RoboGNN: Robustifying Node Classification under Link Perturbation

Sheng Guan , Hanchao Ma , Yinghui Wu
Case Western Reserve University
{sxg967,hxm382,yxw1650}@case.edu

Abstract

Graph neural networks (GNNs) have emerged as powerful approaches for graph representation learning and node classification. Nevertheless, they can be vulnerable (sensitive) to link perturbations due to structural noise or adversarial attacks. This paper introduces RoboGNN, a novel framework that simultaneously robustifies an input classifier to a counterpart with certifiable robustness, and suggests desired graph representation with auxiliary links to ensure the robustness guarantee. (1) We introduce (p, θ)-robustness, which characterizes the robustness guarantee of a GNN-based classifier if its performance is insensitive for at least θ fraction of a targeted set of nodes, under any perturbation of a set of vulnerable links up to a bounded size p. (2) We present a co-learning framework that interacts model learning with graph structural learning to robustify an input model M to a (p, θ)-robustness counterpart. The framework also outputs the desired graph structures that ensure the robustness. Using real-world benchmark graphs, we experimentally verify that RoboGNN can effectively robustify representative GNNs with guaranteed robustness, and desirable gains on accuracy.

1 Introduction

Graph neural networks (GNNs) [Gori *et al.*, 2005] have shown good performance for graph representation learning and downstream tasks. GNNs adopt a label propagation architecture to learn discriminative node embeddings with graph convolutional layers. In each layer, the embedding of a node is updated by aggregating its counterparts from neighbors.

GNNs learning assume and rely on complete and accurate link structures from an underlying graph G. Having this said, they are often sensitive and vulnerable to even small link perturbations (*e.g.,* adding or removing edges) due to noisy links [Tran *et al.*, 2017; Paulheim, 2017] or malicious adversarial attacks [Dai *et al.*, 2018; Zügner and Günnemann, 2019]. For example, a GNN-based classifier M can be sensitive under a set of link perturbations posed to graph G where M is trained on, if its output label of a same node *changes* as G is modified accordingly at training time. It is thus often desirable if (1) the robustness is ensured for designated targeted nodes of interests, (2) a small set of auxiliary links ΔL that should be "protected", are derived to suggest how to mitigate the negative impact of the perturbations. This calls for proper modeling to improve the robustness of pretrained GNNs.

We consider a novel and practical problem as follows.

- **Input**: a (perturbed) graph G, an input model M, a set of targeted nodes V_T, a set of "vulnerable" links E_p that may be perturbed, and a budget p;
- **Output**: a triple $(G', M', \Delta L)$, such that M' is insensitive to a desirable amount (θ fraction) of nodes in V_T, for any perturbation of at most p links in $E_p \setminus \Delta L$.

In a nutshell, the problem aims to (1) "robustify" M to a counterpart M' such that M' ensures robust performance for a desired amount of designated target nodes, and (2) also generate, in accordance, a proper graph representation G' and a small set of links $\Delta L \subseteq E_p$ to be "protected" from attacks to ensure the robustness. In addition, the size $|\Delta L|$ reflects "defense effort" (*e.g.,* cost to the protection of social links or communication networks) and should be minimized.

The above problem has its components in prior work and is of both theoretical and practical interest. (1) Certifying robustness over entire node set [Bojchevski and Günnemann, 2019] (which requires the predicated label is insensitive to perturbations) can be an overkill for models with desirable guarantees. We introduce a configurable robustness in terms of a threshold θ over designated target nodes, enabling flexible robustification scenarios. (2) The generated auxiliary structures ΔL and graph representation G' can be readily used to (explicitly) "recover" the graphs or directly applied to learn other GNN-based models.

Contribution. This paper introduces RoboGNN, a co-learning framework to robustify GNN-based classification with desirable, configurable robustness guarantees.
(1) We introduce a notion of (p, θ)-robustness to characterize the robustness of GNNs (Section 2). Given graph G and classifier M, a pair (G, M) is (p, θ)-robust *w.r.t.* E_p and V_T, if the output of M is insensitive for at least $\theta\%$ of V_T under any manipulation of at most p links in E_p. We formalize (p, θ) verification and robustification problems, establish their hardness, and introduce an algorithm to verify the (p, θ)-robustness for a pair (G, M) *w.r.t.* E_p and V_T. Our goal aims

to refine G to G' and robustify M to M' *w.r.t.* E_p and V_T, such that (G', M') is (p, θ)-robust.

(2) We investigate the impact of changes of E_p to the robustness, and establish a monotonicity property, which states that the (p, θ)-robustness of a model M *w.r.t.* E_p and V_T remains intact on any subset of E_p. Based on the property, we study an optimization problem that aims to compute a smallest set of links $\Delta L \subseteq E_p$ such that (G, M) is (p, θ)-robust over $E_p \setminus \Delta L$ (Section 3). While finding the smallest ΔL remains intractable, we present a fast heuristic strategy to compute a small protection set ΔL, which dynamically ranks V_T based on the likelihood that the model is robust at one single node, and incrementally augment ΔL following efficient traversal.

(3) Based on the verification and minimality condition, we present RoboGNN, a framework to robustify an input model with guaranteed robustness. RoboGNN learns the graph representation and GNNs iteratively towards (p, θ) robustness with a goal to minimize $|\Delta L|$. The learning process is guided by minimizing a combination of robust hinge loss and the distance between perturbed and original adjacency matrix.

Using real-world benchmark datasets, our results confirm that RoboGNN effectively improves the robustness and accuracy of GNN-based classification, provides configurable robustness guarantees, and can explicitly suggest only small amount of links to be protected.

