RoboGNN: Robustifying Node Classification under Link Perturbation

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 and 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) [7] have demonstrated good performance for graph representation learning and downstream classification tasks. GNNs adopt a label propagation architecture to learn discriminative node embeddings with multiple graph convolutional layers. In each layer, the embedding of a node v is updated by aggregating its counterparts from the neighbors of v.

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 [17, 19] or malicious adversarial attacks [4, 26]. For example, a GNN-based classifier M can be sensitive under a set of link perturbation 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 $\Delta L \subseteq E_q$ to be "protected" from attacks to ensure the robustness. In addition, the size $|\Delta L|$ reflects "defense effort" (*e.g.,* cost to 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 [1] (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.

Robustness measure. 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.

Monotonicity property. 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. Our strategy dynamically ranks V_T based on the likelihood that the model is robust at single nodes, and incrementally augment ΔL following efficient traversal from the node.

Co-learning framework. Based on the verification and minimality condition, we present RoboGNN, a co-learning framework to robustify an input model with guaranteed robustness. RoboGNN jointly learns the graph representation and GNNs iteratively towards (p, θ) robustness with an optimization 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. We summarize the related work as follows.

Robustness models. Certifiable robustness is introduced in [1]. 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 robust training that minimizes a hinge loss penalty. We study (p, θ)-robustness that extends certifiably robustness with configurable p and θ to support the need of robustification in practice.

Defense of GNN *models*. Several defense methods have been studied to mitigate the impact of adversarial structural attacks, which typically summarize the (assumed) changes of specific graph properties before and after the attack, and learn graph representations to mitigate the impact. GCN-Jaccard [21] removes malicious links added to nodes with dissimilar features measured by Jaccard similarity. GCN-SVD [5] assumes an attack model [25] that affects high-rank singular components of the graph and performs the low-rank approximation for the graph reconstruction to mitigate the effects. Edge dithering [11] generates auxiliary graphs with edge insertions and deletions against adversarial perturbations to facilitate robust learning. Graph sanitation [23] solves a bilevel optimization problem that aims to modify perturbed graphs to improve underlying semi-supervised learning. Pro-GNN [12] integrates graph properties *e.g.,* sparsity, low rank, and feature smoothness to its loss function and learns to clean the graph.

In contrast to prior work, our approach takes a different strategy that aim 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 $E \subseteq V \times V$. 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 [10]); and A is the adjacency matrix of G. A *link* in G is a node pair $(v, v') \in V \times V$.

Perturbations. 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 simply 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 $V_{\mathcal{T}} \subseteq V$, 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 [22] 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 [9] that samples fixed-size neighbors (c=0,n=1) and GAT [20] that incorporates self-attention for neighbors. Specifically, we make case for GNNs that leverages personalized PageRank, which mitigates over-smoothing [3]. In such models, $Z = \mathbf{\Pi} \mathbf{h}^n$, 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 $\mathcal{L}_{CE}(Z, A) = -\sum_{v \in V_{\mathcal{T}}} Y_v \ln Z'_v$, where Y_v is the label of a training node $v \in V_{\mathcal{T}}$, and Z'_v is the embedding of a training node v in $V_{\mathcal{T}}$. 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 [1], 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 θ = 1 and $p = |E_p|$. The lower bound of the latter follows from a reduction from the link building problem [1, 16], 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 algorithm [1] 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 $E'_p \subseteq E_p$ (see detailed proof in Appendix). □

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. The hardness follows from the intractability of the verification problem. Despite the hardness, 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

Given a pair (G, M), vulnerable set E_p, and a constant θ, 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 verify 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 Fig. 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 7);
- 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 8);
- 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). We remark that each UpdateL may make (G, M) robust at multiple nodes in V_T. minProtect hence early terminates once VerifyM asserts the desired (p, θ) robustness, and returns the current ΔL (line 5).

We next introduce the three procedures.

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

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

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

Figure 1: Algorithm minProtect

Procedure VerifyM. Procedure VerifyM nontrivially optimizes the policy iteration procedure [1] 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 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 by definition.

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 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 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 $G_{E_p \setminus (v,v')}$, obtained by perturbing all links in E_p but (v, v'), and (c) the worst-case margin $m^*_{y_t,*}(v)$ determined by $G_{E_p \setminus (v,v')}$ (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 = 4 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 one (v, v') with the largest worst-case margin in $v_u.M$ and adds (v, v') to ΔL. This processes repeats until no new links can be found.

