Certifiable Robustness and Robust Training for Graph Convolutional Networks

Daniel Zügner Stephan Günnemann
Technical University of Munich, Germany
{zuegnerd,guennemann}@in.tum.de

ABSTRACT

Recent works show that Graph Neural Networks (GNNs) are highly non-robust with respect to adversarial attacks on both the graph structure and the node attributes, making their outcomes unreliable. We propose the first method for certifiable (non-)robustness of graph convolutional networks with respect to perturbations of the node attributes[1]. We consider the case of binary node attributes (e.g. bag-of-words) and perturbations that are L_0-bounded. If a node has been certified with our method, it is guaranteed to be robust under any possible perturbation given the attack model. Likewise, we can certify non-robustness. Finally, we propose a robust semi-supervised training procedure that treats the labeled and unlabeled nodes jointly. As shown in our experimental evaluation, our method significantly improves the robustness of the GNN with only minimal effect on the predictive accuracy.

ACM Reference Format:
Daniel Zügner and Stephan Günnemann. 2019. Certifiable Robustness and Robust Training for Graph Convolutional Networks. In *The 25th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD ’19), August 4–8, 2019, Anchorage, AK, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3292500.3330905

1 INTRODUCTION

Graph data is the core for many high impact applications ranging from the analysis of social networks, over gene interaction networks, to interlinked document collections. One of the most frequently applied tasks on graph data is *node classification*: given a single large (attributed) graph and the class labels of a few nodes, the goal is to predict the labels of the remaining nodes. Applications include the classification of proteins in interaction graphs [9], prediction of customer types in e-commerce networks [6], or the assignment of scientific papers from a citation network into topics [12]. While there exist many classical approaches to node classification [2, 15], recently *graph neural networks* (GNNs), also called graph convolutional networks, have gained much attention and improved the state of the art in node classification [5, 7, 12, 13].

However, there is one big catch: Recently it has been shown that such approaches are vulnerable to adversarial attacks [4, 21, 22]: Even only slight deliberate perturbations of the nodes’ features or the graph structure can lead to completely wrong predictions. Such negative results significantly hinder the applicability of these models. The results become unreliable and such problems open the door for attackers that can exploit these vulnerabilities.

So far, no effective mechanisms are available, which (i) prevent that small changes to the data lead to completely different predictions in a GNN, or (ii) that can verify whether a given GNN is robust w.r.t. specific perturbation model. This is critical, since especially in domains where graph-based learning is used (e.g. the Web) adversaries are omnipresent, e.g., manipulating online reviews and product websites [11]. One of the core challenges is that in a GNN a node’s prediction is also affected when perturbing *other* nodes in the graph – making the space of possible perturbations large. How to make sure that small changes to the input data do not have a dramatic effect to a GNN?

In this work, we shed light on this problem by proposing the first method for *provable robustness* of GNNs. More precisely, we focus on graph convolutional networks and potential perturbations of the node attributes, where we provide:

1) *Certificates:* Given a trained GNN, we can give robustness certificates that state that a node is robust w.r.t. a certain space of perturbations. If the certificate holds, it is guaranteed that *no* perturbation (in the considered space) exists which will change the node’s prediction. Furthermore, we also provide non-robustness certificates that, when they hold, state whether a node is not robust; realized by providing an adversarial example.
2) *Robust Training:* We proposes a learning principle that improves the robustness of the GNN (i.e. making it less sensitive to perturbations) while still ensuring high accuracy for node classification. Specifically, we exploit the semi-supervised nature of the GNN learning task, thus, taking also the unlabeled nodes into account.

In contrast to existing works on provable robustness for classical neural networks/robust training (e.g. [10, 18, 20]), we tackle various additional challenges: Being the first work for graphs, we have to deal with perturbations of multiple instances simultaneously. For this, we introduce a novel space of perturbations where the perturbation budget is constrained locally and globally. Moreover, since the considered data domains are often discrete/binary attributes, we tackle challenging L_0 constraints on the perturbations. Lastly, we exploit a crucial aspect of semi-supervised learning by taking also the unlabeled nodes into account for robust training.

The key idea we will exploit in our work is to estimate the *worst-case* change in the predictions obtained by the GNN under the space of perturbations. If the worst possible change is small, the GNN is robust. Since, however, this worst-case cannot be computed efficiently, we provide *bounds* on this value, providing conservative

[1]Code available at https://www.kdd.in.tum.de/robust-gcn

estimates. More technically, we derive relaxations of the GNN and the perturbations space, enabling efficient computation.

Besides the two core technical contributions mentioned above, we further perform extensive experiments:

3) *Experiments:* We show on various graph datasets that GNNs trained in the traditional way are not robust, i.e. only few of the nodes can be certified to be robust, respectively many are certifiably non-robust even with small perturbation budgets. In contrast, using our robust training we can dramatically improve robustness increasing it by in some cases by factor of four.

Overall, using our method, significantly improves the reliability of GNNs, thus, being highly beneficial when, e.g., using them in real production systems or scientific applications.

2 RELATED WORK

The sensitivity of machine learning models w.r.t. adversarial perturbations has been studied extensively [8] . Only recently, however, researchers have started to investigate adversarial attacks on graph neural networks [4, 21, 22] and node embeddings [1]. All of these works focus on generating adversarial examples. In contrast, we provide the first work to certify and improve the robustness of GNNs. As shown in [21], both perturbations to the node attributes as well as the graph structure are harmful. In this work, we focus on perturbations of the node attributes and we leave structure perturbations for future work.

For 'classical' neural networks various heuristic approaches have been proposed to improve the the robustness to adversarial examples [17]. However, such heuristics are often broken by new attack methods, leading to an arms race. As an alternative, recent works have considered certifiable robustness [3, 10, 18, 20] providing guarantees that no perturbation w.r.t. a specific perturbation space will change an instance's prediction.

For this work, specifically the class of methods based on convex relaxations are of relevance [18, 20]. They construct a convex relaxation for computing a lower bound on the worst-case margin achievable over all possible perturbations. This bound serves as a certificate of robustness. Solving such convex optimization problems can often been done efficiently, and by exploiting duality it enables to even train a robust model [20]. As already mentioned, our work differs significantly from the existing methods since (i) it considers the novel GNN domain with its relational dependencies, (ii) it handles a discrete/binary data domain, while existing works have only handled continuous data; thus also leading to very different constraints on the perturbations, and (iii) we propose a novel robust training procedure which specifically exploits the semi-supervised learning setting of GNNs, i.e. using the unlabeled nodes as well.

3 PRELIMINARIES

We consider the task of (semi-supervised) node classification in a single large graph having binary node features. Let $G = (\boldsymbol{A}, \boldsymbol{X})$ be an attributed graph, where $\boldsymbol{A} \in \{0,1\}^{N \times N}$ is the adjacency matrix and $\boldsymbol{X} \in \{0,1\}^{N \times D}$ represents the nodes' features. W.l.o.g. we assume the node-ids to be $\mathcal{V} = \{1, \ldots, N\}$. Given a subset $\mathcal{V}_L \subseteq \mathcal{V}$ of labeled nodes, with class labels from $\mathcal{C} = \{1, 2, \ldots, K\}$, the goal of node classification is to learn a function $f : \mathcal{V} \to \mathcal{C}$ which maps each node $v \in \mathcal{V}$ to one class in $\mathcal{C}$. In this work, we focus on node classification employing graph neural networks. In particular, we consider graph convolutional networks where the latent representations $\boldsymbol{H}^{(l)}$ at layer l are of the form

$$\boldsymbol{H}^{(l)} = \sigma^{(l)}\left(\hat{\boldsymbol{A}}^{(l-1)} \boldsymbol{H}^{(l-1)} \boldsymbol{W}^{(l-1)} + \boldsymbol{b}^{(l-1)}\right) \text{ for } l = 2, \ldots, L \quad (1)$$

where $\boldsymbol{H}^{(1)} = \boldsymbol{X}$ and with activation functions given by

$$\sigma^{(L)}(\cdot) = \text{softmax}(\cdot), \quad \sigma^{(l)}(\cdot) = \text{ReLU}(\cdot) \quad \text{for } l = 2, \ldots, L-1.$$

The output $\boldsymbol{H}^{(L)}_{vc}$ denotes the probability of assigning node v to class c. The $\hat{\boldsymbol{A}}^{(l)}$ are the message passing matrices that define how the activations are propagated in the network. In GCN [12], for example, $\hat{\boldsymbol{A}}^{(1)} = \ldots = \hat{\boldsymbol{A}}^{(L-1)} = \tilde{\boldsymbol{D}}^{-\frac{1}{2}} \tilde{\boldsymbol{A}} \tilde{\boldsymbol{D}}^{-\frac{1}{2}}$, where $\tilde{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I}_{N \times N}$ and $\tilde{\boldsymbol{D}}_{ii} = \sum_j \tilde{\boldsymbol{A}}_{ij}$. The $\boldsymbol{W}^{(\cdot)}$ and $\boldsymbol{b}^{(\cdot)}$ are the trainable weights of the graph neural network, usually simply learned by minimizing the cross-entropy loss on the given labeled training nodes $\mathcal{V}_L$.

Notations: We denote with $\mathcal{N}_l(t)$ the l-hop neighborhood of a node t, i.e. all nodes which are reachable with l hops (or less) from node t, including the node t itself. Given a matrix $\boldsymbol{X}$, we denote its positive part with $[\boldsymbol{X}]_+ = \max(\boldsymbol{X}, 0)$ where the max is applied entry-wise. Similarly, the negative part is $[\boldsymbol{X}]_- = -\min(\boldsymbol{X}, 0)$, which are non-negative numbers. All matrix norms $||\boldsymbol{X}||_p$ used in the paper are meant to be entry-wise, i.e. flattening $\boldsymbol{X}$ to a vector and applying the corresponding vector norm. We denote with $h^{(l)}$ the dimensionality of the latent space in layer l, i.e. $\boldsymbol{H}^{(l)} \in \mathbb{R}^{N \times h^{(l)}}$. $\boldsymbol{X}_{i:}$ denotes the i-th row of a matrix $\boldsymbol{X}$ and $\boldsymbol{X}_{:j}$ its j-th column.

4 CERTIFYING ROBUSTNESS FOR GRAPH CONVOLUTIONAL NETWORKS

Our first goal is to derive an efficient principle for robustness certificates. That is, given an already trained GNN and a specific node t under consideration (called *target* node), our goal is to provide a certificate which guarantees that the prediction made for node t *will not change* even if the data gets perturbed (given a specific perturbation budget). That is, if the certificate is provided, the prediction for this node is robust under *any* admissible perturbations. Unlike existing works, we cannot restrict perturbations to the instance itself due to the relational dependencies.