Related Work. Certifiable robustness is introduced in [Bojchevski and Günnemann, 2019]. A node is certifiably robust if its label predicted by a model is not sensitive to perturbations to a set of fragile edges. Models can be improved by training that minimizes a hinge loss penalty. We study (p, θ)-robustness that extends certifiable robustness with configurable p and θ to support the need for robustification.

GCN-Jaccard [Wu *et al.*, 2019] removes malicious links added to nodes with dissimilar features measured by Jaccard similarity. GCN-SVD [Entezari *et al.*, 2020] assumes an attack model [Zügner *et al.*, 2018] that affects high-rank singular components of the graph and performs the low-rank approximation for graph reconstruction to mitigate the effects. Edge dithering [Ioannidis and Giannakis, 2019] generates auxiliary graphs with edge insertions and deletions against adversarial perturbations to facilitate robust learning. Graph sanitation [Xu *et al.*, 2021] solves a bilevel optimization problem that aims to modify perturbed graphs to improve underlying semi-supervised learning. Pro-GNN [Jin *et al.*, 2020] integrates graph properties *e.g.,* sparsity, low rank, and feature smoothness to its loss function and learns to clean graph.

In contrast to prior work, our approach takes a different strategy that aims to find protection sets and jointly learns better graph representation to ensure (p, θ)-robustness of targeted nodes that can be specified by users. Robustifying node classification with (a) a configurable robustness guarantee, and (b) both useful auxiliary structures and links that should be protected, is not discussed in prior work.

2 Model Robustness: A Characterization

Graphs. A graph $G = (V, E)$ has a finite set of nodes V and edges E. The representation of G is a pair (X, A), where X is a feature matrix ($X \in \mathbb{R}^{|V| \times d}$) that records a feature vector $x_v \in \mathbb{R}^d$ for each node $v \in V$ (obtained by embedding functions [Harris and Harris, 2010]); and A is the adjacency matrix of G. A *link* in G is a node pair $(v, v') \in V \times V$.

A *perturbation* of a link (v, v') in G is either a deletion of an edge $(v, v') \in E$, or insertion of a link $(v, v') \notin E$. A *vulnerable set* $E_p \subseteq V \times V$ of G refers to a set of links to which an adversarial perturbation may occur. We remark that E_p records the "original" status of links: if $(v, v') \in E_p$ is an edge in E (resp. a node pair not in E), an (adversarial) perturbation ("a flip") removes (v, v') from (resp. inserts (v, v') to) G. We use (v, v') to denote a perturbation of a link (v, v').

Node classification. Given a graph $G = (V, E)$ and a set of labeled training nodes $V_T \subseteq V$, node classification is to learn a model M to infer the labels of a set of unlabeled test nodes.

We consider GNN-based classifiers. A GNN [Wu *et al.*, 2020] transforms (X, A) to proper representation (logits) for downstream tasks. A GNN with n layers iteratively gathers and aggregates information from neighbors of a node v to compute node embedding of v. Denote the output features $\mathbf{h}_v^i$ (with v ranges over V) at layer i as $\mathbf{h}^i$. A GNN computes $\mathbf{h}^i$ as $\mathbf{h}^i = \delta(\|_{j=c}^{n} \tilde{A}^j \mathbf{h}^{i-1} \mathbf{W}_j^i)$, where $\|$ denotes the horizontal concatenation operation, $\mathbf{W}_j^i$ refers to the learnable weight matrix of order j in layer i, $\delta(\cdot)$ is an activation, and $\tilde{A}$ is a normalized adjacency matrix. Notable GNN variants are GCN and GraphSage [Hamilton *et al.*, 2017] that samples fixed-size neighbors (c=0,n=1) and GAT [Veličković *et al.*, 2017] that incorporates self-attention for neighbors. Specifically, we make case for GNNs that leverages personalized PageRank, which mitigates over-smoothing [Cai and Wang, 2020]. In such models, $Z = \mathbf{\Pi h}^n$, where $\mathbf{\Pi}$ is a PageRank matrix, $\mathbf{h}^n$ is the output from the last layer of the GNN.

A GNN-based classifier M outputs logits $Z \in \mathbb{R}^{|V| \times |L|}$ that are fed to a softmax layer and transformed to Z' that encodes the probabilities of assigning a class label to a node. The training of M minimizes a loss function $\mathcal{L}_{CE}(Z, A) = -\sum_{v \in V_T} Y_v \ln Z'_v$, where Y_v is the label of a training node $v \in V_T$, and Z'_v is the embedding of a training node v in V_T. $\mathcal{L}$ can also be specified to minimize a task-specific loss.

2.1 Robustness of GNN-based Classifier

We start with a characterization of robustness that extends certifiable robustness [Bojchevski and Günnemann, 2019], which verifies if predicted labels can be changed by a perturbation of size up to p.