Analysis. Algorithm minProtect correctly returns a protect set ΔL that either ensures a (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 correctly performs policy iteration [1] that converge to the optimal perturbations over E^p and correctly computes $m^*_{y_t,c}(v)$ to verify the 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 [14]), 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 [2] to approximate the dynamically maintained adjacency matrix A' with a sparse matrix $\mathbf{\Pi}^{\varepsilon}$ that approaches $(1-\alpha)(I_N - \alpha D^{-1} A')^{-1}$.

Algorithm RoboGNN

Input: pair (G, M), vulnerable set E_p, target nodes V_T,
constants p and θ;
Output: triple $(S, M', \Delta L)$ with learned structure S of G';

Initialize $S := A$, $\theta' := 0$, $M' := M$, $\Delta L := \emptyset$;
$\theta' :=$ VerifyM $((G, M'), E_p, p, \Delta L)$);
$V_u := \{v_u | v_u \in V_T$ and (G, M) is not robust at $v_u\}$;
while $\theta' < \theta$ **and** $|\Delta L| < |E_p|$ **do**
$\quad$ update V_u and visiting status of nodes in V_u;
$\quad$ **while** there is an unvisited node in V_u **do**
$\quad\quad$ $v :=$ PrioritizeT $(M^*_{y_t,*}, V_u)$;
$\quad\quad$ $\Delta L := \Delta L \cup$ UpdateL $(v, (G, M'), E_p, p, \Delta L)$;
$\quad$ **for** i =1 to ς **do**
$\quad\quad$ S := S $- \eta \nabla_S(\|$ S $-$ A$\|^2_F + \lambda \mathcal{L}_{CEM})$;
$\quad$ **for** i =1 to τ **do** $M' := M' - \eta' \nabla_{M'} \mathcal{L}_{CEM}$;
$\quad$ $\theta' :=$ VerifyM $((G, M'), E_p, p, \Delta L)$);
return $(S, M', \Delta L)$;

Figure 2: RoboGNN Co-learning framework

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 Fig. 2) generates a triple $(S, M', \Delta L)$, where S is a learned graph representation of G'.

	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	848	376
(p, θ)	(520,0.95)	(460,0.95)	(370,1.0)

Table 1: Settings: Datasets, training, and robustification

The framework RoboGNN iteratively improve (G, M) by interleaving two processes, consistently towards improved robustness: (1) for a fixed graph G, computing ΔL to refine E_p (lines 5-8) similarly as in minProtect, and (2) jointly improves graph representation S (lines 9-11) and M (lines 12) in the "context" of vulnerable set $E_p \setminus \Delta L$. It verifies the learned model M' over $E_p \setminus \Delta L$ (line 12), and returns the triple $(S, M', \Delta L)$ whenever (p, θ)-robustness is achieved, or no link can be added to ΔL (line 4).

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 cross-entropy loss is defined as:

$$\mathcal{L} = \sum_{v \in \mathcal{V}_T} [\mathcal{L}_{CE}(y_v^*, Z_v') + \sum_{c \in L(v), c \neq y_v^*} \max(0, m - m_{y_v, c}^*(v))].$$

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 used three real-world datasets: Cora [15], Citeseer [6] and Pubmed [18]. Each node has features derived from a bag-of-words representation of the document it refers to, and a class labeldenoting its topic area (*e.g.,* data mining, deep learning). The details of the datasets are summarized in Table 1.

Generation of graphs G. We adopt a mixture of adversarial manipulation strategies, including non-targeted attacks [26], random edge perturbation [24], and property-preserving link attacks(that aim to maintain degree distribution) [25]. These attacks are designed under the same principle to minimize the probability of the 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 a performance degradation of GNNs if learned from G [26].

We generate vulnerable sets E_p with random-walk based sampling. The generation of E_p and RoboGNN learning do not assume prior knowledge of these attack models.

