Evaluating the Robustness of Graph Convolutional Network through Formal Verification

Lucas Tecot*, Minhao Cheng*, Huan Zhang, Cho-Jui Hsieh
Department of Computer Science, University of California, Los Angeles, CA 90025
lucasmtecot@ucla.edu, mhcheng@ucla.edu, huanzhang@ucla.com, chohsieh@ucla.edu

October 9, 2019

Abstract

Graph Convolutional Network (GCN) is a powerful tool to learn from graph data, however previous works show that GCNs are not robust to adversarial perturbations. Unlike these works which evaluate the robustness of GCNs using heuristic adversarial attacks, we conduct formal robustness analysis on GCNs using efficient neural network verification techniques. Since existing tools on verifying robustness of feed-forward network cannot be efficiently applied to large-scale GCNs, we propose an algorithm that utilizes the special properties of GCN to give tight and more efficient robustness analysis for GCNs. We conduct experiments on a range of GCN models and datasets and show that the certified robustness bounds found by our algorithm are non-trivial, and are aligned with results from adversarial attacks.

1 Introduction

Graphs are at the center of many important problems. They can be found in social networks, protein-interactions, search engines, internet connectivity, and many more [10, 28, 18]. As a result, being able to classify and learn graphical data has been of great interest to the machine learning community. In order to achieve this, there has been increasing interest in Graph Convolutional Networks (GCNs), which are a type of neural network that operate on graphical data [13]. In a nutshell, these networks leverage graphical structure by adding together information from each node with its neighbors at every layer of the neural network. These models are achieving state-of-the-art results, and as a consequence are now receiving a fair amount of attention.

However, neural networks have been shown to be vulnerable to adversarial attacks, in which seemingly inconsequential changes to the input data cause the model to drastically change their output undesirably [8, 23]. These types of attacks have been demonstrated in block-box attacks where the attacker only has access to the model's predictions [17, 4, 5], and succeed in real-world examples [22, 1]. Understandably, this is a massive security problem, especially for neural networks deployed in environments where manipulating features is easy, and changing their outputs can cause large amounts of damage such as aircraft control systems and security cameras. As a result, there has been a large body of work in making models more robust to adversarial of attacks [16, 15, 14, 9, 25, 29].

Unfortunately, GCNs are no exception and are also susceptible to these adversarial attacks [31, 6]. Some recent literature has investigated methods to experimentally make GCNs more robust [24]. However, to our knowledge, no methods have been developed that can make a GCN *provably robust*, in which we can *guarantee* that no adversarial example exists under a given perturbation budget. Specifically, if we allow input features that are perturbed within a certain distance of the original input data (for example, a ℓ_∞ norm bounded perturbation), can we guarantee that no adversarial examples exist in that space?

In our paper, we propose a method to compute a *provably robustness certificate* for GCNs. We borrow techniques from previous works [14, 30, 27] on output range analysis for feed-forward neural networks. However, naively adopting these methods requires to create massive matrices that are extremely computational and

*Equal contribution

memory intensive to calculate. In order to get around this, we propose an efficient method in which we leverage some special properties of 2-layer GCNs in order to compute the bound more efficiently. Then we present an algorithm that extends our method to arbitrary-layer networks with general bilinear weight structure (where GCN is a special case), and discuss relaxations of it for further efficiency improvements.

In experiments, we evaluate the many practical GCNs by comparing our computed upper bound of test error under any attack (*verified error*) with a lower bound of error found via adversarial attack for a variety of datasets and attack scenarios. We found that we can provide good quality of upper bound of test error under attack, and the gap between *verified error* and PGD error is not very large on 3 GCN datasets. We also propose a simple defense method that penalizes the induced norm of weight matrices, which can effectively improve both verified error and PGD error.

2 Background and Related Work

2.1 Graph Convolutional Networks

There are different flavors and formulations of GCNs [13, 3, 11], but the general concept is that at each layer of the network, we do some averaging of data between each node and it's neighbors in the network. In this paper, we'll use a similar notation as Kipf and Welling [13]. Given an adjacency matrix $A \in \mathbb{R}^{N \times N}$, a feature matrix $X \in \mathbb{R}^{N \times D}$, and a non-linear element-wise activation σ, we define that :

$$H^{(l)} = \sigma(\hat{A} H^{(l-1)} W^{(l)}) \qquad (H^{(0)} = X) \tag{1}$$

$$\hat{A} = \tilde{D}^{-1} \tilde{A} \qquad \tilde{A} = A + I_N, \quad \tilde{D}_{ii} = \sum_j \tilde{A}_{ij} \tag{2}$$

H^l is the value of layer l in the GCN, and W^l are trainable weights associated with said layer. Each row in X corresponds to the features of a single node in the network. In the adjacency matrix A, $A_{ij} = 1$ if there is a connection in the graph between node i and j, and 0 otherwise. $\tilde{D}$ serves as a row-regularization term, such that the summation of the nodes doesn't change the scale of the feature vectors. Although typically we deal with undirected graphs where the adjacency matrix is symmetric, for our method we don't require this. Also note that Kipf and Welling [13] use symmetric normalization where $H^{l+1} = \sigma(\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} H^l W^l)$, while we simply row-normalize.

There are a number of works on attacking GCNs via algorithms that find ways to smartly introduce noise that produces outputs desired by the attacker [31, 6]. In response, a few other papers have shown to *empirically* increase robustness via introducing adversarial examples with attacks during training [24]. Our work instead aims to efficiently compute a certified bound on the possible output of the network, which can then be used during training to improve the *guaranteed* area in which the network is robust.