However, we can exploit one key insight: for a GNN with L layers, the output $\boldsymbol{H}^{(L)}_{t:}$ of node t depends only on the nodes in its $L-1$ hop neighborhood $\mathcal{N}_{L-1}(t)$. Therefore, instead of operating with Eq. (1), we can 'slice' the matrices $\boldsymbol{X}$ and $\hat{\boldsymbol{A}}^{(l)}$ at each step to only contain the entries that are required to compute the output for the target node t.[2] This step drastically improves scalability – reducing not only the size of the neural network but also the potential perturbations we have to consider later on. We define the matrix slices for a given target t as follows:[3]

$$\dot{\boldsymbol{A}}^{(l)} = \hat{\boldsymbol{A}}^{(l)}_{\mathcal{N}_{L-l}(t), \mathcal{N}_{L-l+1}(t)} \text{ for } l = 1, \ldots, L-1, \quad \dot{\boldsymbol{X}} = \boldsymbol{X}_{\mathcal{N}_{L-1}(t):} \quad (2)$$

where the set indexing corresponds to slicing the rows and columns of a matrix, i.e. $\hat{\boldsymbol{A}}_{\mathcal{N}_2(t), \mathcal{N}_1(t)}$ contains the rows corresponding to the

[2]Note that the shapes of $\boldsymbol{W}$ and $\boldsymbol{b}$ do not change.

[3]To avoid clutter in the notation, since our method certifies robustness with respect to a specific node t, we omit explicitly mentioning the target node t in the following.

two-hop neighbors of node t and the columns corresponding to its one-hop neighbors. As it becomes clear, for increasing l (i.e. depth in the network), the slices of $\dot{A}^{(l)}$ become smaller, and at the final step we only need the target node's one-hop neighbors.

Overall, we only need to consider the following *sliced* GNN:

$$\hat{H}^{(l)} = \dot{A}^{(l-1)} H^{(l-1)} W^{(l-1)} + b^{(l-1)} \qquad \text{for } l = 2, \dots, L \tag{3}$$

$$H^{(l)}_{nj} = \max\left\{\hat{H}^{(l)}_{nj}, 0\right\} \qquad \text{for } l = 2, \dots, L-1 \tag{4}$$

and $H^{(1)} = \dot{X}$. Here, we replaced the ReLU activation by its analytical form, and we denoted with $\hat{H}^{(l)}$ the input before applying the ReLU, and with $H^{(l)}$ the corresponding output. Note that the matrices are getting smaller in size – with $\hat{H}^{(L)}$ actually reducing to a vector that represents the predicted log probabilities (logits) for node t only. Note that we also omitted the softmax activation function in the final layer L since for the final classification decision it is sufficient to consider the largest value of $\hat{H}^{(L)}$. Overall, we denote the output of this sliced GNN as $f^t_\theta(\dot{X}, \dot{A}) = \hat{H}^{(L)} \in \mathbb{R}^K$. Here θ is the set of all parameters, i.e. $\theta = \{W^{(\cdot)}, b^{(\cdot)}\}$.

4.1 Robustness Certificates for GNNs

Given this set-up, we are now ready to define our actual task: We aim to verify whether no admissible perturbation changes the prediction of the target node t. Formally we aim to solve:

Problem 1. *Given a graph G, a target node t, and an GNN with parameters θ. Let y^* denote the class of node t (e.g. given by the ground truth or predicted). The worst case margin between classes y^* and y achievable under some set $\mathcal{X}_{q,Q}(\dot{X})$ of admissible perturbations to the node attributes is given by*

$$m^t(y^*, y) := \underset{\tilde{X}}{\text{minimize}} \; f^t_\theta(\tilde{X}, \dot{A})_{y^*} - f^t_\theta(\tilde{X}, \dot{A})_y \tag{5}$$
$$\text{subject to } \tilde{X} \in \mathcal{X}_{q,Q}(\dot{X})$$

If $m^t(y^*, y) > 0$ for all $y \neq y^*$, the GNN is certifiably robust w.r.t. node t and $\mathcal{X}_{q,Q}$.

If the minimum in Eq. (5) is positive, it means that there exists *no* adversarial example (within our defined admissible perturbations) that leads to the classifier changing its prediction to the other class y – i.e. the logits of class y^* are always larger than the one of y.

Setting reasonable constraints to adversarial attacks is important to obtain certificates that reflect realistic attacks. Works for *classical* neural networks have constrained the adversarial examples to lie on a small ϵ-ball around the original sample measured by, e.g., the infinity-norm or L2-norm [3, 18, 20], often e.g. $\epsilon < 0.1$ This is clearly not practical in our binary setting as an $\epsilon < 1$ would mean that *no* attribute can be changed. To allow reasonable perturbations in a binary/discrete setting one has to allow much larger changes than the ϵ-balls considered so far.

Therefore, motivated by the existing works on adversarial attacks to graphs [21], we consider a more realistic scenario: We define the set of admissible perturbations by limiting the *number* of changes to the original attributes – i.e. we assume a perturbation budget $Q \in \mathbb{N}$ and measure the L_0 norm in the change to $\dot{X}$. It is important to note that in a graph setting an adversary can attack the target node by also changing the node attributes of its $L-1$ hop neighborhood. Thus, Q acts as a *global* perturbation budget.

However, since changing many attributes for a single node might not be desired, we also allow to limit the number of perturbations *locally* – i.e. *for each node* in the $L-1$ hop neighborhood we can consider a budget of $q \in \mathbb{N}$. Overall, in this work we consider admissible perturbations of the form:

$$\mathcal{X}_{q,Q}(\dot{X}) = \Big\{\tilde{X} \;\Big|\; \tilde{X}_{nj} \in \{0,1\} \wedge \|\tilde{X} - \dot{X}\|_0 \leq Q \tag{6}$$
$$\wedge \|\tilde{X}_{n:} - \dot{X}_{n:}\|_0 \leq q \;\forall n \in \mathcal{N}_{L-1}\Big\}.$$

Challenges: There are two major obstacles preventing us from efficiently finding the minimum in Eq. (5). First, our data domain is discrete, making optimization often intractable. Second, our function (i.e. the GNN) f^t_θ is nonconvex due to the nonlinear activation functions in the neural network. But there is hope: As we will show, we can efficiently find *lower bounds* on the minimum of the original problem by performing specific relaxations of (i) the neural network, and (ii) the data domain. This means that if the lower bound is positive, we are certain that our classifier is robust w.r.t. the set of admissible perturbations. Remarkably, we will even see that our relaxation has an optimal solution which is integral. That is, we obtain an optimal solution (i.e. perturbation) which is binary – thus, we can effectively handle the discrete data domain.

4.2 Convex Relaxations

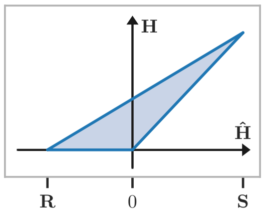
To make the objective function in Eq. (5) convex, we have to find a convex relaxation of the ReLU activation function. While there are many ways to achieve this, we follow the approach of [20] in this work. The core idea is (i) to treat the matrices $H^{(\cdot)}$ and $\hat{H}^{(\cdot)}$ in Eqs. (3,4) no longer as deterministic but as *variables* one can optimize over (besides optimizing over $\tilde{X}$). In this view, Eqs. (3,4) become constraints the variables have to fulfill. Then, (ii) we relax the non-linear ReLU constraint of Eq. (4) by a set of convex ones.

In detail: Consider Eq. (4). Here, $\hat{H}^{(l)}_{nj}$ denotes the input to the ReLU activation function. Let us assume we have given some lower bounds $R^{(l)}_{nj}$ and upper bounds $S^{(l)}_{nj}$ on this input based on the possible perturbations (in Section 4.5 we will discuss how to find these bounds). We denote with $\mathcal{I}^{(l)}$ the set of all tuples (n,j) in layer l for which the lower and upper bounds differ in their sign, i.e. $R^{(l)}_{nj} < 0 < S^{(l)}_{nj}$. We denote with $\mathcal{I}^{(l)}_+$ and $\mathcal{I}^{(l)}_-$ the tuples where both bounds are non-negative and non-positive, respectively.

Consider the case $\mathcal{I}^{(l)}$: We relax Eq. (4) using a convex envelope:

$$H^{(l)}_{nj} \geq \hat{H}^{(l)}_{nj}, \qquad H^{(l)}_{nj} \geq 0,$$
$$H^{(l)}_{nj}\left(S^{(l)}_{nj} - R^{(l)}_{nj}\right) \leq S^{(l)}_{nj}\left(\hat{H}^{(l)}_{nj} - R^{(l)}_{nj}\right) \qquad \text{if } (n,j) \in \mathcal{I}^{(l)}$$

The idea is illustrated in the figure on the right. Note that $H^{(l)}_{nj}$ is no longer the deterministic output of the ReLU given its input but it is a *variable*. For a given input, the variable is constrained to lie on a vertical line above the input and below the upper line of the envelope.



Accordingly, but more simply, for the cases $\mathcal{I}_+^{(l)}$ and $\mathcal{I}_-^{(l)}$ we get:

$$\boldsymbol{H}_{nj}^{(l)} = \hat{\boldsymbol{H}}_{nj}^{(l)} \quad \text{if } (n,j) \in \mathcal{I}_+^{(l)} \qquad \boldsymbol{H}_{nj}^{(l)} = 0 \quad \text{if } (n,j) \in \mathcal{I}_-^{(l)}$$

which are actually not relaxations but exact conditions. Overall, Eq. (4) has now been replaced by a set of linear (i.e. convex) constraints. Together with the linear constraints of Eq. (3) they determine the set of admissible $\boldsymbol{H}^{(\cdot)}$ and $\hat{\boldsymbol{H}}^{(\cdot)}$ we can optimize over. We denote the collection of these matrices that fulfill these constraints by $\mathcal{Z}_{q,Q}(\tilde{\boldsymbol{X}})$. Note that this set depends on $\tilde{\boldsymbol{X}}$ since $\boldsymbol{H}^{(1)} = \tilde{\boldsymbol{X}}$.

Overall, our problem becomes:

$$\tilde{m}^t(y^*, y) := \underset{\tilde{\boldsymbol{X}}, \boldsymbol{H}^{(\cdot)}, \hat{\boldsymbol{H}}^{(\cdot)}}{\text{minimize}} \; \hat{\boldsymbol{H}}_{y^*}^{(L)} - \hat{\boldsymbol{H}}_{y}^{(L)} = \boldsymbol{c}^\top \hat{\boldsymbol{H}}^{(L)} \tag{7}$$
$$\text{subject to } \tilde{\boldsymbol{X}} \in \mathcal{X}_{q,Q}(\dot{\boldsymbol{X}}), \; [\boldsymbol{H}^{(\cdot)}, \hat{\boldsymbol{H}}^{(\cdot)}] \in \mathcal{Z}_{q,Q}(\tilde{\boldsymbol{X}})$$

Here we introduced the constant vector $\boldsymbol{c} = \boldsymbol{e}_{y^*} - \boldsymbol{e}_y$, which is 1 at position y^*, -1 at y, and 0 else. This notation clearly shows that the objective function is a simple linear function.

Corollary 4.1. *The minimum in Eq. (7) is a lower bound on the minimum of the problem in Eq. (5), i.e.* $\tilde{m}^t(y^*, y) \leq m^t(y^*, y)$.

Proof. Let $\tilde{\boldsymbol{X}}$ be the perturbation obtained by Problem 1, and $[\boldsymbol{H}^{(\cdot)}, \hat{\boldsymbol{H}}^{(\cdot)}]$ the resulting *exact* representations based on Eq. (3)+(4). By construction, $[\boldsymbol{H}^{(\cdot)}, \hat{\boldsymbol{H}}^{(\cdot)}] \in \mathcal{Z}_{q,Q}(\dot{\boldsymbol{X}})$. Since Eq. (7) optimizes over the full set $\mathcal{Z}_{q,Q}(\dot{\boldsymbol{X}})$ its minimum can not be larger. □

From Corollary 4.1 it follows that if $\tilde{m}^t(y^*, y) > 0$ for all $y \neq y^*$, the GNN is robust at node t. Directly solving Eq. (7), however, is still intractable due to the discrete data domain.