Generate classifiers M. We use the following GNN-based classifiers as input. (1) GCN [13], (2) GAT [20], and (3) π-PPNP, a class of GNNs that decouple feature transformation from feature aggregation to optimize classification [1]. π-PPNP aims to maximizes the worst-case margin (while the attackers minimize it). (4) LP +GCN, which performs link prediction over G and learns GCN with enhanced G. We adopt node2vec [8] to perform link prediction. 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 [1] learns a more robust counterpart of π-PPNP by robust training [1]; and (2) Pro-GNN [12], which learns graph representations and GNNs from scratch. In addition, we develop a variant of RoboGNN, U_RoboGNN, by removing the optimization on pagerank matrix computation. RoboGNN-b, a variant of RoboGNN that assume a fixed graph G and only iteratively refines M but not refines graph structure S.

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 [12, 13, 20]. we report the average accuracy (acc.) for multiclass classification. All Experiments were executed on a Unix environment with GPU Nvidia K-80. Each test was run 5 times and the average results were reported.

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 4 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 a 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 found 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. Then, we compare RoboGNN (GCN) with other baseline methods as Table 4 shows.

Impact of factors. We next evaluate the impact of perturbation size and configurations of robustness to the effectiveness of robustification. We report the results over Cora. The results from other datasets are consistent.

Impact of $|\Delta L|$. Using the default setting in Table 1, we var-

[1] https://anonymous.4open.science/r/d11f9f34-bbc2-4449-9fd5-3f7896cec946/

dataset	Cora		Citeseer		Pubmed	
metrics	acc.	F_1	acc.	F_1	acc.	F_1.
GCN (A)	71.3%	71.42%	51.0%	48.8%	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.

dataset	Cora	Citeseer
metrics	acc.	acc.
GCN	$61.3 \pm 0.62\%$	$41.8 \pm 0.37\%$
LP +GCN	$68.3 \pm 1.70\%$	$47.0 \pm 0.80\%$
GAT	$67.7 \pm 0.31\%$	$\underline{51.8 \pm 0.90\%}$
Pro-GNN	$\underline{68.9 \pm 0.35\%}$	$50.4 \pm 1.30\%$
certPPNP	$65.2 \pm 1.06\%$	$47.2 \pm 0.62\%$
RoboGNN	$\mathbf{70.3 \pm 0.29\%}$	$\mathbf{53.3 \pm 0.60\%}$

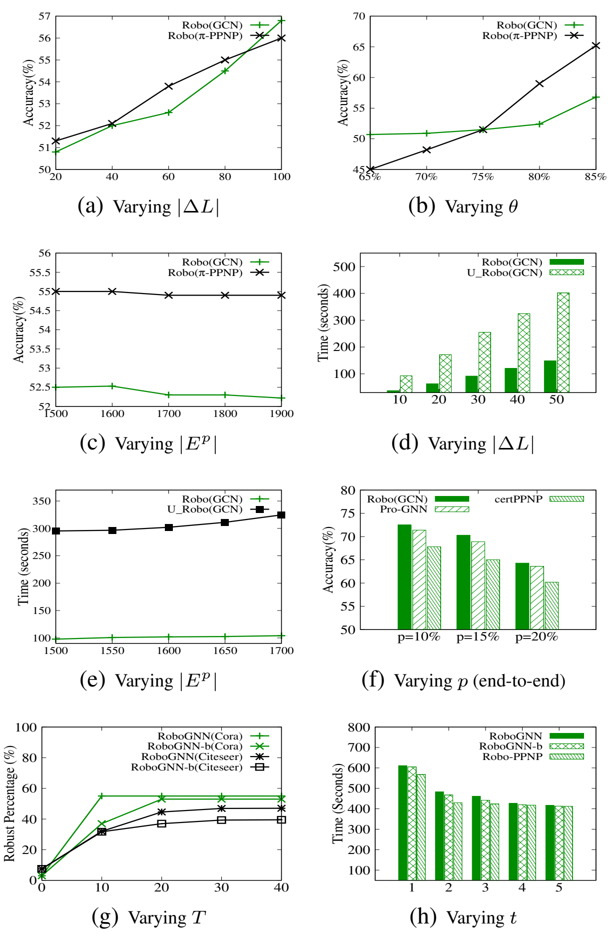
Table 4: Accuracy (under adversarial attack of 15% perturbed edges). Bold: best result; Underlined: second best.

ied the size of allowed protection set from 20 to 100, and terminate RoboGNN whenever ΔL reaches a certain size. Fig. 3(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 (≤ 100, 10% of vulnerable set) of links to be protected.

