm-up steps $W$; The number of epochs for inference $\text{Epochs}$.

1: Train $f_\theta$ by Equation (2) with $Y_m$ on $G_{\text{train}}$.
2: Generate categorical distribution $P(Z \mid V)$ and auto-generated labels $Y_\alpha$ by $f_\theta$.
3: Compute the warm-up label transition matrix $\phi'$ based on $G_{\text{train}}$.
4: Define the inferred labels $P(Z \mid V, Y_m; \alpha)$ and the dynamic label transition matrix $\phi$ based on $G_{\text{test}}$.

5: for step ← 1 to $\text{Epochs}$ do
6: if step $\leq WS$ then
7: Sample $z_n$ with warm-up $\phi'$ by Equation (6).
8: else
9: Sample $z_n$ with dynamic $\phi$ by Equation (6).
10: end if
11: Update dynamic $\phi$, $P(Z \mid V, Y_m; \alpha)$. 
12: if not Alert then
13: Retrain $f_\theta$.
14: end if
15: end for
16: return $P(Z \mid V, Y_m; \alpha)$ and Dynamic $\phi$.

of the inference, GraphSS samples $z_n$ with the warm-up $\phi'$ (line 7), which is built with the categorical distribution of $f_\theta$ and $Y_m$ on $G_{\text{train}}$. The categorical distributions of both $G_{\text{train}}$ and $G_{\text{test}}$ should have high similarity if both follow a similar distribution. Thus, the warm-up $\phi'$ is a key-stone since subsequent inference largely depends on this distribution. After the warm-up stage, GraphSS samples $z_n$ with the dynamic $\phi$ (line 9). This dynamic $\phi$ updates in every epoch with current sampled $z_n$ and corresponding $Y_m$. Simultaneously, $P(Z \mid V, Y_m; \alpha)$ is also updated based on the before-mentioned $z_n$ (line 11). For recovery, GraphSS iteratively re-trains the node classifier to update its parameters (line 13). The inference will ultimately converge, approximating the inferred labels to the latent labels as close as possible. Note that both $G_{\text{train}}$ and $G_{\text{test}}$ contain corresponding symmetric adjacency matrix $A$ and feature matrix $X$. The categorical distributions of $G_{\text{test}}$ may change abruptly when this graph is under perturbation since the original distribution in this graph is being perturbed with. In this case, GraphSS can also help recover the original categorical distribution by dynamic conditional label transition.

According to Algorithm (1), GraphSS applies Gibbs sampling via Equation (6) inside the FOR loop. The time complexity of the sampling is $O(N_{\text{test}} \times K + K^2)$ since element-wise multiplication only traverses the number of elements in matrices once, where $N_{\text{test}}$ 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., $N_{\text{test}} \gg K$. Therefore, the time complexity of this sampling operation is approximately equal to $O(N_{\text{test}})$. Hence, the time complexity of inference except the first training (line 1) is $O(\text{Epochs} \times N_{\text{test}})$, where $\text{Epochs}$ is the number of epochs for inference.

Experiments
In this section, we present and analyze experimental results as follows. We first examine how far Bayesian self-supervision (GraphSS) can defend node classifiers against dynamic graph perturbations. We then assess whether GraphSS can alert such perturbations. Furthermore, we analyze the runtime, parameters and conduct an ablation study. At last, we discuss the limitation and future directions to illustrate how the community benefits from our work.

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

| Dataset  | $|V|$ | $|E|$ | $|F|$ | $|C|$ | Avg.D |
|----------|------|------|------|------|-------|
| Cora     | 2,708| 10,556| 1,433| 7    | 3.85  |
| Citeseer | 3,327| 9,228 | 3,703| 6    | 2.78  |
| Pubmed   | 19,717| 88,651| 500  | 3    | 4.49  |
| AMZcobuy | 7,650| 287,326| 745  | 8    | 32.32 |
| Coauthor | 18,333| 327,576| 6,805| 15   | 10.01 |