As one core contribution, we will show that we can find the optimal solution in a tractable way. We proceed in two steps: (i) We first find a suitable continuous, convex relaxation of the discrete domain of possible adversarial examples. (ii) We show that the relaxed problem has an optimal solution which is integral; thus, by our specific construction the solution is binary.

More precisely, we relax the set $\mathcal{X}_{q,Q}(\dot{\boldsymbol{X}})$ to:

$$\hat{\mathcal{X}}_{q,Q}(\dot{\boldsymbol{X}}) = \Big\{ \tilde{\boldsymbol{X}} \;\Big|\; \tilde{\boldsymbol{X}}_{nj} \in [0,1] \wedge \|\tilde{\boldsymbol{X}} - \dot{\boldsymbol{X}}\|_1 \leq Q \tag{8}$$
$$\wedge \|\tilde{\boldsymbol{X}}_{n:} - \dot{\boldsymbol{X}}_{n:}\|_1 \leq q \; \forall n \in \mathcal{N}_{L-1} \Big\}$$

Note that the entries of $\tilde{\boldsymbol{X}}$ are now continuous between 0 and 1, and we have replaced the L_0 norm with the L_1 norm. This leads to:

$$\hat{m}^t(y^*, y) := \underset{\tilde{\boldsymbol{X}}, \boldsymbol{H}^{(\cdot)}, \hat{\boldsymbol{H}}^{(\cdot)}}{\text{minimize}} \; \hat{\boldsymbol{H}}_{y^*}^{(L)} - \hat{\boldsymbol{H}}_{y}^{(L)} = \boldsymbol{c}^\top \hat{\boldsymbol{H}}^{(L)} \tag{9}$$
$$\text{subject to } \tilde{\boldsymbol{X}} \in \hat{\mathcal{X}}_{q,Q}(\dot{\boldsymbol{X}}), \; [\boldsymbol{H}^{(\cdot)}, \hat{\boldsymbol{H}}^{(\cdot)}] \in \mathcal{Z}_{q,Q}(\tilde{\boldsymbol{X}})$$

It is worth mentioning that Eq. (9) is a linear problem since besides the linear objective function also all constraints are linear. We provide the explicit form of this linear program in the appendix. Accordingly, Eq. (9) can be solved optimally in a tractable way. Since $\hat{\mathcal{X}}_{q,Q}(\dot{\boldsymbol{X}}) \supset \mathcal{X}_{q,Q}(\dot{\boldsymbol{X}})$, we trivially have $\hat{m}^t(y^*, y) \leq \tilde{m}^t(y^*, y)$. But even more, we obtain:

Theorem 4.2. *The minimum in Eq. (7) is equal to the minimum in Eq. (9), i.e.* $\tilde{m}^t(y^*, y) = \hat{m}^t(y^*, y)$.

We will proof this theorem later (see Sec. 4.4) since it requires some further results. In summary, using Theorem 4.2, we can indeed handle the discrete data domain/discrete perturbations exactly and tractably by simply solving Eq. (9) instead of Eq. (7).

4.3 Efficient Lower Bounds via the Dual

In order to provide a robustness *guarantee* w.r.t. the perturbations on $\dot{\boldsymbol{X}}$, we have to find the *minimum* of the linear program in Eq. (9) to ensure that we have covered the worst case. While it is possible to solve linear programs 'efficiently' using highly optimized linear program solvers, the potentially large number of variables in a GNN makes this approach rather slow. As an alternative, we can consider the dual of the linear program [20]. There, *any* dual-feasible solution is a lower bound on the minimum of the primal problem. That is, if we find *any* dual-feasible solution for which the objective function of the dual is positive, we know that the minimum of the primal problem has to be positive as well, guaranteeing robustness of the GNN w.r.t. any perturbation in the set.

Theorem 4.3. *The dual of Eq. (9) is equivalent to:*

$$\underset{\Omega, \boldsymbol{\eta}, \rho}{\text{maximize}} \quad g_{q,Q}^t\left(\dot{\boldsymbol{X}}, \boldsymbol{c}, \Omega, \boldsymbol{\eta}, \rho\right) \tag{10}$$
$$\text{subject to}$$
$$\Omega^{(l)} \in [0,1]^{|\mathcal{N}_{L-l}| \times h^{(l)}} \; \textit{for } l = L-1, \ldots, 2,$$
$$\boldsymbol{\eta} \in \mathbb{R}_{\geq 0}^{|\mathcal{N}_{L-1}|}, \; \rho \in \mathbb{R}_{\geq 0}$$

where

$$g_{q,Q}^t(\ldots) = \sum_{l=2}^{L-1} \sum_{(n,j) \in \mathcal{I}^{(l)}} \frac{S_{nj}^{(l)} R_{nj}^{(l)}}{S_{nj}^{(l)} - R_{nj}^{(l)}} \left[\hat{\Phi}_{nj}^{(l+1)}\right]_+ - \sum_{l=1}^{L-1} \boldsymbol{1}^\top \Phi^{(l+1)} \boldsymbol{b}^{(l)}$$
$$- \operatorname{Tr}\left[\dot{\boldsymbol{X}}^\top \hat{\Phi}^{(1)}\right] - \|\Psi\|_1 - q \cdot \sum_n \boldsymbol{\eta}_n - Q \cdot \rho$$

and

$$\Phi^{(L)} = -\boldsymbol{c} \in \mathbb{R}^k$$
$$\hat{\Phi}^{(l)} = \dot{\boldsymbol{A}}^{(l)\top} \Phi^{(l+1)} \boldsymbol{W}^{(l)\top} \in \mathbb{R}^{|\mathcal{N}_{L-l}| \times h^{(l)}} \; \textit{for } l = L-1, \ldots, 1$$
$$\Phi_{nj}^{(l)} = \begin{cases} 0 & \text{if } (n,j) \in \mathcal{I}_-^{(l)} \\ \hat{\Phi}_{nj} & \text{if } (n,j) \in \mathcal{I}_+^{(l)} \\ \frac{S_{nj}^{(l)}}{S_{nj}^{(l)} - R_{nj}^{(l)}} \left[\hat{\Phi}_{nj}^{(l)}\right]_+ - \Omega_{nj}^{(l)} \left[\hat{\Phi}_{nj}^{(l)}\right]_- & \text{if } (n,j) \in \mathcal{I}^{(l)} \end{cases}$$
$$\textit{for } l = L-1, \ldots, 2$$
$$\Psi_{nd} = \max\{\Delta_{nd} - (\boldsymbol{\eta}_n + \rho), 0\}$$
$$\Delta_{nd} = \left[\hat{\Phi}_{nd}^{(1)}\right]_+ \cdot (1 - \dot{\boldsymbol{X}}_{nd}) + \left[\hat{\Phi}_{nd}^{(1)}\right]_- \cdot \dot{\boldsymbol{X}}_{nd}$$

The proof is given in the appendix. Note that parts of the dual problem in Theorem 4.3 have a similar form to the problem in [20]. For instance, we can interpret this dual problem as a *backward* pass on a GNN, where the $\hat{\Phi}^{(l)}$ and $\Phi^{(l)}$ are the hidden representations of the respective nodes in the graph. Crucially different, however, is the propagation in the dual problem with the message passing matrices $\dot{\boldsymbol{A}}$ coming from the GNN formulation where neighboring nodes influence each other. Furthermore, our novel perturbation constraints from Eq. (8) lead to the dual variables $\boldsymbol{\eta}$ and ρ, which have their origin in the local (q) and global (Q) constraints, respectively. Note that, in principle, our framework allows for different budgets q per node. The term $\mathbf{\Psi}$ has its origin in the constraint $\tilde{X}_{nj} \in [0,1]$.

While on the first look, the above dual problem seems rather complicated, its specific form makes it amenable for *easy optimization.* The variables $\Omega, \boldsymbol{\eta}, \rho$ have only simple, element-wise constraints (e.g. clipping between [0, 1]). All other terms are just deterministic assignments. Thus, straightforward optimization using (projected) gradient ascent in combination with any modern automatic differentiation framework (e.g. TensorFlow, PyTorch) is possible.

Furthermore, while in the above dual we need to optimize over $\boldsymbol{\eta}$ and ρ, it turns out that we can simplify it even further: for any feasible Ω, we get an optimal closed-form solution for $\boldsymbol{\eta}, \rho$.

Theorem 4.4. *Given the dual problem from Theorem 4.3 and any dual-feasible value for Ω. For each node $n \in \mathcal{N}_{L-1}$, let S_n be the set of dimensions d corresponding to the q largest values from the vector $\Delta_{n:}$ (ties broken arbitrarily). Further, denote with $o_n = \min_{d \in S_n} \Delta_{nd}$ the smallest of these values. The optimal ρ that maximizes the dual is the Q-th largest value from $[\Delta_{nd}]_{n \in \mathcal{N}_{L-1}, d \in S_n}$. For later use we denote with S_Q the set of tuples (n,d) corresponding to these Q-largest values. Moreover, the optimal $\boldsymbol{\eta}_n$ is $\boldsymbol{\eta}_n = \max\{0, o_n - \rho\}$.*

The proof is given in the appendix. Using Theo. 4.4, we obtain an even more compact dual where we only have to optimize over Ω. Importantly, the calculations done in Theo. 4.4 are also available in many modern automatic differentiation frameworks (i.e. we can back-propagate through them). Thus, we still get very efficient (and easy to implement) optimization.

Default value: As mentioned before, it is not required to solve the dual problem optimally. Any dual-feasible solution leads to a lower bound on the original problem. Specifically, we can also just evaluate the function $g^t_{q,Q}$ *once* given a single instantiation for Ω. This makes the computation of robustness certificates extremely fast. For example, adopting the result of [20], instead of optimizing over Ω we can set it to

$$\Omega^{(l)}_{nj} = S^{(l)}_{nj} \cdot (S^{(l)}_{nj} - R^{(l)}_{nj})^{-1}, \tag{11}$$

which is dual-feasible, and still obtain strong robustness certificates. In our experimental section, we compare the results obtained using this default value to results for optimizing over Ω. Note that using Theo. 4.4 we always ensure to use the optimal $\boldsymbol{\eta}, \rho$ w.r.t. Ω.

4.4 Primal Solutions and Certificates

Based on the above results, we can now prove the following:

Corollary 4.5. *Eq.* (9) *is an integral linear program with respect to the variables $\tilde{X}$.*

The proof is given in the appendix. Using this result, it is now straightforward to prove Theo. 4.2 from the beginning.