2.2 Certified Adversarial Robustness

The crux of what we are interested in is finding the *minimum adversarial distortion*, which is the minimum change required to the input of the network in order to cause a correct classification to become an incorrect classification. There are a number of methods for finding this exact value [12, 7]. However, unfortunately for common non-linear activations such as ReLU, this is believed to be a NP-Complete problem [12, 21] and is typically too computationally intensive for anything beyond small models. As such, there have been a number of previous works that provide algorithms to efficiently derive lower bounds on the minimum adversarial distortion for multi-layer perceptrons and other common neural network architectures [14, 30, 19, 20, 27, 26]. The state-of-the-art algorithms mostly work in a similar fashion, where we compute a volume in which we can guarantee that our network's output will fall, given that we confine our inputs to a specific p-norm ball around an input point x. Because it is required to "relax" the problem in order to make the computation efficient, it is not guaranteed that the model can output any value in this volume. But, it is guaranteed that any output produced from the specified input space will be in the volume. Using this information, finally it is determined whether or not this output range may allow for an attacker to perturb the input x such that the model will change it's classification of this point.

Our work is based off these type of algorithms. Specifically, we built upon the CROWN method proposed by Zhang et. al. [30]. Their algorithm works in a recursive back-propagation fashion. For a n layer network, they first define the output bounds linearly with respect to the values of layer $n-1$. They then recurse backwards to define it with respect to layer $n-2$, $n-3$, etc., until they have a linear equation with respect to the input features x. Finally, they use the dual-norm to calculate a guaranteed area in which the output neurons will be contained in, given that the input remains within a p-norm ϵ-sized ball around x.

However, in order to apply this algorithm, it is required that linear bounds for each intermediate neuron in the network can be provided. In order to do this, the CROWN method computes the final model output bounds in an incremental manner. First the pre-activation bounds for the first layer are found trivially by simply observing the signs of the weights. Then, the algorithm is applied to find a bound for the second layer, the third layer, etc., until the final layer.

All such previous works only accept linear-form equations, and not the bilinear-form of GCNs. As such, none of them are immediately capable of bounding the outputs of GCNs. However, it is possible to adapt the GCN equation into linear form via the Kronecker product. (Details on how this is done can be found in the appendix.) But generally this modification creates an algorithm that runs in $O(L^2N^3D^3)$ time, where N is the number of nodes in a network, D is the number of features each layer per node, and L is the number of layers. For sizable graphs, the N^3 term in this complexity is not practical. We thus focus on improving the efficiency of these algorithms by exploiting the special bilinear structure of GCNs.

3 Algorithm

In this section we will discuss methods in which to compute bounds on minimal adversarial distortion. First we will present a theorem for more efficiently bounding the output of a two-layer GCN, and a generalized theorem for deeper bilinear neural networks. Then we'll address issues of the computational complexity of the many-layer bound, and briefly touch on possible solutions for future works. Proofs for all theorems, lemmas, and corollaries can be found in the appendix.

3.1 Notations and Definitions

For the following theorems, specifically we are interested in defining two tight global bound values, l and u, such that $l \leq f(X) \leq u, \forall X \in \mathbb{B}_p(X_0, S, \epsilon)$. We define $\mathbb{B}_p(X_0, S, \epsilon) \coloneqq \{X \big| \|X - X_0\|_p \leq \epsilon, X_{[n,:]} = X_{0[n,:]} \forall n \notin S\}$, where S is a set of indices corresponding to specific nodes in the graph. The intuition behind this is we are considering all inputs close to X_0, and seeing if they can produce adversarial results. However, instead of considering the neighborhood of all nodes in the graph, we are only concerning ourselves with perturbations of a few nodes. Therefore, we only allow the perturbation within a p-norm ball of nodes that are in the defined set S, while all the other feature rows of the other nodes must remain equal to the rows of X_0, i.e unchanged. We also use the terms $f^L(X)$ and $f^U(X)$, which represent the lower and upper bound on the network output for a *specific* X. We make this distinction because deriving the specific *explicit bound* first allows us to derive a tighter *global bound*. In this paper, we assume that $f(X)$ is a GCN model as defined by section 2.1. However, note that we will decompose the equation by defining that $y^{(k)} = \hat{A}H^{(k-1)}W^{(k-1)}$ and $H^{(k)} = \sigma(y^{(k)})$. We say that $y^{(k)}$ are the *pre-activations* of the neurons in layer k, while $H^{(k)}$ are the *post-activations*. We will say that each layer $H^{(k)} \in N \times D_k$. In the arbitrary case we define L as the number of layers. Additionally, we define the following:

Definition 3.1 (Linear bounds on an activation function)**.** *For the $[i,j]$-th neuron in the k-th layer of a GCN with pre-activation bounds $l_{i,j}^{(k)}, u_{i,j}^{(k)}$ and the activation function $\sigma(y)$, define two linear functions $h_{U,i,j}^{(k)}, h_{L,i,j}^{(k)} : \mathbb{R} \to \mathbb{R}$, $h_{U,i,j}^{(k)}(y) = \alpha_{U,i,j}^{(k)}(y + \beta_{U,i,j}^{(k)})$, $h_{L,i,j}^{(k)}(y) = \alpha_{L,i,j}^{(k)}(y + \beta_{L,i,j}^{(k)})$, such that $h_{L,i,j}^{(k)}(y) \leq \sigma(y) \leq h_{U,i,j}^{(k)}(y)$, $y \in [l_{i,j}^{(k)}, u_{i,j}^{(k)}]$, $\forall k \in [L-1], i, j \in [N, D_k]$ and $\alpha_{L,i,j}^{(k)}, \alpha_{U,i,j}^{(k)} \in \mathbb{R}^+, \beta_{L,i,j}^{(k)}, \beta_{U,i,j}^{(k)} \in \mathbb{R}$.*