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

Experimental Settings. Our proposed model is evaluated on five public datasets in Table 1. Cora, Citeseer, and Pubmed are famous citation graph data (Sen et al. 2008). AMZcobuy comes from the photo segment of the Amazon co-purchase graph (Shchur et al. 2018). Coauthor is a co-authorship graph of computer science based on the Microsoft Academic Graph from the KDD Cup 2016 challenge. For all five datasets, the percentage of train, validation, and test partition comprise 40%, 20%, and 40% of the nodes, respectively. We select such partitions to simulate the OSNs as the size of the existing graph in OSNs will not be too small compared to the new coming dynamic subgraphs. In this paper, we assume that the train graphs contain manually-annotated labels but the validation and test graphs don’t. In other words, the node classifiers are trained with manually-annotated labels whereas GraphSS applies inference with auto-generated labels. To simulate the manual-annotated labels, we randomly replace the ground-truth label of a node with another label, chosen uniformly. We denote the percentage of such replacement as noise ratio $nr$ and fix $nr = 10\%$ in the experiments. We set the number of training epochs as 200 because all node classifiers converge within 200 epochs in the training phase. To simulate the dynamic graph perturbations, we execute node-level direct evasion non-targeted attacks on the links and features of the nodes (LK&F) in dynamic graphs (Zügner, Akbarnejad, and Günnemann 2018), whereas the trained node classifier remains unchanged.

We compare our proposed model against six state-of-the-art defending methods. AdvTrain (Wang, Liu, and Hsieh 2019; You et al. 2020b) assigns pseudo labels to generated adversarial samples and then re-trains the node classifier with both noisy labeled nodes and adversarial nodes. GNN-Jaccard (Wu et al. 2019b) preprocesses the graph by eliminating suspicious connections, whose Jaccard similarity of

---

For each subgraph, we apply non-targeted adversarial at-
tacks five times to construct a group of dynamic subgraphs.

We randomly sample 20% of the nodes

GraphSS Defends Node Classifiers. We examine how far GraphSS can defend node classifiers against dynamic graph perturbations. We randomly sample 20% of the nodes from the test graph to build a subgraph and repeat this process five times to construct a group of dynamic subgraphs. For each subgraph, we apply non-targeted adversarial at-
tacks on both links and features together (L&F) (Zügner, Akbarnejad, and Günnemann 2018). To assess the general-
zation, we apply the adversarial attacks on two groups of popular node classifiers, spectral models (GCN (Kipf and Welling 2016), SGC (Wu et al. 2019a)) and spatial models (GraphSAGE (Hamilton, Ying, and Leskovec 2017)), across five public datasets. We also compare the defending performance of GraphSS with the competing methods and present the accuracy of each group of dynamic subgraphs in Table 2. The best defending performance on each dataset is high-
lighted. Originally, all three node classifiers have compara-
tive performance. After the adversarial attacks, the results

node’s features is smaller than a given threshold. GNN-
SVD (Entezari et al. 2020) proposes another preprocessing
approach with low-rank approximation on the perturbed graph to mitigate the negative effects from high-rank attacks, such as Nettack (Zügner, Akbarnejad, and Günnemann 2018). RGCN (Zhu et al. 2019) adopts Gaussian distri-
butions as the hidden representations of nodes to mitigate the negative effects of adversarial attacks. GRAND (Feng et al. 2020) proposes random propagation and consistency regularization strategies to address the issues about over-
smoothing and non-robustness of GCNs. ProGNN (Jin et al. 2020) jointly learns the structural graph properties and iter-
atively reconstructs the clean graph to reduce the effects of adversarial structure. NRGGNN (Dai, Aggarwal, and Wang 2021) develops a label noise-resistant framework that brings clean label information to unlabeled nodes based on the feature similarity. Due to limited pages, We present implemen-
tation details in supplementary materials.