Impact of θ. Fixing other parameters as default, we varied θ from 65% to 85%. Fig. 3(b) verifies the following. (1) Ensuring model robustness at more target nodes improves the accuracy, which is consistent with our observation in Fig. 3(a). (2) RoboGNN can effectively response to different robustness requirement from users. For example, it improves the accuracy of π-PPNP from 45% to 65% by ensuring a more desirable $(150, 85\%)$-robustness from a $(150, 65\%)$ counterpart.

Impact of $|E_p|$.Fixing other parameters, we varied the size of vulnerable set from 1500 to 1900. Fig. 3(c) shows that it becomes more difficult for RoboGNN to maintain the robustness and the accuracy accordingly, as expected. Indeed, larger E_p indicates more adversarial perturbations exist to prevent model robustness for the same target nodes. On the other hand, the performance of RoboGNN is not very sensitive, due to its ability to co-learn both models and graph representations that better mitigate the impact of perturbations.

Impact of epoch T. Fig. 3(g) verifies that the robusteness of a model converges as the training epoch goes up. This is consistent with the definition of the robustness, which is determined by the worst-case margin of targeted nodes. On the other hand, RoboGNN converges faster with fewer epoch



(a) Varying $|\Delta L|$ (b) Varying θ

(c) Varying $|E^p|$ (d) Varying $|\Delta L|$

(e) Varying $|E^p|$ (f) Varying p (end-to-end)

(g) Varying T (h) Varying t

Figure 3: Performance: accuracy and efficiency

numbers on both Cora and Citeseer, due to the co-learning of both graph structures and robust models.

Efficiency. Using the same default setting as Fig. 3(a) (resp. Fig. 3(c)), Fig. 3(d) (resp. 3(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) the 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 from scratch. We set $E_p = E$, and p an upper bound of perturbation size. Varying p to be 5% to 25% of $|E_p|$, Fig. 3(f) tells us that RoboGNN achieves best performance improvement due to robustness guarantees, among all baseline methods. In Fig. 3(h), RoboGNN also benefits from a multi-thread implementation. The learning efficiency is improved by 1.43 times as the number of threads increases from 1 to 5.

Exp-3: Usability of protection set. We also evaluate how ΔL can be independently used to suggest "corrections" of adjacency matrix A to improve GNN models. 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'). We compare ΔL and A and correct A to A' whenever it's inconsistent with ΔL (*e.g.,* if (u, u') is an edge in ΔL and not in A, we insert (u, u') to A). Table 4 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 explicate 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 structural perturbations. RoboGNN jointly learns the desired graph topology and a GNN model that ensures (p, θ)-robustness. Our experimental study confirms that RoboGNN can significantly improve the accuracy of input models with robustness guarantees and suggest small protection sets. A future topic is to enable RoboGNN for explicit link repairing, and the robustification need of other GNN-based downstream tasks.

7 Appendix

For each node, given a graph $G = (V, E)$, a GNN-based classifier M with logits Z, a set of *targeted* nodes $V_t \subseteq V$ and *critical links* $E^p \subseteq (V \times V)\backslash E_+$, a pair (G, M) is *robust* for a node $v_t \in V_t$ and E^p, if a "worst-case margin" $m^*_{y_t,*}(v_t)$ = $\min_{c \neq y_t} m^*_{y_t,c}(v_t) > 0$, where y_t denotes the true label of node v_t, and c is any other class label in $L \setminus \{y_t\}$.

We denote that the perturbed graph $\tilde{G} = (V, \tilde{E})$. E^p represents the set of edges the attacker can decide whether to include it in the graph or exclude it from the graph, *e.g.,* set A_{ij} of G to 1 or 0 respectively. It constrains the search space of the attacker, that $|\tilde{E}\backslash E| + |E\backslash\tilde{E}| \leq p$. Similarly, we also constrain the search space of the attacker for each node $v \in V$ as the local budget b_v for node v, that is $|\tilde{E}^v\backslash E^v| + |E^v\backslash\tilde{E}^v| <= b_v$. $E^v = \{(v, i) \in E\}$ is the set of edges that share the same source node v. To get the set of admissible perturbed graphs Q_{E^p}, we have a set of edges from G that are kept as E_+ and a set of edges that are included to $\tilde{G}$ from E^p ($|E^p| = p$) as E^p_+.