This definition is mainly a slight variation on definition 3.1 in [30]. Intuitively, this can be thought of as bounding the possible outputs of the non-linear activation at a specific layer with two linear equations, and thus converting the bounds into a linear relationship. Many previous works perform relaxations similar to this in order to prevent an exponential growth of constraints per layer [14]. There are many different ways to choose these values for different activations, but for our purposes we use the exact same method as [30].

Also note that the theorems below will illustrate how to directly bound the possible outputs of a network given a $X \in \mathbb{B}_p(X_0, S, \epsilon)$ of possible input, but not how to find the minimal adversarial distortion. However, to accomplish this, all that is required is to do a binary search on the ϵ term. If the algorithm produces an output bound that allows for possible adversarial examples, then the ϵ is reduced, otherwise it is raised.

3.2 Two-Layer GCN Bounds

Lemma 3.2 (Pre-activation bounds of the GCN first layer)**.** *Given a data point $X_0 \in \mathbb{R}^{N \times D_0}$ and $S \subseteq [N]$, for a GCN function $f : \mathbb{R}^{N \times D_0} \to \mathbb{R}^{N \times D_L}$ where $f^{(1)}_{i,m}(X) = \hat{A}XW^{(1)}$, there exists two fixed values $l^{(1)}_{i,m}$ and $u^{(1)}_{i,m}$ such that $\forall X \in \mathbb{B}_p(X_0, S, \epsilon)$ and $\forall i, m \in [N, D_1]$ the inequality $l^{(1)}_{i,m} \leq f^{(1)}_{i,m}(X) \leq u^{(1)}_{i,m}$ holds true, where*

$$u^{(1)}_{i,m} = \hat{A}_{i,:}\tilde{u}^{(0)}_{:,:,m}W^{(1)}_{:,m} \qquad l^{(1)}_{i,m} = \hat{A}_{i,:}\tilde{l}^{(0)}_{:,:,m}W^{(1)}_{:,m} \tag{3}$$

and $\forall j, k \in [N, D_0]$,

$$l^{(0)}_{j,:} = \begin{cases} X_{0[j,:]} - \epsilon \text{ if } n \in S \\ X_{0[j,:]} \text{ otherwise} \end{cases} \qquad u^{(0)}_{j,:} = \begin{cases} X_{0[j,:]} + \epsilon \text{ if } n \in S \\ X_{0[j,:]} \text{ otherwise} \end{cases} \tag{4}$$

$$\tilde{u}^{(0)}_{j,k,m} = \begin{cases} u^{(0)}_{j,k} & \text{if } W^{(1)}_{k,m} \geq 0; \\ l^{(0)}_{j,k} & \text{if } W^{(1)}_{k,m} < 0; \end{cases} \qquad \tilde{l}^{(0)}_{j,k,m} = \begin{cases} l^{(0)}_{j,k} & \text{if } W^{(1)}_{k,m} \geq 0; \\ u^{(0)}_{j,k} & \text{if } W^{(1)}_{k,m} < 0; \end{cases} \tag{5}$$

Using lemma 3.2 and definition 3.1, we can find variables $\alpha^{(1)}_L, \alpha^{(1)}_U, \beta^{(1)}_L, \beta^{(1)}_U$ and apply the following equations to obtain bounds on a two-layer GCN.

Theorem 3.3 (Explicit output bounds of a two-layer GCN)**.** *Given an two-layer GCN function $f : \mathbb{R}^{N \times D_0} \to \mathbb{R}^{N \times D_2}$, there exists two explicit functions $f^L_{i,m} : \mathbb{R}^{N \times D_0} \to \mathbb{R}$ and $f^U_{i,m} : \mathbb{R}^{N \times D_0} \to \mathbb{R}$ such that $\forall i, m \in [N, D_2]$, $\forall X \in \mathbb{B}_p(X_0, S, \epsilon)$, the inequality $f^L_{i,m}(X) \leq f_{i,m}(X) \leq f^U_{i,m}(X)$ holds true, where*

$$f^U_{i,m}(X) = \hat{A}_{i,:}(\lambda^{(1)}_{:,:,m} \odot (\hat{A}XW^{(1)}))W^{(2)}_{:,m} + \hat{A}_{i,:}(\lambda^{(1)}_{:,:,m} \odot \Delta^{(1)}_{:,:,m})W^{(2)}_{:,m} \tag{6}$$

$$f^L_{i,m}(X) = \hat{A}_{i,:}(\omega^{(1)}_{:,:,m} \odot (\hat{A}XW^{(1)}))W^{(2)}_{:,m} + \hat{A}_{i,:}(\omega^{(1)}_{:,:,m} \odot \Theta^{(1)}_{:,:,m})W^{(2)}_{:,m} \tag{7}$$