Table 2: The comparison of defending results (test accuracy) between GraphSS and the competing methods (Original/Attack denote the test accuracy before/after the non-target attacks (L&F).)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Cora</td>
<td>69.95 (±0.53)</td>
<td>40.62 (±0.12)</td>
<td>42.60 (±0.95)</td>
<td>54.30 (±1.69)</td>
<td>72.48 (±1.28)</td>
<td>68.40 (±1.70)</td>
<td>75.65 (±2.16)</td>
<td>69.84 (±1.74)</td>
<td>68.70 (±2.04)</td>
<td>79.02 (±2.91)</td>
<td>87.70 (±1.30)</td>
</tr>
<tr>
<td>Citeseer</td>
<td>83.43 (±1.21)</td>
<td>42.60 (±0.95)</td>
<td>42.60 (±0.95)</td>
<td>54.30 (±1.69)</td>
<td>72.48 (±1.28)</td>
<td>68.40 (±1.70)</td>
<td>75.65 (±2.16)</td>
<td>69.84 (±1.74)</td>
<td>68.70 (±2.04)</td>
<td>79.02 (±2.91)</td>
<td>87.70 (±1.30)</td>
</tr>
<tr>
<td>Pubmed</td>
<td>69.95 (±0.53)</td>
<td>40.62 (±0.12)</td>
<td>42.60 (±0.95)</td>
<td>54.30 (±1.69)</td>
<td>72.48 (±1.28)</td>
<td>68.40 (±1.70)</td>
<td>75.65 (±2.16)</td>
<td>69.84 (±1.74)</td>
<td>68.70 (±2.04)</td>
<td>79.02 (±2.91)</td>
<td>87.70 (±1.30)</td>
</tr>
<tr>
<td>AMZcobuy</td>
<td>91.99 (±0.96)</td>
<td>91.99 (±0.96)</td>
<td>91.99 (±0.96)</td>
<td>91.99 (±0.96)</td>
<td>91.99 (±0.96)</td>
<td>91.99 (±0.96)</td>
<td>91.99 (±0.96)</td>
<td>91.99 (±0.96)</td>
<td>91.99 (±0.96)</td>
<td>91.99 (±0.96)</td>
<td>91.99 (±0.96)</td>
</tr>
</tbody>
</table>

Table 2: The comparison of defending results (test accuracy) between GraphSS and the competing methods (Original/Attack denote the test accuracy before/after the non-target attacks (L&F).)
because RGCN adopts a sampling in the hidden representation, whereas this sampling may lose more information on denser graphs. 3) GNN-SVD obtains lower accuracy than GNN-Jaccard across all graphs as GNN-SVD is tailored to fit the targeted attacks and may not adapt well to the non-targeted attacks. 4) GRAND deteriorates seriously on Cora. One of the reasons for this deterioration is that GRAND assumes the graph satisfies the homophily property. The assumption may break when the graph suffers serious perturbations. Besides, GRAND is sensitive to the parameter’s setting as we spent substantial time tuning the optimal parameters on different datasets. On the contrary, GraphSS does not assume such network property, and it is also easy to tune. 5) ProGNN is superior to GNN-Jaccard on denser graphs. We argue that ProGNN can simultaneously update the graph structure and the parameters of the node classifiers with the low rank and feature smoothness constraints. Such an update reaps huge fruits for recovering better graph structures on denser graphs.

Furthermore, we visualize our defending result (on top of GCN) on the test graph of Cora as an example in Figure 4, which presents the result of prediction before/after graph perturbations, and our defending result. The visualization clarifies that GraphSS can recover the prediction after perturbations as close as that in a clean environment.

**GraphSS Can Alert Dynamic Graph Perturbations.** We assess whether GraphSS can alert the perturbations on dynamic graphs. We repeat the aforementioned procedure ten times to construct a group of dynamic subgraphs and then randomly select three subgraphs from the group to be perturbed by non-targeted adversarial attacks (L&F) (Zügner, Akbarnejad, and Günnemann 2018). We assess our model on three node classifiers across five public datasets and present the performance via ROC curve and AUC score in Figure 3. A higher true positive rate (TPR) indicates that GraphSS has a higher probability to successfully alert the true perturbations. On the contrary, a higher false positive rate (FPR) implies that GraphSS has more chance to make a false alert on unperturbed subgraphs. Figure 3 reveals that GraphSS achieves robust alerts on top of GCN across all datasets. For the other two node classifiers, GraphSS has higher TPR and AUC scores with GraphSAGE than with SGC. This indicates that GraphSS accomplishes better alert performance on top of GraphSAGE when the subgraph is undergoing perturbed.