$$
\begin{aligned}
Q_{E^p} = \{(V, \tilde{E} := E_+ \cup E^p_+)|\ & E^p_+ \in P(E^p), \\
& |\tilde{E}\backslash E| + |E\backslash\tilde{E}| \leq p, \\
& |\tilde{E}^v\backslash E^v| + |E^v\backslash\tilde{E}^v| \leq b_v, \forall v\}
\end{aligned}
$$

Lemma 3: *Given a graph G, one target node v, a set of fixed edges E_+ and critical links E^p, the global budget p, the local budget b_v, a model M with logits Z, under any admissible perturbation $\tilde{G} \in Q_{E^p}$ if the pair (G, M) is* robust *for targeted node v and E^p, then any subset critical links $E^{p'} \subset E^p$, the pair (G, M) is also robust for targeted node v and $E^{p'}$* w.r.t. *p and b_v under any corresponding admissible perturbation $\tilde{G}' \in Q_{E^{p'}}$, where $Q_{E^{p'}} = \{(V, \tilde{E}' := E_+ \cup \{E^p\backslash E^{p'}\}_+ \cup E^{p'}_+)$* □

Proof:

By proof by contradiction, let us assume that there exists a specific label $c_v \in L\backslash\{y_t\}$ under perturbed graph $\tilde{G}'$ such that $m^*_{y_t,*}(v) = m^*_{y_t,c_v}(v) \leq 0$. In this case, classifier M predicts node v with class label c_v instead of y_t. At this time, the perturbed graph $\tilde{G}'$ by definition consists of three parts (1)E_+, (2) $\{E^p\backslash E^{p'}\}_+$, and (3) $E^{p'}_+$. Then,

$$\pi_{\tilde{G}'}(v)^T \left(Z_{\{:,y_t\}} - Z_{\{:,c_v\}}\right) \leq 0 \tag{1}$$

Constructing the new graph $\tilde{G}^p$ based on $\tilde{G}'$, we keep all node pairs that belong to $\{E^p\backslash E^{p'}\}$ whose connection statuses unchanged when we expand critical link set from $E^{p'}$ to E^p. Similarly, we keep all node pairs that belong to $E^{p'}$ whose connection statuses unchanged. Clearly, following this construction, no new node pair connection status is changed when we expand critical link set from $E^{p'}$ to E^p. Hence, the global perturbation constraint p and the local perturbation constraint b_v are not violated. Based on the construction, we obtain $\tilde{G}^p := \tilde{G}'$. By definition, $\tilde{G}^p \in Q_{E^p}$ holds and for any label $c \in L\backslash\{y_t\}$, there is $m^*_{y_t,c}(v) > 0$. Then,

$$\pi_{\tilde{G}'}(v)^T \left(Z_{\{:,y_t\}} - Z_{\{:,c_v\}}\right) > 0 \tag{2}$$

From Eq. 1 and Eq. 2, we have a contradiction. Thus, Lemma 3 follows.

□

Theorem 4: *Given a graph $G = (V, E)$, a set of* targeted *nodes V_t, a set of fixed edges E_+ and critical links E^p, the global budget p, local budgets b_v for all $v \in V$, a model M with logits Z, under any admissible perturbation $\tilde{G} \in Q_{E^p}$ if the pair (G, M) is (p, θ)-robust* w.r.t. *V_t and E^p, then any subset critical links $E^{p'} \subset E^p$, under any admissible perturbation $\tilde{G} \in Q_{E^{p'}}$, the pair (G, M) is robust for at least θ fraction of V_t under perturbations of $E^{p'}$.* □

Proof: From Lemma 3, we know for any $v \in V_t$, if the pair (G, M) is robust for node v *w.r.t.* E^p, then the pair (G, M) is also robust for node v *w.r.t.* $E^{p'}$ if $E^{p'} \subset E^p$. Then, θ

fraction of nodes from V_t remain to be robust *w.r.t.* the pair (G, M) when the critical link set is reduced from E^p to $E^{p'}$. Thus, Theorem 4 follows. □

Lemma 5: *Given a graph G, a set of targeted nodes V_t, a set of fixed edges E_+, local budgets b_v for all $v \in V$, a model M with logits Z, the pair (G, M) is robust for $V_1 \in V_t$ under the perturbations of $E^p \backslash \Delta E_1$ and the pair (G, M) is also robust for $V_2 \in V_t$ under the perturbations of $E^p \backslash \Delta E_2$. Then, the pair (G, M) is robust for $\{V_1 \cup V_2\} \in V_t$ under the perturbations of $E^p \backslash (\Delta E_1 \cup \Delta E_2)$* □