Proof. Since Eq. (9) has an optimal (thus, feasible) solution where $\tilde{X}$ is integral, we have $\tilde{X} \in \hat{\mathcal{X}}_{q,Q}(\dot{X})$ and, thus, $\tilde{X}$ has to be binary to be integral. Since in this case the L_1 constraints are equivalent to the L_0 constraints, it follows that $\tilde{X} \in \mathcal{X}_{q,Q}(\dot{X})$. Thus, this optimal solution of Eq. 9 is feasible for Eq. 7 as well. Together with $\hat{m}^t(y^*, y) \leq \tilde{m}^t(y^*, y)$ it follows that $\hat{m}^t(y^*, y) = \tilde{m}^t(y^*, y)$. □

In the proof of Corollary 4.5, we have seen that in the optimal solution, the set $\{(n,d) \in S_Q \mid \Delta_{nd} > 0\} =: P$ indicates those elements which are perturbed. That is, we constructed the worst-case perturbation. Clearly, this mechanism can also be used even if Ω (and, thus, Δ) is not optimal: simply perturbing the elements in P. In this case, of course, the primal solution might not be optimal and we cannot use it for a robustness certificate. However, since the resulting perturbation is primal feasible (regarding the set $\mathcal{X}_{q,Q}(\dot{X})$), we can use it for our non-robustness certificate: After constructing the perturbation $\tilde{X}$ based on P, we pass it through the *exact* GNN, i.e. we evaluate Eq. (5). If the value is negative, we found a harmful perturbation, certifying non-robustness.

In summary: By considering the dual program, we obtain robustness certificates if the obtained (dual) values are positive for every $y \neq y^*$. In contrast, by constructing the primal feasible perturbation using P, we obtain non-robustness certificates if the obtained (exact, primal) values are negative for one $y \neq y^*$. For some nodes, neither of these certificates can be given. We analyze this aspect in more detail in our experiments.

4.5 Activation Bounds

One crucial component of our method, the computation of the bounds $R^{(l)}$ and $S^{(l)}$ on the activations in the relaxed GNN, remains to be defined. Again, existing bounds for classical neural networks are not applicable since they neither consider L_0 constraints nor do they take neighboring instances into account. Obtaining good upper and lower bounds is crucial to obtain robustness certificates, as tighter bounds lead to lower relaxation error of the GNN activations.

While in Sec. 4.3, we relax the discreteness condition of the node attributes $\dot{X}$ in the linear program, it turns out that for the bounds the binary nature of the data can be exploited. More precisely, for every node $m \in \mathcal{N}_{L-2}(t)$, we compute the upper bound $S^{(2)}_{mj}$ in the second layer for latent dimension j as

$$S^{(2)}_{mj} = \text{sum\_top\_Q}\left([\dot{A}^{(1)}_{mn}\hat{S}^{(2)}_{nji}]_{n \in \mathcal{N}_1(m), i \in \{1,\dots,q\}}\right) + \dot{H}^{(2)}_{mj} \tag{12}$$

$$\hat{S}^{(2)}_{nji} = \text{i-th\_largest}\left((1 - \dot{X}_{n:}) \odot \left[W^{(1)}_{:j}\right]_+ + \dot{X}_{n:} \odot \left[W^{(1)}_{:j}\right]_-\right)$$

Here, i-th_largest($\cdot$) denotes the selection of the i-th largest element from the corresponding vector, and sum_top_Q($\cdot$) the sum of the Q largest elements from the corresponding list. The first term of the sum in Eq. (12) is an upper bound on the *change/increase* in the first hidden layer's activations of node m and hidden dimension j for any admissible perturbation on the attributes $\dot{X}$. The second term are the hidden activations obtained for the (un-perturbed) input $\dot{X}$, i.e. $\dot{H}^{(2)}_{mj} = \dot{A}^{(1)}\dot{X}W^{(1)} + b^{(1)}$. In sum we have an upper bound on the hidden activations in the first hidden layer for the perturbed input $\tilde{X}$. Note that, reflecting the interdependence of nodes in the graph, the bounds of a node m depend on the attributes of its neighbors n.

Likewise for the lower bound we use:

$$R^{(2)}_{mj} = \text{- sum\_top\_Q}\left([\dot{A}^{(1)}_{mn}\hat{R}^{(2)}_{nji}]_{n \in \mathcal{N}_1(m), i \in \{1,\dots,q\}}\right) + \dot{H}^{(2)}_{mj} \tag{13}$$

$$\hat{R}^{(2)}_{nji} = \text{i-th\_largest}\left(\dot{X}_{n:} \odot \left[W^{(1)}_{:j}\right]_+ + (1 - \dot{X}_{n:}) \odot \left[W^{(1)}_{:j}\right]_-\right)$$

We need to compute the bounds for each node in the $L-2$ hop neighborhood of the target, i.e. for a GNN with a single hidden layer ($L = 3$) we have $R^{(2)}, S^{(2)} \in \mathbb{R}^{\mathcal{N}_1(t) \times h^{(2)}}$.

Corollary 4.6. *Eqs. (12) and (13) are valid, and the tightest possible, lower/upper bounds w.r.t. the set of admissible perturbations.*

The proof is in the appendix. For the remaining layers, since the input to them is no longer binary, we adapt the bounds proposed in [18]. Generalized to the GNN we therefore obtain:

$$\begin{aligned}
\boldsymbol{R}^{(l)} &= \dot{\boldsymbol{A}}^{(l-1)}\left(\boldsymbol{R}^{(l-1)}\left[\boldsymbol{W}^{(l-1)}\right]_+ - \boldsymbol{S}^{(l-1)}\left[\boldsymbol{W}^{(l-1)}\right]_-\right)\\
\boldsymbol{S}^{(l)} &= \dot{\boldsymbol{A}}^{(l-1)}\left(\boldsymbol{S}^{(l-1)}\left[\boldsymbol{W}^{(l-1)}\right]_+ - \boldsymbol{R}^{(l-1)}\left[\boldsymbol{W}^{(l-1)}\right]_-\right)\\
&\text{for } l = 3, \dots, L-1.
\end{aligned}$$

Intuitively, for the upper bounds we assume that the activations in the previous layer take their respective upper bound wherever we have positive weights, and their lower bounds whenever we have negative weights (and the lower bounds are analogous to this). While there exist more computationally involved algorithms to compute more accurate bounds [20], we leave adaptation of such bounds to the graph domain for future work.

It is important to note that all bounds can be computed highly efficiently and one can even back-propagate through them – important aspects for the robust training (Sec. 5). Specifically, one can compute Eqs. (12) and (13) for all $m \in \mathcal{V}$ (!) and all j *together* in time $O(h^{(2)} \cdot (N \cdot D + E \cdot q))$ where E is the number of edges in the graph. Note that $\hat{\boldsymbol{R}}^{(2)}_{nj:}$ can be computed in time $O(D)$ by unordered partial sorting; overall leading to the complexity $O(N \cdot h^{(2)} \cdot D)$. Likewise the sum of top Q elements can be computed in time $O(\mathcal{N}_1(m) \cdot q)$ for every $1 \leq j \leq h^{(2)}$ and $m \in \mathcal{V}$, together leading to $O(E \cdot q \cdot h^{(2)})$.

5 ROBUST TRAINING OF GNNS

While being able to certify robustness of a given GNN by itself is extremely valuable for being able to trust the model's output in real-world applications, it is also highly desirable to train classifiers that are (certifiably) robust to adversarial attacks. In this section we show how to use our findings from before to train robust GNNs.

Recall that the value of the dual g can be interpreted as a lower bound on the margin between the two considered classes. As a shortcut, we denote with $\boldsymbol{p}^t_\theta(y, \Omega^{(\cdot)}) = \left[-g^t_{q,Q}\left(\dot{\boldsymbol{X}}, \boldsymbol{c}^k, \Omega^k\right)\right]_{1\leq k\leq K}$ the K-dimensional vector containing the (negative) dual objective function values for any class k compared to the given class y, i.e. $\boldsymbol{c}^k = \boldsymbol{e}_y - \boldsymbol{e}_k$. That is, node t with class y^*_t is certifiably robust if $\boldsymbol{p}^t_\theta < 0$ for all entries (except the entry at y^*_t which is always 0). Here, θ denotes the parameters of the GNN.

First consider the training objective typically used to train GNNs for node classification:

$$\underset{\theta}{\text{minimize}} \quad \sum_{t\in\mathcal{V}_L} \mathcal{L}\left(f^t_\theta(\dot{\boldsymbol{X}}, \dot{\boldsymbol{A}}), y^*_t\right), \tag{14}$$

where $\mathcal{L}$ is the cross entropy function (operating on the logits) and $\mathcal{V}_L$ the set of labeled nodes in the graph. y^*_t denotes the (known) class label of node t.

To improve robustness, in [20] (for classical neural networks) it has been proposed to instead optimize

$$\underset{\theta, \left\{\Omega^{t,k}\right\}_{t\in\mathcal{V}_L, 1\leq k\leq K}}{\text{minimize}} \quad \sum_{t\in\mathcal{V}_L} \mathcal{L}\left(\boldsymbol{p}^t_\theta(y^*_t, \Omega^{t,\cdot}), y^*_t\right) \tag{15}$$

which is an upper bound on the worst-case loss achievable. Note that we can omit optimizing over Ω by setting it to Eq. (11). We refer to the loss function in Eq. (15) as *robust cross entropy* loss.

One common issue with deep learning models is overconfidence [14], i.e. the models predicting effectively a probability of 1 for one and 0 for the other classes. Applied to Eq. (15), this means that the vector $\boldsymbol{p}^t_\theta$ is pushed to contain very large negative numbers: the predictions will not only be robust but also very certain even under the worst perturbation. To facilitate *true* robustness and not false certainty in our model's predictions, we therefore propose an alternative robust loss that we refer to as *robust hinge loss*:

$$\hat{\mathcal{L}}_M\left(\boldsymbol{p}, y^*\right) = \sum_{k\neq y*} \max\left\{0, \boldsymbol{p}_k + M\right\}. \tag{16}$$

This loss is positive if $-\boldsymbol{p}^t_{\theta k} = g^t_{q,Q}\left(\dot{\boldsymbol{X}}, \boldsymbol{c}^k, \Omega^k\right) < M$; and zero otherwise. Put simply: If the loss is zero, the node t is certifiably robust – in this case even guaranteeing a margin of at least M to the decision boundary. Importantly, realizing even larger margins (for the worst-case) is not 'rewarded'.

We combine the *robust hinge loss* with standard cross entropy to obtain the following robust optimization problem

$$\min_{\theta,\Omega} \sum_{t\in\mathcal{V}_L} \hat{\mathcal{L}}_M\left(\boldsymbol{p}^t_\theta(y^*_t, \Omega^{t,\cdot}), y^*_t\right) + \mathcal{L}\left(f^t_\theta(\dot{\boldsymbol{X}}, \dot{\boldsymbol{A}}), y^*_t\right). \tag{17}$$

Note that the cross entropy term is operating on the *exact*, non-relaxed GNN, which is a strong advantage over the robust cross entropy loss that only uses the *relaxed* GNN. Thus, we are using the exact GNN model for the node predictions, while the relaxed GNN is only used to ensure robustness. Effectively, if all nodes are robust, the term $\hat{\mathcal{L}}_M$ becomes zero, thus, reducing to the standard cross-entropy loss on the exact GNN (with robustness guarantee).

Robustness in the semi-supervised setting: While Eq. (17) improves the robustness regarding the labeled nodes, we do not consider the given unlabeled nodes. How to handle the semi-supervised setting which is prevalent in the graph domain, ensuring also robustness for the unlabeled nodes? Note that for the unlabeled nodes, we do not necessarily want robustness certificates with a very large margin (i.e. strongly negative $\boldsymbol{p}^t_\theta$) since the classifier's prediction may be wrong in the first place; this would mean that we encourage the classifier to make very certain predictions even when the predictions are wrong. Instead, we want to reflect in our model that some unlabeled nodes might be close to the decision boundary and not make overconfident predictions in these cases.