**Analysis of Runtime.** We analyze the runtime of GraphSS between defense and alert. For each dataset in Table 3, the left column presents the average runtime whereas the right column presents the unit runtime. It's expected that GraphSS has longer runtime on a larger graph. However, the unit runtime maintains a comparable level as the size of the graph increases. This result indicates that the size of the graph does not affect the speed of inference. We observe that the average runtime of defense is slightly higher than that of alert. We argue that such a gap is reasonable as GraphSS iteratively retrains the node classifiers in the defense scenario. Overall, this analysis illustrates that GraphSS retains a stable speed of inference regardless of the size of the graph. We also sort the defense runtime of 5 methods on Cora as an example in Table 4. The sorting indicates that topological denoising methods mainly apply on the preprocessing stage.

---

**Table 3: Analysis of the average runtime and the unit runtime (per 100 nodes) of GraphSS between defense and alert**

<table>
<thead>
<tr>
<th></th>
<th>Cora</th>
<th>Citeeseer</th>
<th>Pubmed</th>
<th>AMZcobuy</th>
<th>Coauthor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Runtime (s)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>GCN</strong></td>
<td>Defense</td>
<td>Alert</td>
<td>Defense</td>
<td>Alert</td>
<td>Defense</td>
</tr>
<tr>
<td></td>
<td>22.75 ± 0.65</td>
<td>0.84</td>
<td>27.86 ± 0.66</td>
<td>0.84</td>
<td>163.57 ± 3.29</td>
</tr>
<tr>
<td></td>
<td>21.28 ± 0.31</td>
<td>0.79</td>
<td>26.14 ± 0.44</td>
<td>0.79</td>
<td>155.84 ± 1.57</td>
</tr>
<tr>
<td><strong>SGC</strong></td>
<td>Defense</td>
<td>Alert</td>
<td>Defense</td>
<td>Alert</td>
<td>Defense</td>
</tr>
<tr>
<td></td>
<td>22.07 ± 1.13</td>
<td>0.81</td>
<td>28.06 ± 0.45</td>
<td>0.84</td>
<td>163.29 ± 3.28</td>
</tr>
<tr>
<td></td>
<td>22.13 ± 2.17</td>
<td>0.82</td>
<td>26.20 ± 0.54</td>
<td>0.79</td>
<td>156.06 ± 1.51</td>
</tr>
<tr>
<td><strong>GraphSAGE</strong></td>
<td>Defense</td>
<td>Alert</td>
<td>Defense</td>
<td>Alert</td>
<td>Defense</td>
</tr>
<tr>
<td></td>
<td>22.04 ± 0.91</td>
<td>0.81</td>
<td>28.51 ± 0.59</td>
<td>0.86</td>
<td>165.44 ± 0.86</td>
</tr>
<tr>
<td></td>
<td>21.38 ± 0.22</td>
<td>0.79</td>
<td>26.06 ± 0.36</td>
<td>0.78</td>
<td>154.94 ± 1.04</td>
</tr>
</tbody>
</table>

**Figure 3: The assessment of alerting dynamic graph perturbations by GraphSS via ROC curve**

**Figure 4: The confusion matrices (heatmap) of the defending result on Cora by GraphSS (We apply log-scale to the confusion matrix for fine-grained visualization.)**

---

4410
and may have faster speed on smaller graphs.