(p,θ)-robustness. Given $G = (V, E)$, a GNN-based classifier M with logits Z, a set of *targeted* nodes $V_T \subseteq V$, a number p, and a vulnerable set $E_p \subseteq (V \times V)$, a pair (G, M) is *robust* at a node $v \in V_T$ and E_p, if a "worst-case margin" $m^*_{y_t,*}(v) = \min_{c \neq y_t} m^*_{y_t,c}(v) > 0$, where y_t is the true label of v, and c is any other label ($c \neq y_t$). Here $m^*_{y_t,c}(v)$ is defined as:

$$
\begin{aligned}
m^*_{y_t,c}(v) &= \min_{\tilde{G} \in G \cup E_p} m_{y_t,c}(v) \\
&= \min_{\tilde{G} \in G \cup E_p} \pi_{\tilde{G}}(v)^T \left(Z_{\{:,y_t\}} - Z_{\{:,c\}} \right)
\end{aligned}
$$

where $\tilde{G}$ ranges over all the possible graphs obtained by applying perturbation of up to p links from the vulnerable

set E_p, and $\pi_{\tilde{G}}(v) = \mathbf{\Pi}_{v,:}$ is the PageRank vector of node v in the PageRank matrix $\mathbf{\Pi} = (1-\alpha)(I_N - \alpha D^{-1}A)^{-1}$. Here D is the diagonal matrix of node out-degrees with $D_{ii} = \sum_j A_{ij}$, I_N is an identity matrix, and α is teleport probability.

By verifying $m^*_{y_t,*}(v) > 0$ (*i.e.,* $m^*_{y_t,c}(v) > 0$ for any $c \in L(v)$ other than the correct label of v), it indicates that under any links manipulation of size p over E_p, M always predicts the label of node v as y_t *w.r.t.* the logits Z.

We say (G, M) is (p, θ)-*robust w.r.t.* V_T and E_p, if (G, M) is robust for at least θ fraction of V_T ($\theta \in [0, 1]$) under any perturbations of size at most p over E_p ($p \leq |E_p|$).

Verification Given a pair (G, M), vulnerable set E_p, target nodes V_T and p, the (p, θ)-verification problem is to decide if (G, M) is (p, θ)-robust *w.r.t.* E_p and V_T.

Lemma 1: *The (p, θ)-verification problem is NP-hard even for fixed θ and E_p.* □

Proof sketch: We can show that it is already NP-hard to verify a special case when $\theta = 1$ and $p = |E_p|$. The lower bound of the latter follows from a reduction from the link building problem [Bojchevski and Günnemann, 2019; Olsen *et al.*, 2012], which maximizes the PageRank of a given target node in a graph by adding k new links. □

We outline an algorithm to verify the (p, θ)-robustness of a pair (G, M). The algorithm verifies if (G, M) is robust at up to θ fraction of the nodes in V_T given E_p. Specifically, it invokes a policy iteration [Bojchevski and Günnemann, 2019] to compute a set of optimal links W_k from E_p such that minimizes $m^*_{y_t,*}(v)$ if perturbed. Each policy induces a perturbed graph. For any pair of labels c_1,c_2 of node v and E_p, it greedily selects edges that improve the policy (lower the robustness of node v) and converge to W_k that forms $\bar{G}$, such that $min_{\bar{G} \in G \cup E_p} \pi_{\bar{G}}(v)^T \left(Z_{\{:,y_t\}} - Z_{\{:,c\}}\right)$ is obtained.

Monotonicity property. We next show a monotonicity property of (p, θ)-robustness in terms of the vulnerable set E_p.

Theorem 2: *A (p, θ)-robust pair (G, M)* w.r.t. *E_p and V_T remains to be (p, θ)-robust for V_T and any $E'_p \subseteq E_p$.* □

Proof sketch: We prove the result by contradiction. Assume (G, M) is not robust at v over E'_p, then there is a specific label $c_v \neq y_t$ (y_t is the predicted label of v), and a perturbed graph G' obtained by perturbing at most p links in E'_p, such that (a) $m^*_{y_t,*}(v) = m^*_{y_t,c_v}(v) \leq 0$, and (b) $\pi_{G'}(v)^T \left(Z_{\{:,y_t\}} - Z_{\{:,c_v\}}\right) \leq 0$. For each such G', we can construct a perturbed graph G'' which bear a perturbation that leads to the change of the label of v over E_p, which contradicts that (G, M) is robust at v *w.r.t.* E_p. As (G, M) is robust for at least θ fraction of V_T *w.r.t.* E_p, it remains (p, θ)-robust *w.r.t.* V_T and any $E'_p \subseteq E_p$ (detailed proof in [Guan *et al.*, 2022]). □

2.2 Robustification Problem

Theorem 2 tells us that it is desirable to compute a smallest set of links $\Delta L \subset E_p$ that should be "protected" from the vulnerable set, up to a point that (G, M) becomes robust for a desirable fraction of targeted nodes over $E_p \setminus \Delta L$. Indeed, (1) protecting any set larger than ΔL will not be necessary (unless a new threshold $\theta' > \theta$ is required by user); and (2) ensuring robustness for entire V_T can be an overkill for finding models that are "robust enough", and may cause expensive defending cost, even if $(p, 1)$-robustness is achievable.

We formalize a pragmatic optimization problem, called (p, θ)-*robustification* as follows.

- **Input**: a pair (G, M), target nodes $V_T \subseteq V$, vulnerable set E^p, constants p and θ ($p \leq |E_p|; \theta \in [0, 1]$).
- **Output**: a triple $(G', M', \Delta L)$, such that (1) (G', M') is (p, θ)-robust *w.r.t.* V_T and $E_p \setminus \Delta L$; and (2)ΔL is a smallest subset of E_p that ensures (1).

Although desirable, the problem is nontrivial (NP-hard) even when M and E_p are fixed, given the hardness of the verification problem. We next introduce (1) a feasible algorithm to compute a small *protection set* ΔL such that (G, M) is (p, θ) robust *w.r.t.* $E_p \setminus \Delta L$, and (2) a co-learning framework that incorporates protection set computation, verification, and robust learning to robustify (G, M) to (G', M').