Our robust hinge loss provides a natural way to incorporate these goals. By setting a smaller margin M_2 for the unlabeled nodes, we can train our classifier to be robust, but does not encourage worst-case logit differences larger than the specified M_2. Importantly, this does *not* mean that the classifier will be less certain in general, since the cross entropy term is unchanged and if the classifier is already robust, the robust hinge loss is 0. Overall:

$$\begin{aligned}
\min_{\theta,\Omega} \sum_{t\in\mathcal{V}_L} &\hat{\mathcal{L}}_{M_1}\left(\boldsymbol{p}^t_\theta(y^*_t, \Omega^{t,\cdot}), y^*_t\right) + \mathcal{L}\left(f^t_\theta(\dot{\boldsymbol{X}}, \dot{\boldsymbol{A}}), y^*_t\right) \\
&+ \sum_{t\in\mathcal{V}\setminus\mathcal{V}_L} \hat{\mathcal{L}}_{M_2}\left(\boldsymbol{p}^t_\theta(\tilde{y}_t, \Omega^{t,\cdot}), \tilde{y}_t\right)
\end{aligned} \tag{18}$$

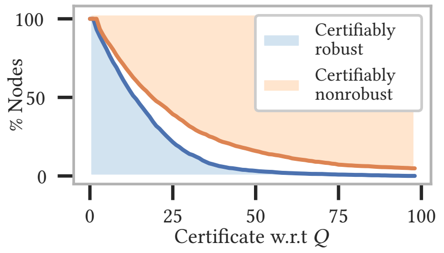


Figure 1: Certificates for a GNN trained with standard training on CORA-ML.

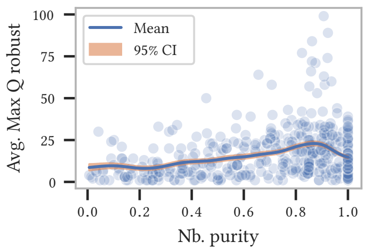


Figure 2: Neighborhood purity correlates with robustness.

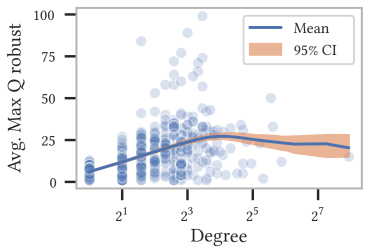


Figure 3: Robustness of nodes vs. their degree.

where $\tilde{y}_t = \arg\max_k f_\theta^t(\dot{X}, \dot{A})_k$ is the predicted label for node t. Note again that the unlabeled nodes are used for robustness purposes only – making it very different to the principle of self-training (see below). Overall, Eq. (18) aims to correctly classify all labeled nodes using the exact GNN, while making sure that *every* node has at least a margin of M_* from the decision boundary even under *worst-case perturbations*.

Eq. (18) can be optimized as is. In practice, however, we proceed as follows: We first train the GNN on the *labeled* nodes using Eq. (17) until convergence. Then we train on *all* nodes using Eq. (18) until convergence.

Discussion: Note that the above idea is not applicable to the robust cross entropy loss from Eq. (15). One might argue that one could use a GNN trained using Eq. (15) to compute predictions for all (or some of the) unlabeled nodes. Then, treating these predictions as the correct (soft-)labels for the nodes and recursively apply the training. This has two undesired effects: If the prediction is very uncertain (i.e. the soft-labels are flat), Eq. (15) tries to find a GNN where the worst-case margin exactly matches these uncertain labels (since this minimizes the cross-entropy). The GNN will be forced to keep the prediction uncertain for such instances even if it could do better. On the other hand, if the prediction is very certain (i.e. very peaky), Eq. (15) tries to make sure that even in the worst-case the prediction has such high certainty – thus being overconfident in the prediction (which might even be wrong in the first place). Indeed, this case mimics the idea of self-training: In self-training, we first train our model on the labeled nodes. Subsequently, we use the predicted classes of (some of) the unlabeled nodes, pretending these are their true labels; and continue training with them as well. Self-training, however, serves an orthogonal purpose and, in principle, can be used with any of the above models.

Summary: When training the GNN, the lower and upper activation bounds are treated as a function of θ, i.e. they are updated accordingly. While this can be done efficiently as discussed in Sec. 4.5, it is still the least efficient part of our model and future work might consider incremental computations. Overall, since the dual program in Theorem 4.3 and the upper/lower activations bounds are differentiable, we can train a robust GNN with gradient descent and standard deep learning libraries. Note again that by setting $\mathbf{\Omega}$ to its default value, we actually only have to optimize over θ – like in standard training. Furthermore, computing $\boldsymbol{p}_\theta^t$ for the default parameters has roughly the same cost as evaluating a usual (sliced) GNN K many times, i.e. it is very efficient.

6 EXPERIMENTAL EVALUATION

Our experimental contributions are twofold. (i) We evaluate the robustness of traditionally trained GNNs using, and thus analyzing, our certification method. (ii) We show that our robust training procedure can dramatically improve GNNs' robustness while sacrificing only minimal accuracy on the unlabeled nodes.

We evaluate our method on the widely used and publicly available datasets CORA-ML (N=2,995, E=8,416, D=2,879, K=7) [16], CITESEER (N=3,312, E=4,715, D=3,703, K=6) [19], and PUBMED (N=19,717, E=44,324, D=500, K=3) [19]. For every dataset, we allow *local* (i.e. per-node) changes to the node attributes amounting to 1% of the attribute dimension, i.e. $q = 0.01D$. Q is analyzed in detail in the experiments reflecting different perturbation spaces.

We refer to the traditional training of GNNs as *Cross Entropy* (short *CE*), to the robust variant of cross entropy as *Robust Cross Entropy (RCE)*, and to our hinge loss variants as *Robust Hinge Loss (RH)* and *Robust Hinge Loss with Unlabeled (RH-U)*, where the latter enforces a margin loss also on the unlabeled nodes. We set M_1, i.e. the margin on the training nodes to $\log(0.9/0.1)$ and M_2 to $\log(0.6/0.4)$ for the unlabeled nodes (*RH-U* only). This means that we train the GNN to (correctly) classify the labeled nodes with output probability of 90% *in the worst case*, and the unlabeled nodes with 60%, reflecting that we do not want our model to be overconfident on the unlabeled nodes. Please note that we do not need to compare against graph adversarial attack models such as [21] since our method gives provable guarantees on the robustness.

While our method can be used for any GNN of the form in Eq. (1), we study the well-established GCN [12], which has shown to outperform many more complicated models. Following [12], we consider GCNs with one hidden layer (i.e. $L = 3$), and choose a latent dimensionality of 32. We split the datasets into 10% labeled and 90% unlabeled nodes. See the appendix for further details.

6.1 Certificates: Robustness of GNNs

We first start to investigate our (non-)robustness certificates by analyzing GNNs trained using standard cross entropy training. Figure 1 shows the main result: for varying Q we report the percentage of nodes (train+test) which are certifiable robust/non-robust on CORA-ML. We can make two important observations: (i) Our certificates are often very tight. That is, the white area (nodes for which we cannot give any – robustness or non-robustness – certificate) is rather small. Indeed, for any given Q, *at most* 30% of the nodes cannot be certified across all datasets and despite no robust training,

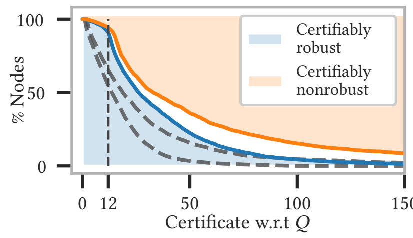


Figure 4: Robust training (Cora-ML). Dashed lines are w/o robust training.

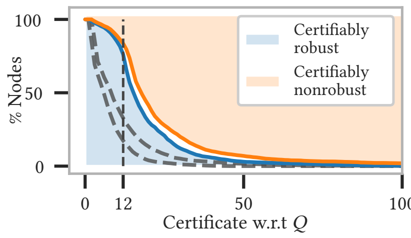


Figure 5: Robust training (Citeseer). Dashed lines are w/o robust training.

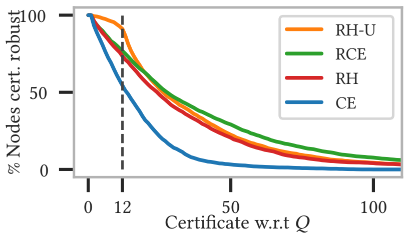


Figure 6: RH-U is most successful for robustness at $Q = 12$ (Cora-ML).

highlighting the tightness of our bounds and relaxations and the effectiveness of our certification method. (ii) GNNs trained traditionally are only certifiably robust up to very small perturbations. At $Q = 12$, less than 55% of the nodes are certifiably robust on Cora-ML. In case of Citeseer even less than 20% (Table 1; training: CE). Even worse, at this point already *two thirds* (for Citeseer) and a quarter (Cora-ML) of the nodes are certifiably non-robust (i.e. we can find adversarial examples), confirming the issues reported in [21]. PubMed behaves similarly (as we will see later, e.g., in Table 1). In our experiments, the labeled nodes are on average more robust than the unlabeled nodes, which is not surprising given that the classifier was not trained using the labels of the latter.

We also investigate what contributes to certain nodes being more robust than others. In Figure 2 we see that neighborhood purity (i.e. the share of nodes in a respective node's two-hop neighborhood that is assigned the same class by the classifier) plays an important role. On Cora-ML, almost *all* nodes that are certifiably robust above $Q \geq 50$ have a neighborhood purity of at least 80%. When analyzing the degree (Figure 3), it seems that nodes with a medium degree are most robust. While counterintuitive at first, having many neighbors also means a large surface for adversarial attacks. Nodes with low degree, in contrast, might be affected more strongly since each node in its neighborhood has a larger influence.

Tightness of lower bounds: Next, we aim to analyze how tight our dual lower bounds are, which we needed to obtain efficient certification. For this, we analyze (i) the value of $g_{q,Q}(\cdot)$ we obtain from our dual solution (either when optimizing over Ω are using the default value), compared to (ii) the value of the primal solution we obtain using our construction from Sec. 4.4. The smaller the difference, the better. As seen in Figure 7, when optimizing over Ω, for most of the nodes the gap is 0. Thus, indeed we can often find the *exact* minimum of the primal via the dual. As expected, when using the default value for Ω the difference between dual and primal is larger. Still, for most nodes the difference is small. Indeed, and more importantly, when considering the actual certificates (where we only need to verify whether the dual is positive; its actual value is not important), the difference between optimizing Ω and its default value become negligible: on Cora-ML, the average maximal Q for which we can certify robustness drops by 0.54; Citeseer 0.18; PubMed 2.3. This highlights that

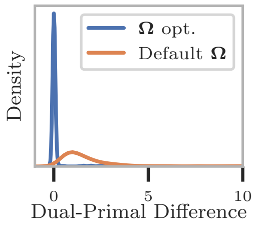


Figure 7: Difference of Primal and Dual Bound.

we can use the default values of Ω to very efficiently certify many or even all nodes in a GNN. In all remaining experiments we, thus, only operate with this default choice.

6.2 Robust Training of GNNs

Next, we analyze our robust training procedure. If not mentioned otherwise, we use our robust hinge-loss including the unlabeled nodes *RH-U* and we robustify the models with $Q = 12$ since for this value more than 50% of nodes across our datasets were not certifiably robust (when using standard training).