and $\forall j, k \in [N, D_1]$,

$$\lambda^{(1)}_{j,k,m} = \begin{cases} \alpha^{(1)}_{U,j,k} & \text{if } W^{(2)}_{k,m} \geq 0; \\ \alpha^{(1)}_{L,j,k} & \text{if } W^{(2)}_{k,m} < 0. \end{cases} \qquad \Delta^{(1)}_{j,k,m} = \begin{cases} \beta^{(1)}_{U,j,k} & \text{if } W^{(2)}_{k,m} \geq 0; \\ \beta^{(1)}_{L,j,k} & \text{if } W^{(2)}_{k,m} < 0. \end{cases} \tag{8}$$

$$\omega^{(1)}_{j,k,m} = \begin{cases} \alpha^{(1)}_{L,j,k} & \text{if } W^{(2)}_{k,m} \geq 0; \\ \alpha^{(1)}_{U,j,k} & \text{if } W^{(2)}_{k,m} < 0. \end{cases} \qquad \Theta^{(1)}_{j,k,m} = \begin{cases} \beta^{(1)}_{L,j,k} & \text{if } W^{(2)}_{k,m} \geq 0; \\ \beta^{(1)}_{U,j,k} & \text{if } W^{(2)}_{k,m} < 0. \end{cases} \tag{9}$$

where $\odot$ is the Hadamard product.

Corollary 3.4 (Closed-form global bounds of a two-layer GCN)**.** *Given a data point $X_0 \in \mathbb{R}^{N \times D_0}$, ℓ_p ball parameters $p \geq 1$, $\epsilon > 0$, and $S \subseteq [N]$, for an two-layer GCN function $f : \mathbb{R}^{N \times D_0} \to \mathbb{R}^{N \times D_2}$, there exist two fixed values $l^{(2)}_{i,m}$ and $u^{(2)}_{i,m}$ such that $\forall X \in \mathbb{B}_p(X_0, S, \epsilon)$ and $\forall i, m \in [N, D_2]$, $1/q = 1 - 1/p$, the inequality $l^{(2)}_{i,m} \leq f_{i,m}(X) \leq u^{(2)}_{i,m}$ holds true, where*

$$\begin{aligned} u^{(2)}_{i,m} = {} & \epsilon \|(\hat{A}_{i,:} \otimes W^{(2)\top}_{:,m})\Lambda^{(1)}_{:,:,m}(\hat{A}_S \otimes W^{(1)\top})\|_q \\ & + \hat{A}_{i,:}(\lambda^{(1)}_{:,:,m} \odot (\hat{A}X_0W^{(1)}))W^{(2)}_{:,m} + \hat{A}_{i,:}(\lambda^{(1)}_{:,:,m} \odot \Delta^{(1)}_{:,:,m})W^{(2)}_{:,m} \end{aligned} \tag{10}$$

$$\begin{aligned} l^{(2)}_{i,m} = {} & -\epsilon \|(\hat{A}_{i,:} \otimes W^{(2)\top}_{:,m})\Omega^{(1)}_{:,:,m}(\hat{A}_S \otimes W^{(1)\top})\|_q \\ & + \hat{A}_{i,:}(\omega^{(1)}_{:,:,m} \odot (\hat{A}X_0W^{(1)}))W^{(2)}_{:,m} + \hat{A}_{i,:}(\omega^{(1)}_{:,:,m} \odot \Theta^{(1)}_{:,:,m})W^{(2)}_{:,m} \end{aligned} \tag{11}$$

where vec *is the row-major vector operator, $\otimes$ is the Kronecker product, $\hat{A}_S \in \mathbb{R}^{N \times |S|} : \hat{A}_{S[:,S_n]} \coloneqq \hat{A}_{:,n} \forall n \in S$ (where S_n indicates the index of n in the sorted elements of S), $\Lambda^{(1)}_{:,:,m}$ is a diagonal matrix where* $\text{diag}(\Lambda^{(1)}_{:,:,m}) = \text{vec}(\lambda^{(1)}_{:,:,m})$*, and $\Omega^{(1)}_{:,:,m}$ is a diagonal matrix where* $\text{diag}(\Omega^{(1)}_{:,:,m}) = \text{vec}(\omega^{(1)}_{:,:,m})$.

3.3 Time Complexity

The complexity of finding the bounds is dominated by the first term in l and u for corollary 3.4. This term consists of a ND vector, times a $ND \times ND$ diagonal matrix, times a $SD \times ND$ matrix. Multiplying the first two together is $O(ND)$ and leaves us with a new ND vector. Using the property that $vec(B)(A_S \otimes W^\top) = vec(A_S^T BW)$, where B denotes the new vector reshaped into a $N \times D$ matrix. This leaves us with a $S \times N$ matrix, times a $N \times D$ matrix, times a $D \times D$ matrix. Multiplying these matrices out is $O(SND + SD^2)$. Doing this for each element in each target multiplies this by TD, leaving us with $O(NSTD^2 + STD^3)$ for the whole algorithm. In most cases where S and T are much less than N, this is significantly better than previous methods. However, even in the worst case this becomes $O(N^3D^2 + N^2D^3)$, which is a marginal improvement over the original $O(N^3D^3)$.