3 Computing Protection Set

We first develop an algorithm with a goal to compute a smallest protection set $\Delta L \subseteq E_p$ such that (G, M) is (p, θ)-robust *w.r.t.* V_T and $E_p \setminus \Delta L$. An exact algorithm that enumerates subsets of E_p and verifies the model robustness (Section 2.1) is expensive when G is large. We introduce a fast heuristic optimized by a traversal-based greedy selection strategy.

Algorithm. The algorithm, denoted as minProtect and illustrated in Algorithm 1, keeps track of the following auxiliary structures: (1) a set $V_u \subseteq V_T$, which contains the target nodes at which (G, M) is currently not robust, (2) a vector $M^*_{y_t,*}$, where each of its entry records $m^*_{y_t,*}(v_u)$ for each node $v_u \in V_u$, and (3) the current fraction $\theta' = 1 - \frac{|V_u|}{|V_T|}$. In addition, each node v_u has a Boolean flag that indicates whether it is inspected by PrioritizeT. It initializes ΔL and V_u (line 1), and iteratively performs three major steps.

- Target prioritization (procedure PrioritizeT) estimates and selects a next target node v in V_u at which (G, M) is most likely to be robust upon augmenting ΔL with small amount of links (line 8);
- Protection augmentation (procedure UpdateL) augments ΔL with a set of new links in E_p, computed by traversing from the selected target node v_u (line 9);
- Verification (procedure VerifyM), that verifies (p, θ) robustness of (G, M) (line 3).

The above process repeats until a protection set ΔL is identified that enable a (p, θ)-robust pair (G, M) (Theorem 2), or $\Delta L = E_p$ (line 2). As UpdateL may make (G, M) robust at multiple nodes in V_T, minProtect early terminates once VerifyM asserts the desired (p, θ) robustness (lines 5-6).

We next introduce the three procedures.

Procedure VerifyM. Procedure VerifyM nontrivially optimizes the policy iteration procedure [Bojchevski and

Algorithm 1 minProtect

Input: pair (G, M), vulnerable set E_p, target nodes V_T, constants p and θ;
Output: a protection set ΔL;

1: set $\Delta L := \emptyset$; $\theta' := 0$; list $V_u := \emptyset$;
2: **while** $\theta' < \theta$ **and** $\Delta L \neq E_p$ **do**
3: $\theta' :=$ VerifyM $((G, M), E_p, p, \Delta L))$;
4: $V_u := \{v_u | v_u \in V_T$ and (G, M) is not robust at $v_u\}$;
5: **if** $(\theta' \geq \theta)$ **then**
6: **return** ΔL;
7: **while** there is an unvisited node in V_u **do**
8: $v :=$ PrioritizeT $(M^*_{y_t,*}, V_u)$;
9: $\Delta L := \Delta L \cup$ UpdateL $(v, (G, M), E_p, p, \Delta L)$;
10: **return** ΔL;

Procedure: UpdateL $(v_u, (G, M), E_p, p, \Delta L)$

1: set $N_d(v_u) := \emptyset$; set $\Delta L' := \emptyset$; heap $v_u.H := \emptyset$;
2: **while** there is an unvisited link $(u, u') \in N_d(v_u) \cap \{E_p \setminus \Delta L\}$ **do**
3: initializes v_u.H with (u, u');
4: updates worst-case margin $m^*_{y_t,*}(v_u)$;
5: set $\Delta L'$ as all links (u, u') in $v_u.H$ that ensures a maximum worst-case margin.
6: **return** $\Delta L'$;

Günnemann, 2019] to test if (G, M) is (p, θ)-robust at current $E_p \setminus \Delta L$. (1) It first computes a value $m^*_{c_1,c_2}(v)$ for each node $v \in V_T$ and derives a set of optimal links W_k $(|W_k| \leq p)$ over $E_p \setminus \Delta L$ and for any pair of labels c_1 and c_2, such that W_k is most likely to minimize the worst-case margin of node v. It returns $K \times K$ pairs of W_k, where K is the size of label set. (2) For each node $v_u \in V_u$ and its predicted label y_t by M (often set as the true label), it computes $m^*_{y_t,c}(v_u)$ and updates the PageRank vector $\pi_{\tilde{G}}(v)$ over $\tilde{G}$. Here $\tilde{G}$ is obtained by flipping all pairs $(v, v') \in W_k$. If $m^*_{y_t,*}(v_u) = \min_{c \neq y_t} m^*_{y_t,c}(v_u) \leq 0$, it asserts that (G, M) is not robust at v_u. It then updates V_u, and returns $\theta' = 1 - \frac{|V_u|}{|V_T|}$. If $\theta' \geq \theta$, then (G, M) is (p, θ)-robust *w.r.t.* $E_p \setminus \Delta L$ and V_T.

Procedure PrioritizeT. PrioritizeT consults the values $m^*_{y_t,*}(\cdot)$ (obtained from procedure VerifyM) of each node $v_u \in V_u$, dynamically reranks V_u following the descending order of $m^*_{y_t,*}(\cdot)$, and selects the next node v with the current largest $m^*_{y_t,*}(v)$ ($m^*_{y_t,*}(v) < 0$ for any $v \in V_u$). Intuitively, it indicates that v is likely to be the next node at which (G, M) becomes robust as more links are protected to the current ΔL.