Figure 4 and 5 show again the percentage of certified nodes w.r.t. a certain Q – now when using a robustly trained GCN. With dotted lines, we have plotted the curves one obtains for the standard (non-robust) training – e.g. the dotted lines in Fig. 4 are the ones already seen in Fig. 1. As it becomes clear, with robust training, we can dramatically increase the number of nodes which are robust. Almost every node is robust when considering the Q for which the model has been trained for. E.g. for Citeseer, our method is able to quadruple the number of certifiable nodes for $Q = 12$. Put simply: When performing an adversarial attack with $Q \leq 12$ on this model, it cannot do any harm! Moreover the share of nodes that can be certified for any given Q has increased significantly (even though we have not trained the model for $Q > 12$). Most remarkably, nodes for which we certified non-robustness before become now certifiably robust (the blue region above the gray lines).

Accuracy: The increased robustness comes at almost *no loss in classification accuracy* as Table 1 shows. There we report the results for all datasets and all training principles. The last two columns show the accuracy obtained for node classification (for train and test nodes separately). In some cases, our robust classifiers even outperform the non-robust one on the unlabeled nodes. Interestingly, for PubMed we see that the accuracy on the labeled nodes drops to the accuracy on the unlabeled nodes. This indicates that our method can even improve generalization.

Training principles: Comparing the different robust training procedures (also given in more detail in Figure 6), we see that RH-U achieves significantly higher robustness when considering $Q = 12$. This is shown by the third-last column in the table, where the percentage of nodes which are certifiably robust for $Q = 12$ (i.e. the Q the models have been robustified for) is shown. The third column shows the largest Q for which a node is still certifiably robust (averaged over all nodes). As shown, for all training principles the average exceeds the value of 12.

Effect of training with Q: If we strongly increase the Q for which the classifier is trained for, we only observe a small drop in

Dataset	Training	Avg. Max Q robust	% Robust $Q = 12$	Acc. (labeled)	Acc. (unlabeled)
Citeseer	CE	6.77	0.17	1.00	0.67
	RCE	18.62	0.58	0.99	0.69
	RH	15.51	0.54	0.99	0.68
	RH-U	18.48	0.76	0.99	0.68
Cora-ML	CE	16.36	0.54	1.00	0.83
	RCE	38.58	0.77	1.00	0.83
	RH	32.49	0.74	1.00	0.83
	RH-U	35.58	0.91	1.00	0.83
PubMed	CE	5.82	0.15	0.99	0.86
	RCE	50.68	0.62	0.88	0.84
	RH	48.56	0.62	0.90	0.85
	RH-U	47.56	0.63	0.90	0.86

Table 1: Robust training results. Our robust training methods significantly improve the robustness of GNNs while not sacrificing accuracy. Robust training was done for $Q = 12$. Results are averaged over five random data splits.

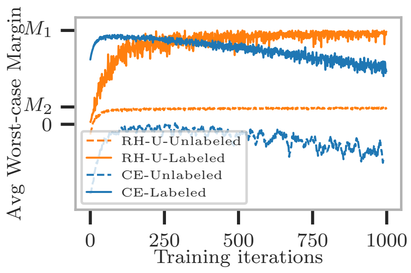


Figure 8: Training dynamics.

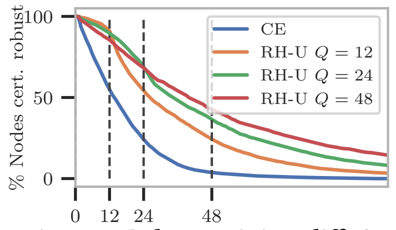


Figure 9: Robust training, diff. Q

the classification accuracy. E.g., training accuracy drops from 99% to 87% when going from $Q = 12$ to 48, while test accuracy stays almost unchanged (68% vs. 66%) on Citeseer. We attribute this to the fact that the GNN still uses the *normal* CE loss in addition to our robust hinge loss during training. Figure 9 shows the results for Cora where we trained three models with different Q. To clarify: We have to distinguish between the Q used for training a model (mentioned in the legend) and the Q we are computing certificates for (the x-axis). We see: (i) Clearly, all trainings lead to significantly more robust models. Though, the larger Q, the harder it gets. (ii) Importantly, each model is the 'winner in robustness' when considering the Q for which the model has been trained for.

Training Dynamics: Lastly, we analyze the behavior when training a GCN using either standard training or robust training with *RH-U*. In Figure 8 we monitor the worst-case margin (averaged over a minibatch of nodes; separately for the labeled and unlabeled nodes) obtained in each training iteration. As seen, with *RH-U* the worst-case margin *increases* to the specified values M_1/M_2 – i.e. making them robust. In contrast, for standard training the worst-case margin *decreases*. Specifically the unlabeled nodes (which account to 90% of all nodes) are not robust.

Overall, all experiments show that our robust training is highly effective: robustness is increased while the accuracy is still high.

7 CONCLUSION

We proposed the first work on certifying robustness of GNNs, considering perturbations of the node attributes under a challenging L_0 perturbation budget and tackling the discrete data domain. By relaxing the GNN and considering the dual, we realized an efficient computation of our certificates – simultaneously our experiments have shown that our certificates are tight since for most nodes a certificate can be given. We have shown that traditional training of GNNs leads to non-robust models that can easily be fooled. In contrast, using our novel (semi-supervised) robust training the resulting GNNs are shown to be much more robust. All this is achieved with only a minor effect on the classification accuracy. As future work we aim to consider perturbations of the graph structure.

REFERENCES

[1] Aleksandar Bojchevski and Stephan Günnemann. 2019. Adversarial Attacks on Node Embeddings via Graph Poisoning. In *ICML*.
[2] Olivier Chapelle, Bernhard Schölkopf, and Alexander Zien. 2006. *Semi-Supervised Learning. Adaptive Computation and Machine Learning series.* The MIT Press.
[3] Francesco Croce, Maksym Andriushchenko, and Matthias Hein. 2018. Provable robustness of relu networks via maximization of linear regions. In *AISTATS*.
[4] Hanjun Dai, Hui Li, Tian Tian, Xin Huang, Lin Wang, Jun Zhu, and Le Song. 2018. Adversarial Attack on Graph Structured Data. In *ICML*.
[5] Michaël Defferrard et al. 2016. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. In *NIPS*.
[6] Dhivya Eswaran, Stephan Günnemann, Christos Faloutsos, Disha Makhija, and Mohit Kumar. 2017. ZooBP: Belief Propagation for Heterogeneous Networks. *PVLDB* 10, 5 (2017), 625–636.
[7] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. 2017. Neural Message Passing for Quantum Chemistry. In *ICML*. 1263–1272.
[8] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and harnessing adversarial examples. In *ICLR*.
[9] William L Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *NIPS*.
[10] Matthias Hein and Maksym Andriushchenko. 2017. Formal Guarantees on the Robustness of a Classifier against Adversarial Manipulation. In *NIPS*. 2263–2273.
[11] Bryan Hooi, Neil Shah, Alex Beutel, Stephan Günnemann, Leman Akoglu, Mohit Kumar, Disha Makhija, and Christos Faloutsos. 2016. BIRDNEST: Bayesian Inference for Ratings-Fraud Detection. In *SDM*. 495–503.
[12] Thomas N Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *ICLR*.
[13] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. 2019. Predict then Propagate: Graph Neural Networks meet Personalized PageRank. In *ICLR*.
[14] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. 2017. Simple and scalable predictive uncertainty estimation using deep ensembles. In *NIPS*.
[15] Ben London and Lise Getoor. 2014. Collective Classification of Network Data. *Data Classification: Algorithms and Applications* 399 (2014).
[16] Andrew Kachites McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. 2000. Automating the construction of internet portals with machine learning. *Information Retrieval* 3, 2 (2000), 127–163.
[17] Nicolas Papernot et al. 2016. Distillation as a Defense to Adversarial Perturbations Against Deep Neural Networks. In *IEEE Symposium on Security and Privacy*.
[18] Aditi Raghunathan, Jacob Steinhardt, and Percy S Liang. 2018. Semidefinite relaxations for certifying robustness to adversarial examples. In *NIPS*.
[19] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. 2008. Collective classification in network data. *AI magazine* 29, 3 (2008), 93.
[20] Eric Wong and Zico Kolter. 2018. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *ICML*. 5283–5292.
[21] Daniel Zügner, Amir Akbarnejad, and Stephan Günnemann. 2018. Adversarial attacks on neural networks for graph data. In *SIGKDD*. 2847–2856.
[22] Daniel Zügner and Stephan Günnemann. 2019. Adversarial Attacks on Graph Neural Networks via Meta Learning. In *ICLR*.

ACKNOWLEDGEMENTS
This research was supported by the German Research Foundation, grant GU 1409/2-1.

8 APPENDIX

Implementation Details: We perform the robust training using stochastic gradient descent with mini-batches and Adam Optimizer. For this we randomly sample in each iteration 20 nodes from the labeled nodes (for RH-U from all nodes) and compute the nodes' twohop neighbors. We then slice the adjacency and attribute matrices appropriately and compute the lower/upper activation bounds for all nodes in the batch. We use dropout of 0.5, L_2 regularization with strength $1e-5$, learning rate of 0.001. We use Tensorflow 1.12 and train on NVIDIA GTX 1080 Ti.

8.1 Proofs

We reformulate the problem in Eq. (9) as the linear program below.

$$\underset{\tilde{\boldsymbol{X}}, \boldsymbol{H}^{(\cdot)}, \hat{\boldsymbol{H}}^{(\cdot)}, \hat{\varepsilon}}{\text{minimize}} \;\; \boldsymbol{c}^\top \hat{\boldsymbol{H}}^{(L)} \text{subject to} \tag{19}$$