3.4 Generalization to Many-Layer Bilinear Weight Networks

Although two-layers are sufficient in a large number of works [13, 18], it is possible to extend this technique to the many-layer case. Additionally, we can generalize further and instead bound general bilinear weight networks, in which we say that :

$$H^{(l)} = \sigma(A^{(l)} H^{(l-1)} W^{(l)}) \qquad (H^{(0)} = X) \tag{12}$$

Where $A^{(l)} \in \mathbb{R}^{N_l \times N_{l-1}}$, $X \in \mathbb{R}^{N_0 \times D_0}$, and $W^{(l)} \in \mathbb{R}^{D_{l-1} \times D_l}$. A can be any matrix, not necessarily an adjacency matrix. Note that GCNs are simply a special case of this equation. However, this method is computationally costly. First we will introduce the theorem corresponding to extending the method from the two-layer case, and then we will analyze possible solutions to improve the complexity.

Theorem 3.5 (Explicit output bounds of a many-layer bilinear networks)**.** *Given an L-layer bilinear neural network function $f : \mathbb{R}^{N_0 \times D_0} \to \mathbb{R}^{N_L \times D_L}$, there exists two explicit functions $f^L_{i,m} : \mathbb{R}^{N_0 \times D_0} \to \mathbb{R}$ and $f^U_{i,m} : \mathbb{R}^{N_0 \times D_0} \to \mathbb{R}$ such that $\forall i, m \in [N_L, D_L]$, $\forall X \in \mathbb{B}_p(X_0, \epsilon)$, the inequality $f^L_{i,m}(X) \le f_{i,m}(X) \le f^U_{i,m}(X)$ holds true, where*

$$f^U_{i,m}(X) = \Gamma^{(L)}_{i,m,i,m}(X) + \sum_{k=1}^{L-1} \Psi^{(k,k)}_{i,m,i,m} \qquad f^L_{i,m}(X) = \Sigma^{(L)}_{i,m,i,m}(X) + \sum_{k=1}^{L-1} \Phi^{(k,k)}_{i,m,i,m} \tag{13}$$

$$\Gamma^{(k)}_{:,:,i,m}(X) = \begin{cases} A^{(k)}(\lambda^{(k-1)}_{:,:,i,m} \odot \Gamma^{(k-1)}_{:,:,i,m}(X))W^{(k)} & \text{if } k = [2, L] \\ A^{(k)} X W^{(k)} & \text{if } k = 1 \end{cases} \tag{14}$$

$$\Sigma^{(k)}_{:,:,i,m}(X) = \begin{cases} A^{(k)}(\omega^{(k-1)}_{:,:,i,m} \odot \Sigma^{(k-1)}_{:,:,i,m}(X))W^{(k)} & \text{if } k = [2, L] \\ A^{(k)} X W^{(k)} & \text{if } k = 1 \end{cases} \tag{15}$$

$$\Psi^{(k,\bar{k})}_{:,:,i,m} = \begin{cases} A^{(L-k+\bar{k})}(\lambda^{(L-k+\bar{k}-1)}_{:,:,i,m} \odot \Psi^{(k,\bar{k}-1)}_{:,:,i,m})W^{(L-k+\bar{k})} & \text{if } \bar{k} = [2, L-1] \\ A^{(L-k+\bar{k})}(\lambda^{(L-k+\bar{k}-1)}_{:,:,i,m} \odot \Delta^{(L-k)}_{:,:,i,m})W^{(L-k+\bar{k})} & \text{if } \bar{k} = 1 \end{cases} \tag{16}$$

$$\Phi^{(k,\bar{k})}_{:,:,i,m} = \begin{cases} A^{(L-k+\bar{k})}(\omega^{(L-k+\bar{k}-1)}_{:,:,i,m} \odot \Phi^{(k,\bar{k}-1)}_{:,:,i,m})W^{(L-k+\bar{k})} & \text{if } \bar{k} = [2, L-1] \\ A^{(L-k+\bar{k})}(\omega^{(L-k+\bar{k}-1)}_{:,:,i,m} \odot \Theta^{(L-k)}_{:,:,i,m})W^{(L-k+\bar{k})} & \text{if } \bar{k} = 1 \end{cases} \tag{17}$$

and $\forall j, n \in [N_k, D_k]$,

$$\xi^{(k)}_{j,n,i,m} = \left[\left[\prod_{\bar{k}=k+1}^{L-1} (A^{(\bar{k})\top} \otimes W^{(\bar{k})}) \Lambda^{(\bar{k})}_{:,:,i,m} \right] (A^{(L)\top}_{i,:} \otimes W^{(L)}_{:,m}) \right]_{j \cdot D_k + n} \tag{18}$$

$$\lambda^{(k)}_{j,n,i,m} = \begin{cases} \alpha^{(k)}_{U,j,n} & \text{if } \xi^{(k)}_{j,n,i,m} \ge 0; \\ \alpha^{(k)}_{L,j,n} & \text{if } \xi^{(k)}_{j,n,i,m} < 0. \end{cases} \qquad \Delta^{(k)}_{j,n,i,m} = \begin{cases} \beta^{(k)}_{U,j,n} & \text{if } \xi^{(k)}_{j,n,i,m} \ge 0; \\ \beta^{(k)}_{L,j,n} & \text{if } \xi^{(k)}_{j,n,i,m} < 0. \end{cases} \tag{19}$$