Procedure UpdateL. Given a target node $v_u \in V_u$, UpdateL augments ΔL with new links to be "protected", such that (G, M) is likely to be robust at v_u. Our idea is to "rehearse" the protection of single links *near* v_u, and greedily augment ΔL with (u, u') whose protection best mitigates the impact against a "worst case" perturbation (obtained by perturbing all $E_p \setminus \Delta L$ but (u, u')). This can be achieved by ranking the links following a descending order of their resulting worst margin $m^*_{y_t,*}(v)$ of v_us. Intuitively, "protecting" (u, u') maximally improves the worst margin of v_u (hence likely to make M robust at v_u), thus (u, u') should be selected.

We say a link (v, v') is in d-hop neighborhood $(d \geq 1)$ of a node v_u (denoted as $(v, v') \in N_d(v_u)$) if there is a sequence of d links $(v_0, v_1), \ldots (v_{d-1}, v_d)$, such that $v_u = v_0$, $v_{d-1} = v$, $v_d = v'$, and $(v_i, v_{i+1}) \in E_p$ for $i \in [0, d-1]$. For each $v_u \in V_u$, UpdateL maintains a heap $v_u.H$. Each entry in $v_u.H$ contains (a) a link $(v, v') \in N_d(v_u)$, (b) a graph $G_{E_p \setminus (v,v') \cup \Delta L}$, obtained by perturbing all links in $E_p \setminus \Delta L$ but (v, v'), and (c) the worst-case margin $m^*_{y_t,*}(v)$ determined by $G_{E_p \setminus (v,v') \cup \Delta L}$ (Section 2.1).

Given a node $v_u \in V_u$ selected by PrioritizeT, UpdateL starts a breadth first traversal and explores up to $N_d(v_u)$ (d is set as the smallest integer value such that $N_d(v_u) \cap \{E_p \setminus \Delta L\} \neq \emptyset$ by default). During the traversal, it dynamically inserts unvisited link $(v, v') \in N_d(v_u)$. For each visited (v, v'), it initializes the entry $v_u.H$, and computes the worst-case margin. For all the links in $N_d(v_u)$, it selects the link (v, v') with the largest worst-case margin in $v_u.H$ and adds it to ΔL. This process repeats until no new links can be found.

Analysis. Algorithm minProtect correctly returns a protect set ΔL that either ensures (p, θ)-robust pair (G, M) *w.r.t.* $E_p \setminus \Delta L$ and V_T, or a counterpart that ensures a largest fraction θ' of V_T at which (G, M) is robust when terminates. This is ensured by several invariants below. (1) VerifyM performs policy iteration [Bojchevski and Günnemann, 2019] that converges to optimal perturbations over $E_p \setminus \Delta L$ and correctly computes $m^*_{y_t,c}(v)$ to verify model robustness. (2) UpdateL augments ΔL in a non-decreasing manner, which ensures the termination of minProtect (Theorem 2). (3) PrioritizeT does not miss nodes at which (G, M) is not robust.

Optimization. A main bottleneck is the computation of the matrix inverse operation [Ma *et al.*, 2021]), for computing $m^*_{c_1,c_2}(v)$ (VerifyM, line 3 of minProtect) and $m^*_{y_t,*}(v)$ (UpdateL). To reduce the cost, we leverage approximate computation [Bojchevski *et al.*, 2020] to approximate the dynamically maintained adjacency matrix A' with a sparse matrix $\mathbf{\Pi}^{\epsilon}$ that approaches $(1-\alpha)(I_N - \alpha D^{-1} A')^{-1}$ (see [Guan *et al.*, 2022] for details).

4 RoboGNN: A Co-learning Framework

We next present RoboGNN, a co-learning framework to robustify (G, M) towards (p, θ)-robustness. RoboGNN (illustrated in Algorithm 2) generates a triple ($<S', A'>$, M', ΔL), where S' is a learned graph structure representation of G' and A' is the "recovered" adjacency matrix of G'.

The framework RoboGNN iteratively improves (G, M) by interleaving two processes, consistently towards improved robustness: (1) For a fixed graph A' (initialized as A), computing ΔL to refine E_p (lines 5-8) as in ("invoking") minProtect, and (2) Jointly improves structure representation $<S', A'>$ (lines 9-11) and M (lines 12-13) in the "context" of current vulnerable set $E_p \setminus \Delta L$. Before fixing graph structure and alternating updating model parameters, RoboGNN updates G' in UpdateA (line 11) by fixing the inconsistency between the adjacency matrix A' of G' and ΔL (*e.g.,* if (u, u') is an edge in ΔL but not in A', RoboGNN inserts (u, u') to A'). It eventually verifies the modified M' over $E_p \setminus \Delta L$ (line 14), and returns the triple ($<S', A'>$, M', ΔL) whenever (p, θ)-robustness is achieved, or no link can be added to ΔL (line 4).

Algorithm 2 RoboGNN

Input: pair (G, M), vulnerable set E_p, target nodes V_T, constants p and θ;
Output: triple ($<S', A'>$, M', ΔL);

1: Initialize $<S', A'>:=<A, A>$; $\theta':=0$, $M':=M$, $\Delta L:=\emptyset$;
2: θ' := VerifyM $((G', M'), E_p, p, \Delta L))$;
3: $V_u := \{v_u | v_u \in V_T$ and (G', M') is not robust at $v_u\}$;
4: **while** $\theta' < \theta$ **and** $|\Delta L| < |E_p|$ **do**
5: update V_u and visiting status of nodes in V_u;
6: **while** there is an unvisited node in V_u **do**
7: v := PrioritizeT $(M^*_{y_t,*}, V_u)$;
8: $\Delta L := \Delta L \cup$ UpdateL $(v, (G', M'), E_p, p, \Delta L)$;
9: **for** i =1 to ς **do**
10: $\mathrm{S}' := \mathrm{S}' - \eta \nabla_{S'} (\| \mathrm{S}' - \mathrm{A} \|_F^2 + \lambda \mathcal{L})$;
11: $A' :=$ UpdateA$(A', \Delta L)$;
12: **for** i =1 to τ **do**
13: $M' := M' - \eta' \nabla_{M'} \mathcal{L}$;
14: θ' := VerifyM $((G', M'), E_p, p, \Delta L))$;
15: **return** ($<S', A'>$, M', ΔL);