<table>
<thead>
<tr>
<th>dataset</th>
<th>#WS</th>
<th>#Retrain</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cora</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Citeseer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pubmed</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AMZcobuy</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Coauthor</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4: Runtime (s) comparison among 5 methods on Cora

Analysis of Parameters. We investigate how the number of warm-up steps $W S$ and the number of retraining epochs $\text{Retrain}$ affects the defending performance. We select GCN as the node classifier and conduct this analysis on the validation set. We fix the number of inference epochs as 100 and then apply grid search to look for the optimal parameters, where $W S \in [5, 80]$ and $\text{Retrain} \in [20, 100]$. The results turn out that $W S, \text{Retrain}$ 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 $\text{Retrain}$ as 60 and investigate how the validation accuracy changes with different $W S$ in the blue curve. To enlarge the difference, we display these curves separately in Figure 5. We observe that both larger and smaller $W S$ have a negative effect on accuracy. Larger $W S$ means insufficient inference, whereas smaller $W S$ implies inadequate epochs to build the dynamic label transition matrix $\phi$. This property also applies to $\text{Retrain}$. Larger $\text{Retrain}$ is even harmful to the performance. In this study, we select the aforementioned optimal values for our model.

Ablation Study. To better understanding the model architecture, we conduct an ablation study on top of GCN to illustrate how the label transition matrix $\phi$ and retraining affect the defending performance of GraphSS under four scenarios. We follow the same procedure as our previous defense experiments. According to Figure 6, we observe that the accuracy has no change without dynamically updating the transition matrix $\phi$. In the last two scenarios (Dynamic $\phi$), we notice that retraining the node classifier can help increase the accuracy further. This study reveals that dynamic transition matrix $\phi$ is a crucial component of GraphSS. Retraining can further promote the robustness of the node classifier with the help of dynamic $\phi$.

Limitation and Future Directions. 1) The inference of GraphSS depends on the warm-up label transition matrix $\phi^t$, which is built with the categorical distribution on the train graph. Such dependence indicates that GraphSS can only handle the evasion attacks at this point. In the future, GraphSS can be improved to defend the poisoning attacks by iteratively updating the auto-generated labels. Besides, this update may benefit the node classifier under the label scarcity scenario for self-training purposes. 2) On the other hand, GraphSS can only alert the perturbations in this version. However, such a version could be extended to detect the anomaly nodes on dynamic graphs.

Related Work

Related works about the defense of GCNs are already discussed in Introduction. Due to the page limit, we only present the studies about learning deep neural networks with noisy labels in this section (Reed et al. 2014; Goldberger and Ben-Reuven 2016; Patrini et al. 2017; Yao et al. 2019; Sukhbaatar et al. 2014; Misra et al. 2016). Those works achieve great improvement in conventional CNNs.

Conclusion

In this paper, we generalize noisy supervision as a subset of self-supervised learning methods. This generalization regards the noisy labels, including both manual-annotated labels and auto-generated labels, as one kind of self-information for each node. The robustness of the node classifier can be improved by utilizing such self-information. We then propose a new Bayesian self-supervision model, namely GraphSS, 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 GraphSS can not only affirmatively alert the perturbations on dynamic graphs but also effectively recover the prediction of a node classifier when the graph is under such perturbations. These two advantages prove to be generalized over three classic GCNs across five public graph datasets.
Acknowledgments
This research is partially supported by National Science Foundation with grant number IIS-1909916.

Broader Impact
We confirm that we fulfill the author responsibilities.

Many existing defending methods confront the adversarial perturbations from the perspective of topological denoising and message-passing mechanism. The low-hanging fruits from these two aspects have been widely explored. On the contrary, our model improves the robustness of the node classifiers by recovering the node distribution. Also, incorporating self-supervision into the defense of GCNs is a promising direction in the recent two years. Our work provides an innovative alternative in this path and could be a baseline model for the above-mentioned future directions.

References


Feng, F.; He, X.; Tang, J.; and Chua, T.-S. 2019. Graph adversarial training: Dynamically regularizing based on graph structure. IEEE Transactions on Knowledge and Data Engineering.
the 14th ACM International Conference on Web Search and Data Mining.