$$\omega^{(k)}_{j,n,i,m} = \begin{cases} \alpha^{(k)}_{L,j,n} & \text{if } \xi^{(k)}_{j,n,i,m} \geq 0; \\ \alpha^{(k)}_{U,j,n} & \text{if } \xi^{(k)}_{j,n,i,m} < 0. \end{cases} \qquad \Theta^{(k)}_{j,n,i,m} = \begin{cases} \beta^{(k)}_{L,j,n} & \text{if } \xi^{(k)}_{j,n,i,m} \geq 0; \\ \beta^{(k)}_{U,j,n} & \text{if } \xi^{(k)}_{j,n,i,m} < 0. \end{cases} \tag{20}$$

where $\odot$ *is the Hadamard product,* vec *is the row-major vector operator,* $\otimes$ *is the Kronecker product,* $\Lambda^{(k)}_{:,:,i,m}$ *is a diagonal matrix where* $\text{diag}(\Lambda^{(k)}_{:,:,i,m}) = \text{vec}(\lambda^{(k)}_{:,:,i,m})$, $\Omega^{(k)}_{:,:,i,m}$ *is a diagonal matrix where* $\text{diag}(\Omega^{(k)}_{:,:,i,m}) = \text{vec}(\omega^{(k)}_{:,:,i,m})$, *and* $\mathbb{B}_p(X_0, \epsilon) \coloneqq \{X \big| \|X - X_0\|_p \leq \epsilon\}$.

Corollary 3.6 (Closed-form global bounds of a many-layer bilinear networks)**.** *Given a data point* $X_0 \in \mathbb{R}^{N_0 \times D_0}$, ℓ_p *ball parameters* $p \geq 1$ *and* $\epsilon > 0$, *for an* L*-layer bilinear neural network function* $f : \mathbb{R}^{N_0 \times D_0} \to \mathbb{R}^{N_L \times D_L}$, *there exist two fixed values* $l^{(L)}_{i,m}$ *and* $u^{(L)}_{i,m}$ *such that* $\forall X \in \mathbb{B}_p(X_0, \epsilon)$ *and* $\forall i, m \in [N_L, D_L]$, $1/q = 1 - 1/p$, *the inequality* $l^{(L)}_{i,m} \leq f_{i,m}(X) \leq u^{(L)}_{i,m}$ *holds true, where*

$$u^{(L)}_{i,m} = \epsilon \left\| \left[\prod_{\bar{k}=1}^{L-1} (A^{(\bar{k})\top} \otimes W^{(\bar{k})}) \Lambda^{(\bar{k})}_{:,:,i,m} \right] (A^{(L)\top}_{i,:} \otimes W^{(L)}_{:,m}) \right\|_q + \Gamma^{(L)}_{i,m,i,m}(X_0) + \sum_{k=1}^{L-1} \Psi^{(k,k)}_{i,m,i,m} \tag{21}$$

$$l^{(L)}_{i,m} = -\epsilon \left\| \left[\prod_{\bar{k}=1}^{L-1} (A^{(\bar{k})\top} \otimes W^{(\bar{k})}) \Omega^{(\bar{k})}_{:,:,i,m} \right] (A^{(L)\top}_{i,:} \otimes W^{(L)}_{:,m}) \right\|_q + \Sigma^{(L)}_{i,m,i,m}(X_0) + \sum_{k=1}^{L-1} \Phi^{(k,k)}_{i,m,i,m} \tag{22}$$

We present the proofs in Appendix.

This algorithm is intended to be applied iteratively. First it would be applied to a network consisting of the first two layers, and the bounds produced would be used in definition 3.1 to produce additional $\alpha^{(k)}$ and $\beta^{(k)}$ terms, which would then allow this algorithm to be applied to the third layer, etc.

3.5 Time Complexity

For this algorithm, we can follow the exact same procedure used in section 3.3 to compute $\xi^{(k)}$, $u^{(k)}$, and $l^{(k)}$. Only difference is that we have to repeat the process for each layer, adding an additional L term to the complexity. And because we must run this algorithm stopping at layer $1, 2, \ldots, L$ in order to obtain the $\alpha^{(k)}$ and $\beta^{(k)}$ terms at each intermediate layer, we must add another multiplicative term of L. As such, the complexity for this algorithm comes out to $O(L^2N^3D^2 + L^2N^2D^3)$. Although this is an improvement from the original methods, it's only marginal.

However, we could improve this complexity by further relaxing these terms. Note that the crux of the issue in all three of these terms is the presence of the $\Lambda^{(k)}$ and $\Omega^{(k)}$ matrices in-between Kronecker products. If we could decompose this term into its own Kronecker product of two matrices, via the mixed-product property it is possible to reduce the explicit bound into the standard bilinear form AXW, which then becomes much more tractable to compute the global bound for. We tried applying such relaxations to our networks, however in our small-scale tests they extremely worsened the bound. As such, we did not run experiments for them, but solving this fundamental problem will be critical in future works. Additional discussion on this topic can be found in the appendix.

4 Experiments

4.1 Models and datasets

We focus on 3 widely-used real-world datasets, namely the Citeseer, Cora, and Pubmed. These datasets are citation networks used for node classification. The detailed data statistics are shown in Table 1. We use the implementation in [2] to train all our models using the suggested hyper-parameters. We followed the GCN network architectures in [2] (2-layer with ReLU activation).

4.2 Implementation and setup

We implement our proposed algorithm using Pytorch. When computing the robustness bound, we modify the last layer weights using the elision trick [9], in which a new weight matrix is formed to compute the difference between a specific label and the ground truth label of the specific node. This has been shown to be useful in previous works [30, 27]. Most computations in our method are matrix operations that can be automatically parallelized by the BLAS library; however, we set the number of BLAS threads to 1 for a fair comparison to other methods. Experiments were conducted on a cluster with 160 CPUs on Google Cloud.