	Cora	Citeseer	Pubmed
# Nodes	2,708	3,327	19717
# Edges	5,429	4,732	44338
# Features per Node	1,433	3,703	500
# Classes	7	6	3
# Training Nodes	140	120	60
# Validation Nodes	500	500	500
# Test Nodes	1,000	1,000	1000
# $\lvert E_p \rvert$	1650	1200	376
(p, θ)	(177,0.95)	(848,0.95)	(370,1.0)

Table 1: Settings: Datasets, training, and robustification

Robust cross-entropy loss. RoboGNN co-learns S' and M' by consistently minimizing a hinge loss penalty, which aims to enforce (S', M'), *w.r.t.* current vulnerable set $E_p \setminus \Delta L)$, to be robust at the nodes by ensuring a margin of at least positive threshold m. Specifically, the robust loss is defined as:

$$\mathcal{L} = \sum_{v \in \mathcal{V}_T} [\mathcal{L}_{CE}(y^*_v, \pi_{\tilde{G}}(v)^T Z) + \sum_{c \in L(v), c \neq y^*_v} \max(0, m - m^*_{y_v, c}(v))].$$

It then learns S' by minimizing a weighted combination of $\mathcal{L}$ and feature difference (lines 9-10), and M' by minimizing $\mathcal{L}$.

5 Experiments

We next experimentally verify the effectiveness of RoboGNN on improving the robustness and accuracy of GNN-based classification, the learning cost, and the impact of parameters.

Experiment Setting. We use three real-world datasets: Cora [McCallum *et al.*, 2000], Citeseer [Giles *et al.*, 1998] and Pubmed [Sen *et al.*, 2008]. Each node has features derived from a bag-of-words representation of the document it refers to, and a class label denoting topic area (see Table 1).

Generation of graphs G. We adopt a mixture of adversarial strategies, including non-targeted attacks [Zügner and Günnemann, 2019], random perturbation [Yuan *et al.*, 2017], and property-preserving link attacks(that aim to maintain degree distribution) [Zügner *et al.*, 2018]. These attacks are designed under the same principle to minimize the probability of correct class prediction. For each dataset, we manipulate at most 30% edges to produce a graph G as input graph for RoboGNN. This suffices to cause performance degradation of GNNs if learned from G [Zügner and Günnemann, 2019].

We generate vulnerable sets E_p with random-walk based sampling [Guan *et al.*, 2022]. The generation of E_p and RoboGNN learning do not assume prior knowledge.

Generate classifiers M. We use the following GNN-based classifiers as input. (1) GCN [Kipf and Welling, 2016], (2) GAT [Veličković *et al.*, 2017], and (3) π-PPNP, a class of GNNs that decouples feature transformation from feature aggregation to optimize classification [Bojchevski and Günnemann, 2019]. We compare the accuracy and robustness of an input model M and its robustified counterpart M'.

We also evaluate RoboGNN as an "end-to-end" framework, which directly learns a robust model from scratch (*i.e.,* generate (G', M') given $(G, \emptyset)$), with the following baselines. (1) certPPNP [Bojchevski and Günnemann, 2019] learns a more robust counterpart of π-PPNP by robust training [Bojchevski and Günnemann, 2019]; and (2) Pro-GNN [Jin *et al.*, 2020], which learns graph representations and GNNs from scratch. In addition, we develop a variant as U_RoboGNN, by removing the optimization on pagerank.

Configuration. We train a two-layer network for all the input models with the same set of hyper-parameters settings (*e.g.,* dropout rate, number of hidden units). The training epoch number is set as 300. For each dataset, we fix the learning rate for Pro-GNN, certPPNP, and RoboGNN. The configuration of input GCN, GAT and Pro-GNN are calibrated to yield consistent and best performance over benchmark metrics as in [Kipf and Welling, 2016; Veličković *et al.*, 2017; Jin *et al.*, 2020]. we report the average accuracy (acc.) for multiclass classification. All Experiments are executed on a Unix environment with GPU Nvidia P-100.

The source code and datasets are available[1].

Experimental Results. We next present our findings.

Exp-1: Effectiveness of Robustification. We first evaluate RoboGNN on improving the accuracy of input models. Table 3 reports the results using GCN and π-PPNP. Here RoboGNN (GCN) and RoboGNN (π-PPNP) shows the counterparts M' over G for the same set of test nodes. (1) During the co-learning, RoboGNN ensures an increasing robustness of the improved model compared with a previous counterpart, in all cases (not shown). (2) The improved robustness in turn significantly improves the accuracy of input models over test nodes. For example, RoboGNN achieves on average 45.3% (resp. 31%) gains on F_1 for GCN (resp. π-PPNP). We find that the robustified M' over G' yields more consistent label prediction, and better approaches to the performance of "yardstick" models learned from original (unknown) graphs that are not perturbed. ([Guan *et al.*, 2022] for more analysis).