Proof: From Lemma 3, we know for any $v \in V_t$, if the pair (G, M) is robust for node v *w.r.t.* $E^p \backslash \Delta E_1$, the pair (G, M) is also robust for node v *w.r.t.* $E^p \backslash (\Delta E_1 \cup \Delta E_2)$. Hence, the pair (G, M) is robust for $V_1 \in V_t$ under the perturbations of $E^p \backslash (\Delta E_1 \cup \Delta E_2)$. Similarly, from Lemma 3, the pair (G, M) is robust for $V_2 \in V_t$ under the perturbations of $E^p \backslash (\Delta E_1 \cup \Delta E_2)$. Any node $v \in \{V_1 \cup V_2\}$ is certifiably robust *w.r.t.* the pair (G, M) and $E^p \backslash (\Delta E_1 \cup \Delta E_2)$. Thus, Lemma 5 follows. □

References

[1] A. Bojchevski and S. Günnemann. Certifiable robustness to graph perturbations. In *NeurIPS*, 2019.

[2] A. Bojchevski, J. Klicpera, B. Perozzi, A. Kapoor, M. Blais, B. Rózemberczki, M. Lukasik, and S. Günnemann. Scaling graph neural networks with approximate pagerank. In *KDD*, 2020.

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

[4] H. Dai, H. Li, T. Tian, X. Huang, L. Wang, J. Zhu, and L. Song. Adversarial attack on graph structured data. *arXiv preprint arXiv:1806.02371*, 2018.

[5] N. Entezari, S. A. Al-Sayouri, A. Darvishzadeh, and E. E. Papalexakis. All you need is low (rank) defending against adversarial attacks on graphs. In *WSDM*, 2020.

[6] C. L. Giles, K. D. Bollacker, and S. Lawrence. Citeseer: An automatic citation indexing system. In *Proceedings of the Third ACM Conference on Digital Libraries*, 1998.

[7] M. Gori, G. Monfardini, and F. Scarselli. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, 2005.

[8] A. Grover and J. Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864, 2016.

[9] W. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, 2017.

[10] D. Harris and S. Harris. *Digital design and computer architecture*. Morgan Kaufmann, 2010.

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

[12] W. Jin, Y. Ma, X. Liu, X. Tang, S. Wang, and J. Tang. Graph structure learning for robust graph neural networks. *KDD*, 2020.

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

[14] Y. Ma, X. Liu, T. Zhao, Y. Liu, J. Tang, and N. Shah. A unified view on graph neural networks as graph signal denoising. In *CIKM*, 2021.

[15] A. K. McCallum, K. Nigam, J. Rennie, and K. Seymore. Automating the construction of internet portals with machine learning. *Information Retrieval*, 2000.

[16] M. Olsen, A. Viglas, and I. Zvedeniouk. An approximation algorithm for the link building problem. *arXiv preprint arXiv:1204.1369*, 2012.

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

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

[19] C. Tran, W.-Y. Shin, and A. Spitz. Community detection in partially observable social networks. *arXiv preprint arXiv:1801.00132*, 2017.

[20] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[21] H. Wu, C. Wang, Y. Tyshetskiy, A. Docherty, K. Lu, and L. Zhu. Adversarial examples on graph data: Deep insights into attack and defense. *IJCAI*, 2019.

[22] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip. A comprehensive survey on graph neural networks. *NeurIPS*, 2020.

[23] Z. Xu, B. Du, and H. Tong. Graph sanitation with application to node classification. *arXiv preprint arXiv:2105.09384*, 2021.

[24] X. Yuan, P. He, Q. Zhu, R. R. Bhat, and X. Li. Adversarial examples: Attacks and defenses for deep learning. corr abs/1712.07107 (2017). *arXiv preprint arXiv:1712.07107*, 2017.

[25] D. Zügner, A. Akbarnejad, and S. Günnemann. Adversarial attacks on neural networks for graph data. In *KDD*, 2018.

[26] D. Zügner and S. Günnemann. Adversarial attacks on graph neural networks via meta learning. *ICLR*, 2019.