$$\begin{aligned}
&\hat{\boldsymbol{H}}^{(l+1)} = \dot{\boldsymbol{A}}^{(l)}\boldsymbol{H}^{(l)}\boldsymbol{W}^{(l)} + \boldsymbol{b}^{(l)}, l = 2,\dots,L && \Rightarrow \boldsymbol{\Phi}^{(l+1)} \in \mathbb{R}^{\mathcal{N}_{L-l}\times h^{(l)}}\\
&\boldsymbol{H}^{(1)}_{nj} \le 1 && \Rightarrow \varepsilon^+ \in \mathbb{R}^{\mathcal{N}_{L-1}\times h^{(1)}}\\
&\boldsymbol{H}^{(1)}_{nj} \ge 0 && \Rightarrow \varepsilon^- \in \mathbb{R}^{\mathcal{N}_{L-1}\times h^{(1)}}\\
&\boldsymbol{H}^{(1)}_{nj} \le \dot{\boldsymbol{X}}_{nj} + \hat{\varepsilon}_{nj} && \Rightarrow \gamma^+ \in \mathbb{R}^{\mathcal{N}_{L-1}\times h^{(1)}}\\
&\boldsymbol{H}^{(1)}_{nj} \ge \dot{\boldsymbol{X}}_{nj} - \hat{\varepsilon}_{nj} && \Rightarrow \gamma^- \in \mathbb{R}^{\mathcal{N}_{L-1}\times h^{(1)}}\\
&\sum_j \hat{\varepsilon}_{nj} \le q \quad \forall n \in \mathcal{N}_{L-1} && \Rightarrow \boldsymbol{\eta} \in \mathbb{R}^{\mathcal{N}_{L-1}}\\
&\sum_{n,j} \hat{\varepsilon}_{nj} \le Q && \Rightarrow \rho \in \mathbb{R}\\
&\boldsymbol{H}^{(l)}_{nj} = 0, l = 2,\dots,L-1, (n,j) \in \mathcal{I}^{(l)}_-\\
&\boldsymbol{H}^{(l)}_{nj} = \hat{\boldsymbol{H}}^{(l)}_{nj}, l = 2,\dots,L-1, (n,j) \in \mathcal{I}^{(l)}_+\\
&\boldsymbol{H}^{(l)}_{nj} \ge 0, l = 2,\dots,L-1, (n,j) \in \mathcal{I}^{(l)} && \Rightarrow \tau^{(l)} \in \mathbb{R}^{\mathcal{N}_{L-l}\times h^{(l)}}\\
&\boldsymbol{H}^{(l)}_{nj} \ge \hat{\boldsymbol{H}}^{(l)}, l = 2,\dots,L-1, (n,j) \in \mathcal{I}^{(l)} && \Rightarrow \mu^{(l)} \in \mathbb{R}^{\mathcal{N}_{L-l}\times h^{(l)}}\\
&\boldsymbol{H}^{(l)}_{nj}\left(\boldsymbol{S}^{(l)}_{nj} - \boldsymbol{R}^{(l)}_{nj}\right) \le \boldsymbol{S}^{(l)}_{nj}\left(\hat{\boldsymbol{H}}^{(l)}_{nj} - \boldsymbol{R}^{(l)}_{nj}\right), && \Rightarrow \lambda^{(l)} \in \mathbb{R}^{\mathcal{N}_{L-l}\times h^{(l)}}\\
&\qquad l = 2,\dots,L-1, (n,j) \in \mathcal{I}^{(l)}
\end{aligned}$$

Note that $\tilde{\boldsymbol{X}} = \boldsymbol{H}^{(1)}$; moreover the $\boldsymbol{H}^{(l)}_{nj} = 0$ and $\boldsymbol{H}^{(l)}_{nj} = \hat{\boldsymbol{H}}^{(l)}$ can be simply eliminated from the optimization. λ, μ, and τ are only defined for $(n,j) \in \mathcal{I}^{(l)}$; we keep the matrix notation for simplicity.

Proof of Theorem 4.3. Applying standard duality construction, the (non-simplified!) dual problem of the above linear program is

$$\begin{aligned}
\max \sum_{l=2}^{L-1} \sum_{(n,j)\in\mathcal{I}^{(l)}} \lambda^{(l)}_{nj}\boldsymbol{S}^{(l)}_{nj}\boldsymbol{R}^{(l)}_{nj} - \sum_{l=1}^{L-1} \boldsymbol{1}^\top\boldsymbol{\Phi}^{(l+1)}\boldsymbol{b}^{(l)}\\
+ \sum_{n,j} \dot{\boldsymbol{X}}_{nj}\left[\gamma^-_{nj} - \gamma^+_{nj}\right] - \varepsilon^+ - q\sum_n \boldsymbol{\eta}_n - Q\rho \quad \text{subject to}
\end{aligned}$$

$$\begin{aligned}
\boldsymbol{\Phi}^{(L)} = -\boldsymbol{c} \qquad \boldsymbol{\Phi}^{(l)}_{nj} &= 0 && \text{for } l = 2,\dots L, (n,j) \in \mathcal{I}^{(l)}_-\\
\boldsymbol{\Phi}^{(l)} &= \dot{\boldsymbol{A}}^{(l)\top}\boldsymbol{\Phi}^{(l+1)}\boldsymbol{W}^{(l)\top}, && \text{for } l = 2,\dots L, (n,j) \in \mathcal{I}^{(l)}_+\\
\boldsymbol{\Phi}^{(l)}_{nj} &= \lambda_{nj}\boldsymbol{S}^{(l)}_{nj} - \mu^{(l)}_{nj} && \text{for } l = 2,\dots L, (n,j) \in \mathcal{I}^{(l)}\\
\dot{\boldsymbol{A}}^{(l)\top}\boldsymbol{\Phi}^{(l+1)}\boldsymbol{W}^{(l)\top} &= \lambda^{(l)} \odot \left[\boldsymbol{S}^{(l)} - \boldsymbol{R}^{(l)}\right]\\
&\quad - \tau^{(l)} - \mu^{(l)} && \text{for } l = 2,\dots L, (n,j) \in \mathcal{I}^{(l)}
\end{aligned}$$

$$\dot{\boldsymbol{A}}^{(1)\top}\boldsymbol{\Phi}^{(2)}\boldsymbol{W}^{(1)\top} = \varepsilon^+ - \varepsilon^- + \gamma^+ - \gamma^- \tag{20}$$

$$\rho + \boldsymbol{\eta}_n \ge \gamma^+_{nj} + \gamma^-_{nj} \quad \forall n, j \tag{21}$$

$$\lambda, \tau, \mu, \varepsilon^+, \varepsilon^-, \gamma^+, \gamma^-, \boldsymbol{\eta}, \rho \ge 0$$

As done in [20] we can exploit complementarity of the ReLU constraints corresponding to $\boldsymbol{H}^{(l)} \ge 0$ and $\boldsymbol{H}^{(l)} \ge \hat{\boldsymbol{H}}^{(l)}$ to eliminate τ, μ, and λ from the problem. For this we write

$$\begin{aligned}
\left[\boldsymbol{S}^{(l)} - \boldsymbol{R}^{(l)}\right] \odot \lambda^{(l)} &= \left[\dot{\boldsymbol{A}}^{(l)\top}\boldsymbol{\Phi}^{(l+1)}\boldsymbol{W}^{(l)\top}\right]_+ = \left[\hat{\boldsymbol{\Phi}}^{(l)}\right]_+\\
\tau^{(l)} + \mu^{(l)} &= \left[\dot{\boldsymbol{A}}^{(l)\top}\boldsymbol{\Phi}^{(l+1)}\boldsymbol{W}^{(l)\top}\right]_- = \left[\hat{\boldsymbol{\Phi}}^{(l)}\right]_-
\end{aligned}$$

where we have defined $\hat{\boldsymbol{\Phi}}^{(l)} := \dot{\boldsymbol{A}}^{(l)\top}\boldsymbol{\Phi}^{(l+1)}\boldsymbol{W}^{(l)\top}$. Given the non-negativity of the dual-variables, it becomes apparent that $\tau^{(l)}$ and $\mu^{(l)}$ "share" the negative part of $\hat{\boldsymbol{\Phi}}^{(l)}$. Thus, we define new variables $\Omega^{(l)}_{nj} \in [0,1]$ such that $\mu^{(l)}_{nj} = \Omega^{(l)}_{nj}\left[\hat{\boldsymbol{\Phi}}^{(l)}_{nj}\right]_-$. Combining this with the constraint $\boldsymbol{\Phi}^{(l)}_{nj} = \lambda_{nj}\boldsymbol{S}^{(l)}_{nj} - \mu^{(l)}_{nj}$ we can rephrase to get

$$\begin{aligned}
\boldsymbol{\Phi}^{(l)}_{nj} &= \frac{\boldsymbol{S}^{(l)}_{nj}}{\boldsymbol{S}^{(l)}_{nj} - \boldsymbol{R}^{(l)}_{nj}}\left[\hat{\boldsymbol{\Phi}}^{(l)}_{nj}\right]_+ - \Omega^{(l)}_{nj}\left[\hat{\boldsymbol{\Phi}}^{(l)}_{nj}\right]_-\\
\lambda^{(l)}_{nd} &= \left[\hat{\boldsymbol{\Phi}}^{(l)}_{nd}\right]_+ \cdot (\boldsymbol{S}^{(l)}_{nj} - \boldsymbol{R}^{(l)}_{nj})^{-1}
\end{aligned}$$

Similarly, by complementarity of the constraints, we know that only one of ε^+_{nj} and ε^-_{nj} and only one of γ^+_{nj} and γ^-_{nj} can be positive. From Eq. (20) we can therefore see that ε^+_{nj} and γ^+_{nj} need to "share" the positive part of the left hand side in Eq. (20) (since all variables are ≥ 0); similarly ε^-_{nj} and γ^-_{nj} share the negative part. We denote this (unknown) share by a new variable $\beta_{nj} \in [0,1]$ and get

$$\begin{aligned}
\varepsilon^+_{nj} &= \beta_{nj}\left[\hat{\boldsymbol{\Phi}}^{(1)}_{nj}\right]_+, & \gamma^+_{nj} &= (1-\beta_{nj})\left[\hat{\boldsymbol{\Phi}}^{(1)}_{nj}\right]_+\\
\varepsilon^-_{nj} &= \beta_{nj}\left[\hat{\boldsymbol{\Phi}}^{(1)}_{nj}\right]_-, & \gamma^-_{nj} &= (1-\beta_{nj})\left[\hat{\boldsymbol{\Phi}}^{(1)}_{nj}\right]_-
\end{aligned}$$

Putting this into Eq. (21) we can now see that

$$\boldsymbol{\eta}_n + \rho \ge (1-\beta_{nj})|\hat{\boldsymbol{\Phi}}^{(1)}_{nj}| \quad \forall 1 \le n \le \mathcal{N}_{L-1}, 1 \le j \le h^{(1)},$$

from which we can get

$$\beta_{nj} \ge 1 - \frac{\rho + \boldsymbol{\eta}_n}{|\hat{\boldsymbol{\Phi}}^{(1)}_{nj}|}, \beta_{nj} \ge 0 \quad \forall n, j \tag{22}$$

to replace the constraint in Eq. (21). Now we can simplify the following term from the dual objective

$$\begin{aligned}
\dot{\boldsymbol{X}}_{nj}\left[\gamma^-_{nj} - \gamma^+_{nj}\right] - \varepsilon^+_{nj} &= -\hat{\boldsymbol{\Phi}}^{(1)}_{nj}\dot{\boldsymbol{X}}_{nj} + \beta_{nj}\hat{\boldsymbol{\Phi}}^{(1)}_{nj}\dot{\boldsymbol{X}}_{nj} - \beta_{nj}\left[\hat{\boldsymbol{\Phi}}^{(1)}_{nj}\right]_+\\
&= -\hat{\boldsymbol{\Phi}}^{(l)}_{nj}\dot{\boldsymbol{X}}_{nj} - \Delta_{nj}\beta_{nj} \text{ where}\\
\Delta_{nj} &:= \left[\hat{\boldsymbol{\Phi}}^{(1)}_{nj}\right]_+ \cdot (1 - \dot{\boldsymbol{X}}_{nd}) + \left[\hat{\boldsymbol{\Phi}}^{(1)}_{nj}\right]_- \cdot \dot{\boldsymbol{X}}_{nd}.
\end{aligned}$$

In the definition of Δ_{nj} we essentially have a case distinction: if $\hat{\Phi}^{(1)}_{nj}$ is positive, we know that increasing the value of the corresponding primal variable $\tilde{X}_{nd}$ will improve the primal objective. If $\dot{X}_{nd}$ is already 1, however, we set the value of Δ_{nj} to zero by multiplying by $1 - \dot{X}_{nd}$ (similarly for the case when $\hat{\Phi}^{(1)}_{nj}$ is negative). This effectively enforces the $0 \le \tilde{X} \le 1$ constraint on the perturbations.