Impact of factors. We next evaluate the impact of perturbation size and configurations of robustness to the effectivenes. We

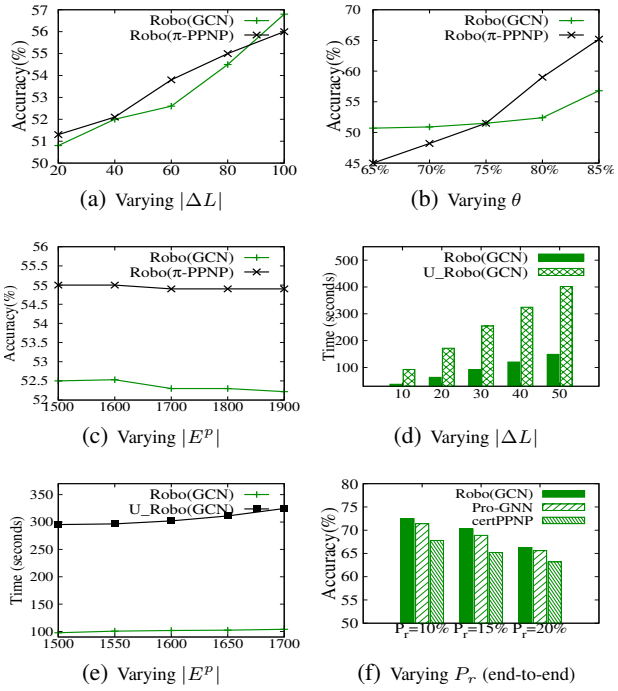
[1] https://github.com/CWRU-DB-Group/robognn

dataset	Cora		Citeseer		Pubmed	
metrics	acc.	F_1	acc.	F_1	acc.	F_1.
GCN (A)	71.3%	71.42%	51.0%	48.80%	64.1%	63.19%
GCN(A')	73.1%	72.83%	56.1%	53.79%	65.0%	63.99%
GAT (A)	74.8%	73.68%	65.0%	61.06%	63.7%	62.79%
GAT(A')	76.7%	76.69%	65.3%	62.05%	63.7%	62.79%

Table 2: Improved graph structure A' benefits GNN learning. Bold: models learned with A' from scratch.

dataset	Cora		Citeseer		Pubmed	
metrics	acc.	F_1	acc.	F_1	acc.	F_1.
GCN	44.1%	44.89%	39.7%	38.80%	49.7%	48.49%
RoboGNN(GCN)	76.7%	75.65%	54.0%	51.66%	65.4%	63.99%
π-PPNP	43.0%	43.64%	50.0%	48.48%	40.8%	34.03%
RoboGNN(π-PPNP)	75.9%	74.95%	56.9%	54.58%	43.9%	36.18%

Table 3: Robustify GNN models with RoboGNN framework. Bold: robustified models.



(a) Varying $|\Delta L|$ (b) Varying θ (c) Varying $|E^p|$ (d) Varying $|\Delta L|$ (e) Varying $|E^p|$ (f) Varying P_r (end-to-end)

Figure 1: Performance: accuracy and efficiency

report the results over Cora. The results from other datasets are consistent (see [Guan *et al.*, 2022] for details).

Impact of $|\Delta L|$. Using the default setting in Table 1, we vary the size of allowed protection set from 20 to 100, and terminate RoboGNN whenever ΔL reaches a certain size. Fig. 1(a) tells us that RoboGNN (1) can effectively improve the accuracy of input models (from 51% to 57%) as more links are protected, (2) ensures a desirable (p, θ)-robustness, and to achieve these, (3) explicitly suggests only a small set ($\leq$ 100, 10% of vulnerable set) of links to be protected.

Impact of θ. Fixing other parameters as default, we vary θ from 65% to 85% (Fig. 1(b)). (1) Ensuring robustness at more target nodes improves the accuracy, which is consistent with our observation in Fig. 1(a). (2) RoboGNN effectively responses to different robustness requirements. It improves the accuracy of π-PPNP from 45% to 65% by ensuring a $(150, 85\%)$-robust model from a $(150, 65\%)$ counterpart.

Impact of $|E_p|$.Fixing other parameters, we vary the size of vulnerable set from 1500 to 1900. Fig. 1(c) shows that it becomes more difficult for RoboGNN to maintain the robustness. Indeed, larger E_p indicates more adversarial perturbations to prevent robustness for the target nodes. On the other hand, its performance is not very sensitive, due to its ability to co-learn both models and graph representations that better mitigate the impact of perturbations.

Efficiency. Using the same default setting as Fig. 1(a) (resp. Fig. 1(c)), Fig. 1(d) (resp. Fig. 1(e)) verifies that RoboGNN takes more time to learn robustified models and graph representation with larger $|\Delta L|$ (resp. $|E_p|$). On the other hand, (1) optimization reduces the learning cost by 67% on average, and (2) the learning is less sensitive to $|E_p|$ given the early termination of minProtect as it augments ΔL (Theorem 2).

Exp-2: End-to-end performance. RoboGNN supports "end-to-end" learning of a robust model with desired robustness. When generating graph G, we vary the perturbation rate P_r, *e.g.,* the ratio of changed edges, from 10% to 20%, Fig. 1(f) shows that RoboGNN achieves best performance improvement due to robustness guarantees, among all baselines.

Exp-3: Usability of protection set. We also evaluate how ΔL can be used to suggest "corrections" of adjacency matrix A to improve GNNs. Given an original graph G, we first perform adversarial perturbation and derive a perturbed adjacency matrix A. We then use RoboGNN to obtain ΔL over a robustified counterpart (G', M'). Table 2 verifies that ΔL can effectively suggest "recovered' adjacency matrix which directly leads to the training of more accurate models. This indicates the application of RoboGNN in explicit link correction.