4.3 Evaluation settings

In adversarial attacks, the attackers try to modify the input data (in our case, the feature matrix X of the graph) such that the changes are unnoticeable. If we limited the perturbation into an ϵ ball over *all* nodes in the graph, the overall feature manipulation would affect a huge number of nodes, and it would make attack too easy (and impractical) and the verified bound error too large to be meaningful. Therefore, we use several practical settings to evaluate the robustness of GCN using our formal verification based technique. We denote the target node set under attack as Ξ. Specifically, we consider two different threat models as follows:

- **Self**: Only features of target node are allowed to perturbed. Other nodes features stay unchanged.
- **k-Hop Neighbours**: Only features of target node's k-hop neighbour are allowed to perturbed. Other nodes features stay unchanged. We use $k = 1, 2, 5$ in our experiments.

For each setting, we conduct a projected gradient descent (PGD) attack with different ϵ and calculate the classification error on our target nodes after the attack. Note that our proposed algorithm is also flexible enough for other settings. Similarly, for the "verified error", we calculate the bounds from corollary 3.4, and then determine the percentage of our target nodes that could be classified incorrectly according to the produced output bounds. Formally, we define the verified error as follows:

Definition 4.1 (Robustness Certificate and robust error)**.** *Given a perturbation radius ϵ where each input feature row X_n for $n \in S$ can be perturbed arbitrarily within $B_p(X, S, \epsilon)$, X is certified robust against any attack if its target nodes $k \in \Xi$'s margin $M_k(X, j) > 0$ for all j. Robust error is the percentage of examples that do* not *have this provable robustness certificate:*

$$\textit{Robust error} := 1 - \frac{|\{k \in \Xi | M_k(X, j) > 0 \textit{ for all } j \in [C-1]\}|}{|\Xi|}. \tag{23}$$

We include three types of error in the experiments defined as follows:

- **Verified bound error**: the error derived by our algorithm. Since our verified bound is got from a relaxation version of the original network. Therefore, margin calculated by our algorithm is a lower bound of the exact margin. And it means the verified bound error could serve as a upper bound for the robust error.
- **PGD error**: the error derived by PGD attack. Since PGD attack is just a kind of attack defined in the robust error, it can serve as a lower bound for the robust error. On the other hand, robust error is an upper bound of guaranteed error under any attacks; i.e., regardlessof any advanced attacks used, an attacker cannot achieve better error than our reported verified error.
- **Interval bound error**: the error derived by interval bound algorithm. Specifically, this corresponds to generating the first-layer bounds from lemma 3.2, applying ReLu to them, then re-applying lemma 3.2 with these bounds as l_0 and u_0, in addition to using $W^{(2)}$ instead of $W^{(1)}$. Intuitively this can be thought of as simply bounding based on the absolute possible lowest and highest output value of each neuron at each layer, instead of finding ways to more smartly bound the non-linear activations with respect to a specific input value. We calculate the interval error using these bound with definition 4.1.

4.4 Experiments on Undefended Networks

Suggested by the original implementation, we have reproduced the performance reported in [2] for all the models. Specifically, we could achieve 87.27% accuracy for Cora, 73.15% accuracy for Citeseer, and 78.60% accuracy for PubMed. For node classification tasks, we randomly select 110 correctly classified test vertexes as our target nodes to be attacked. Specifically, for k-Hop Neighbour setting, to limit the number of nodes allowed to be perturbed, we random sample a maximum of 20 nodes from the k-Hop neighbours. Table 2 shows the our robust bound error, interval bound error and PGD error of different attack settings. From the result, we could see the robust bound error and interval bound error are consistently larger than PGD error under all perturbation strength ϵ among all the datasets, because PGD error serves as verified error lower bound. Also, as showed in Table 3 in appendix, the errors got by robust bound are always smaller than those in interval bound, which indicates our robust bound are strictly tighter than interval bound.

We generally observe both the PGD and verified error increasing as we increase the number of hops, which intuitively makes sense because in most cases we are just increasing the number of nodes perturbed. This trend only doesn't hold for the 5-Hop case, which makes sense as we limit the number of perturbed nodes to 20, meaning that we are likely selecting a couple of nodes that have no chance of affecting the target's output. The main exception to this appears to be the PubMed dataset, where both the PGD and verified error dip at 1-Hop, and then go back up at 2-Hops. This is very interesting behavior, as it suggests that potentially perturbing neighbors in this specific graph is more robust than just perturbing the target node itself.

Table 1: Data statistics. Number of nodes, edges, an average number of 1-hop and 2-hop neighbors per node for each dataset.

Dataset	|V|	|E|	Degree	Degree 2	# of Features
Cora	3,327	12,341	4	15	1,433
Citeseer	2,708	13,264	5	37	3,703
PubMed	19,717	108,365	6	60	500

4.5 Experiments on Networks with Weight Regularization

To further show the effectiveness of our verified bound performance, we conduct experiments on defensive model with weight regularization. [] shows the robustness of model could be improved by adding weight regularization. Therefore, instead of only using cross-entropy loss in the training procedure, we add ℓ_1 penalty on model parameter as a regularization. By setting regularizer hyper-parameter with 0.005, we could achieve 83.21% accuracy for Cora, 72.40% accuracy for Citeseer.