Plugging all terms defined above into our dual objective we get

$$\max \sum_{l=2}^{L-1} \sum_{(n,j)\in\mathcal{I}^{(l)}} \frac{S^{(l)}_{nj} R^{(l)}_{nj}}{S^{(l)}_{nj} - R^{(l)}_{nj}} \left[\hat{\Phi}^{(l+1)}_{nj}\right]_+ - \sum_{l=1}^{L-1} \mathbf{1}^\top \Phi^{(l+1)} \boldsymbol{b}^{(l)}$$
$$- \operatorname{Tr}\left[\dot{X}^\top \hat{\Phi}^{(1)}\right] - \|\Delta \odot \beta\|_1 - q \cdot \sum_n \boldsymbol{\eta}_n - Q \cdot \rho$$

Notice that $\operatorname{Tr}\left[\dot{X}^\top \hat{\Phi}^{(1)}\right] = \sum_n \sum_j \hat{\Phi}_{nj} \dot{X}_{nj}$. Since $\Delta \ge 0$ and $\beta \ge 0$ we could safely write $\|\Delta \odot \beta\|_1$. By observing that $\Delta \ge 0$ for all entries we see that to maximize the objective, we will set β to a value as small as is admissible. This means we can replace Eq. (22) with $\beta = \max\left\{1 - \frac{\rho + \boldsymbol{\eta}_n}{|\hat{\Phi}^{(1)}_{nj}|}, 0\right\}$. Thus, we have now eliminated all dual variables except Ω, ρ, and $\boldsymbol{\eta}_n$. Finally, we define $\Psi_{nj} := \Delta_{nj}\beta_{nj} = \max\left\{\Delta_{nj} - (\boldsymbol{\eta}_n + \rho), 0\right\}$, which finalizes the proof. □

Proof of Theorem 4.4. Given a fixed Ω, the dual function $g^t_{q,Q}$ reduces to $-\|\Psi\|_1 - q \cdot \sum_n \boldsymbol{\eta}_n - Q \cdot \rho + const$ with the term $\Psi_{nd} = \max\{\Delta_{nd} - (\boldsymbol{\eta}_n + \rho), 0\}$ and Δ_{nd} constant. Noticing that $\Psi_{nd} \ge 0$, we see that maximizing the dual is equivalent to minimizing

$$\min_{\rho, \boldsymbol{\eta}_n \ge 0} h(\rho, \boldsymbol{\eta}_n) = \sum_{n,d} \max\{\Delta_{nd} - (\boldsymbol{\eta}_n + \rho), 0\} + q \cdot \sum_n \boldsymbol{\eta}_n + Q \cdot \rho$$

Observe that we can equivalently rephrase this as

$$\min_{\rho, \boldsymbol{\eta}_n, \boldsymbol{U}_{nd} \ge 0} h'(\rho, \boldsymbol{\eta}_n, \boldsymbol{U}) = \sum_{n,d} \boldsymbol{U}_{nd} + q \cdot \sum_n \boldsymbol{\eta}_n + Q \cdot \rho$$
$$s.t.\ \boldsymbol{U}_{nd} \ge \Delta_{nd} - \boldsymbol{\eta}_n - \rho$$

Here we have replaced $\max\{\Delta_{nd} - (\boldsymbol{\eta}_n + \rho), 0\}$ in $h(\rho, \boldsymbol{\eta}_n)$ with a new variable $\boldsymbol{U}_{nd}$ with the constraints $\boldsymbol{U}_{nd} \ge 0$ and $\boldsymbol{U}_{nd} \ge \Delta_{nd} - \boldsymbol{\eta}_n - \rho$ (for each $1 \le n \le \mathcal{N}_{L-1}, 1 \le d \le h^{(1)}$). Since we are minimizing, the optimal values w.r.t. $h'(\rho, \boldsymbol{\eta}_n, \boldsymbol{U})$ and $h(\rho, \boldsymbol{\eta}_n)$ will be the same.

Finding the minimum of $h'(\cdot)$ is a linear program. Thus, we can again form its dual (denoting the dual variables as α_{nd}):

$$\max_{\alpha_{nd} \ge 0} g'(\alpha_{nd}) = \sum_{n,d} \Delta_{nd}\alpha_{nd}$$
$$s.t.\ \alpha_{nd} \le 1, \qquad \sum_{n,d} \alpha_{nd} \le Q, \qquad \sum_d \alpha_{nd} \le q\ \forall n$$

An optimal solution of this dual can be seen and computed easily. Since all Δ_{nd} are nonnegative, we simply set those α_{nd} to 1 corresponding to the largest values of Δ_{nd} – additionally taking the two other constraints into account: The third constraint tells us that the row sums of the α matrix can be at most q, hence we can only set the α_{nd} corresponding to the q largest Δ_{nd} to 1 for each row to maximize the objective. The second constraint means that we can set at most Q entries α_{nd} to 1. So among the set of all q largest Δ_{nd} of the rows we select again the Q largest Δ_{nd} and set their corresponding α_{nd} to 1[4]. Observe that this is precisely the selection process described in the main text for Thm. 4.4. That is, an optimal solution of the dual can be found by setting $\alpha_{nd} = 1 \Leftrightarrow (n,d) \in S_Q$.

We now prove that the variables ρ and $\boldsymbol{\eta}_n$ as described in the main text (along with $\boldsymbol{U}_{nd} = \max\{\Delta_{nd} - \boldsymbol{\eta}_n - \rho, 0\}$) correspond to an *optimal* solution of their respective problem. For this we show that the Karush-Kuhn-Tucker (KKT) conditions hold – using the above constructed solution for α_{nd}. (1) Dual and primal feasibility hold by construction. (2) Next, we check complementary slackness

$$\alpha_{nd}(\Delta_{nd} - \boldsymbol{\eta}_n - \rho - \boldsymbol{U}_{nd}) = 0 \tag{23}$$

If $\alpha_{nd} > 0$ it must hold that the second term is 0. In this case we know that $\Delta_{nd} \ge \boldsymbol{\eta}_n + \rho$ and thus $\boldsymbol{U}_{nd} = \Delta_{nd} - \boldsymbol{\eta}_n - \rho$, which means the term is always 0. When the second term in Eq. (23) is nonzero, α_{nd} must be 0. This is given since when $\Delta_{nd} < \boldsymbol{\eta}_n + \rho + \boldsymbol{U}_{nd}$ it is smaller than the smallest Δ_{nd} for which α_{nd} is set to 1 and therefore $\alpha_{nd} = 0$. (3) Finally we show that $\nabla_\theta L(\theta, \alpha) = \nabla_\theta \sum_{n,d} \boldsymbol{U}_{nd} + q \cdot \sum_n \boldsymbol{\eta}_n + Q \cdot \rho + \sum_{n,d} \alpha_{nd}(\Delta_{nd} - \boldsymbol{\eta}_n - \rho - \boldsymbol{U}_{nd}) = 0$ for $\theta = \{\boldsymbol{U}_{nd}, \boldsymbol{\eta}_n, \rho\}$. Consider first ρ: $\nabla_\rho L(\theta, \alpha) = Q - \sum_{n,d} \alpha_{nd} = 0$ since we set exactly Q many α_{nd} to 1 (and the rest to 0); $\boldsymbol{\eta}_n$ follows analogously. $\nabla_{\boldsymbol{U}_{nd}} L(\theta, \alpha) = \mathbb{I}_{\boldsymbol{U}_{nd} > 0}(1 - \alpha_{nd}) = 0$ holds since when $\alpha_{nd} = 0$, $\Delta_{nd} - \boldsymbol{\eta}_n - \rho \le 0$ which means that $\boldsymbol{U}_{nd}$ must be 0 because of its constraints. Thus, all KKT conditions hold. □

Proof of Corollary 4.5. Assume we are given the optimal values for Ω. We can then compute the optimal values of ρ and $\boldsymbol{\eta}$ as described in Theorem 4.4. Recall from the proof of Thm. 4.3 that the Δ_{nd} denote the improvement in the primal function objective when changing the attribute $\dot{X}_{nd}$. As shown in the proof of Thm. 4.4 with the optimal α_{nd} we exactly choose the values Δ_{nd} that lead to the largest improvement of the objective function – and we trivially observed $\alpha_{nd} = 1$ for those elements. Thus, an optimal solution can be obtained by perturbing the attribute entries $\dot{X}_{nd}$ from the set $P := \{(n,d) | \alpha_{nd} = 1, \Delta_{nd} > 0\}$, i.e. setting them to $1 - \dot{X}_{nd}$. Thus, by construction we found an optimal solution which is integral, making the original linear program integral w.r.t. the attributes $\tilde{X}_{nd}$. By construction of α_{nd} the set $P = \{(n,d) \in S_Q \mid \Delta_{nd} > 0\}$. □

Proof of Corollary 4.6. Using Eq. (3), the (un-perturbed) $\hat{\boldsymbol{H}}^{(2)}_{mj}$ is $\dot{\boldsymbol{A}}^{(1)}_{m:} \dot{X} \boldsymbol{W}^{(1)}_{:j} + \boldsymbol{b}^{(1)}_j = \sum_n \sum_d \dot{\boldsymbol{A}}^{(1)}_{mn} \dot{X}_{nd} \boldsymbol{W}^{(1)}_{dj} + \boldsymbol{b}^{(1)}_j$ which is simply a linear sum in $\dot{X}_{nd}$. Clearly, for maximizing $\hat{\boldsymbol{H}}^{(2)}_{mj}$, one should only perturb $\dot{X}_{nd}$ if ($\dot{\boldsymbol{A}}^{(1)}_{mn} \boldsymbol{W}^{(1)}_{dj}$ is positive and $\dot{X}_{nd} = 0$) or ($\dot{\boldsymbol{A}}^{(1)}_{mn} \boldsymbol{W}^{(1)}_{dj}$ is negative and $\dot{X}_{nd} = 1$). Thus, the maximal *increase* of $\hat{\boldsymbol{H}}^{(2)}_{mj}$ based on $\dot{X}_{nd}$ one can achieve is $\dot{\boldsymbol{A}}^{(1)}_{mn} \cdot \boldsymbol{W}^{(1)}_{dj}$ if the first condition holds, $-\dot{\boldsymbol{A}}^{(1)}_{mn} \cdot \boldsymbol{W}^{(1)}_{dj}$ if the second holds, and 0 else. This can compactly be written as $\dot{\boldsymbol{A}}^{(1)}_{mn} \cdot ([\boldsymbol{W}^{(1)}_{dj}]_+ \cdot (1 - \dot{X}_{nd}) + [\boldsymbol{W}^{(1)}_{dj}]_- \cdot \dot{X}_{nd})$, which matches the terms in Eq. (12). To obtain the maximal *overall* increase in $\hat{\boldsymbol{H}}^{(2)}_{mj}$, and, thus, an upper bound, one simply picks the largest elements that still adhere to the budget constraints (Q,q). Obviously, since this is an admissible perturbation, this upper bound is tight. The proof for the lower bound is accordingly. □

[4]W.l.o.g. we assume $Q \le |\mathcal{N}_{L-1}| \cdot q$ here; otherwise we simply select all of the q largest Δ_{nd} per row (which is equivalent to choosing $Q = |\mathcal{N}_{L-1}| \cdot q$)