6 Conclusion

We have proposed a novel framework, RoboGNN, that can improve the robustness of GNN-based classification and also suggest desired graph structures under link perturbations. Our experimental study confirms that RoboGNN achieves such effects. A future topic is to enable RoboGNN for link repairing to improve downstream GNN-based tasks.

Acknowledgments

This work is supported by NSF under CNS-1932574, OIA-1937143, ECCS-1933279, CNS-2028748, OAC-2104007 and DoE under DE-EE0009353.

References

[Bojchevski and Günnemann, 2019] Aleksandar Bojchevski and Stephan Günnemann. Certifiable robustness to graph perturbations. In *NeurIPS*, 2019.

[Bojchevski *et al.*, 2020] Aleksandar Bojchevski, Johannes Klicpera, Bryan Perozzi, Amol Kapoor, Martin Blais, Benedek Rózemberczki, Michal Lukasik, and Stephan Günnemann. Scaling graph neural networks with approximate pagerank. In *KDD*, 2020.

[Cai and Wang, 2020] Chen Cai and Yusu Wang. A note on over-smoothing for graph neural networks. *arXiv preprint arXiv:2006.13318*, 2020.

[Dai *et al.*, 2018] Hanjun Dai, Hui Li, Tian Tian, Xin Huang, Lin Wang, Jun Zhu, and Le Song. Adversarial attack on graph structured data. *arXiv preprint arXiv:1806.02371*, 2018.

[Entezari *et al.*, 2020] Negin Entezari, Saba A Al-Sayouri, Amirali Darvishzadeh, and Evangelos E Papalexakis. All you need is low (rank) defending against adversarial attacks on graphs. In *WSDM*, 2020.

[Giles *et al.*, 1998] C. Lee Giles, Kurt D. Bollacker, and Steve Lawrence. Citeseer: An automatic citation indexing system. In *Proceedings of the Third ACM Conference on Digital Libraries*, 1998.

[Gori *et al.*, 2005] Marco Gori, Gabriele Monfardini, and Franco Scarselli. A new model for learning in graph domains. In *Proceedings. 2005 IEEE international joint conference on neural networks*, volume 2, pages 729–734, 2005.

[Guan *et al.*, 2022] Sheng Guan, Hanchao Ma, and Yinghui Wu. Robognn: Robustifying node classification under link perturbation, full version. https://github.com/CWRU-DB-Group/robognn/blob/main/full.pdf, 2022. Accessed: 2022-06-08.

[Hamilton *et al.*, 2017] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, 2017.

[Harris and Harris, 2010] David Harris and Sarah Harris. *Digital design and computer architecture*. Morgan Kaufmann, 2010.

[Ioannidis and Giannakis, 2019] Vassilis N Ioannidis and Georgios B Giannakis. Edge dithering for robust adaptive graph convolutional networks. *arXiv preprint arXiv:1910.09590*, 2019.

[Jin *et al.*, 2020] Wei Jin, Yao Ma, Xiaorui Liu, Xianfeng Tang, Suhang Wang, and Jiliang Tang. Graph structure learning for robust graph neural networks. *KDD*, 2020.

[Kipf and Welling, 2016] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[Ma *et al.*, 2021] Yao Ma, Xiaorui Liu, Tong Zhao, Yozen Liu, Jiliang Tang, and Neil Shah. A unified view on graph neural networks as graph signal denoising. In *CIKM*, 2021.

[McCallum *et al.*, 2000] Andrew Kachites McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. Automating the construction of internet portals with machine learning. *Information Retrieval*, 2000.

[Olsen *et al.*, 2012] Martin Olsen, Anastasios Viglas, and Ilia Zvedeniouk. An approximation algorithm for the link building problem. *arXiv preprint arXiv:1204.1369*, 2012.

[Paulheim, 2017] Heiko Paulheim. Knowledge graph refinement: A survey of approaches and evaluation methods. *Semantic web*, 2017.

[Sen *et al.*, 2008] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 2008.

[Tran *et al.*, 2017] Cong Tran, Won-Yong Shin, and Andreas Spitz. Community detection in partially observable social networks. *arXiv preprint arXiv:1801.00132*, 2017.

[Veličković *et al.*, 2017] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[Wu *et al.*, 2019] Huijun Wu, Chen Wang, Yuriy Tyshetskiy, Andrew Docherty, Kai Lu, and Liming Zhu. Adversarial examples on graph data: Deep insights into attack and defense. *IJCAI*, 2019.

[Wu *et al.*, 2020] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *NeurIPS*, 2020.

[Xu *et al.*, 2021] Zhe Xu, Boxin Du, and Hanghang Tong. Graph sanitation with application to node classification. *arXiv preprint arXiv:2105.09384*, 2021.

[Yuan *et al.*, 2017] Xiaoyong Yuan, Pan He, Qile Zhu, Rajendra Rana Bhat, and Xiaolin Li. Adversarial examples: Attacks and defenses for deep learning. corr abs/1712.07107 (2017). *arXiv preprint arXiv:1712.07107*, 2017.

[Zügner and Günnemann, 2019] Daniel Zügner and Stephan Günnemann. Adversarial attacks on graph neural networks via meta learning. *ICLR*, 2019.

[Zügner *et al.*, 2018] Daniel Zügner, Amir Akbarnejad, and Stephan Günnemann. Adversarial attacks on neural networks for graph data. In *KDD*, 2018.