5 Conclusion

We have proposed methods that provide certified perturbation bounds for graph convolutional networks, and achieved speedup in comparison to previous works for two-layer networks. Our experiments show that our method is able to achieve non-trivial upper bounds on guaranteed error under any attacks on large graph datasets under different strengths of perturbations.

Table 2: Comparison of verified error, PGD error, interval error of different attack settings on node classification tasks in Cora, Citeseer, and PubMed under different perturbation strength (ϵ). **Self** represents for the setting that we only allow to perturb the target node's feature. k-Hop stand for the setting that we only allow to perturb the target node's k-hop neighbours' feature.

Datatset	ϵ	Error type	Self	1-Hop	2-Hop	5-Hop
Cora	0.01	PGD	16.67%	19.05%	23.01%	13.49%
		Interval	22.22%	34.13%	57.14%	19.84%
		Verified	19.04%	28.57%	46.03%	15.87%
	0.03	PGD	25.40%	45.23%	69.05%	17.46%
		Interval	55.56%	94.44%	85.71%	27.78%
		Verified	50.00%	85.71%	81.75%	23.01%
	0.05	PGD	40.47%	76.98%	84.13%	23.01%
		Interval	84.92%	98.41%	91.27%	31.75%
		Verified	77.78%	98.41%	89.68 %	26.19%
Citeseer	0.01	PGD	34.67%	40.00 %	51.33%	36.00%
		Interval	59.33%	72.00%	92.67%	59.33%
		Verified	54.67%	64.67%	87.33%	54.67%
	0.02	PGD	55.33%	65.33 %	89.33%	56.00%
		Interval	88.67%	98.00%	97.33%	66.67%
		Verified	81.33%	96.00%	96.67%	66.00%
	0.03	PGD	74.67%	89.33 %	96.67%	64.67%
		Interval	98.00%	99.33%	98.00%	69.33%
		Verified	94.00%	98.00%	97.33%	68.00%
PubMed	0.001	PGD	26.43%	24.29%	30.00%	21.43%
		Interval	34.29%	30.71%	42.86%	21.43%
		Verified	34.29%	30.71%	42.86%	21.43%
	0.005	PGD	63.57%	49.29%	71.43%	22.14%
		Interval	80.71%	72.14%	90.00%	22.14%
		Verified	76.43%	67.14%	87.86%	23.57%
	0.01	PGD	81.42%	70.71%	91.43%	23.57%
		Interval	95.71%	92.86%	95.00%	25.00%
		Verified	94.29%	85.00%	93.57%	25.00%

Table 3: Comparison of verified error and PGD error of different attack settings on ℓ_1 regularization defensive node classification model in Cora under different perturbation strength (ϵ). **Self** represents for the setting that we only allow to perturb the target node's feature. k-Hop stand for the setting that we only allow to perturb the target node's k-hop neighbours' feature.

Datatset	ϵ	Error type	Self	1-Hop	2-Hop	5-Hop
Cora	0.01	PGD	11.8%	11.8%	13.6%	10.9%
		Ours	12.7%	13.6%	17.2%	10.9%
	0.03	PGD	14.5%	18.2%	20.0%	11.8%
		Ours	15.5%	22.7%	27.2%	12.7%
	0.05	PGD	15.5%	20.9%	26.4%	12.7%
		Ours	20.9%	30.9%	48.2%	18.2%
Citeseer	0.01	PGD	13.6%	13.6%	14.5%	13.6%
		Ours	15.5%	16.5%	18.3%	13.8%
	0.03	PGD	16.4%	16.4%	18.2%	13.6%
		Ours	18.2%	19.3%	23.9%	15.6%
	0.05	PGD	17.2%	16.4%	21.8%	14.5%
		Ours	21.8%	25.7%	29.4%	20.2%

References

[1] Anish Athalye, Logan Engstrom, Andrew Ilyas, and Kevin Kwok. Synthesizing robust adversarial examples. *CoRR*, abs/1707.07397, 2017.

[2] Jianfei Chen, Jun Zhu, and Le Song. Stochastic training of graph convolutional networks with variance reduction. *arXiv preprint arXiv:1710.10568*, 2017.

[3] Jie Chen, Tengfei Ma, and Cao Xiao. FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling. *arXiv e-prints*, page arXiv:1801.10247, Jan 2018.

[4] Pin-Yu Chen, Huan Zhang, Yash Sharma, Jinfeng Yi, and Cho-Jui Hsieh. Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pages 15–26. ACM, 2017.

[5] Minhao Cheng, Thong Le, Pin-Yu Chen, Huan Zhang, JinFeng Yi, and Cho-Jui Hsieh. Query-efficient hard-label black-box attack: An optimization-based approach. In *International Conference on Learning Representations*, 2019.

[6] Hanjun Dai, Hui Li, Tian Tian, Xin Huang, Lin Wang, Jun Zhu, and Le Song. Adversarial Attack on Graph Structured Data. *arXiv e-prints*, page arXiv:1806.02371, Jun 2018.

[7] Ruediger Ehlers. Formal Verification of Piece-Wise Linear Feed-Forward Neural Networks. *arXiv e-prints*, page arXiv:1705.01320, May 2017.

[8] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*, 2015.

[9] Sven Gowal, Krishnamurthy Dvijotham, Robert Stanforth, Rudy Bunel, Chongli Qin, Jonathan Uesato, Relja Arandjelovic, Timothy A. Mann, and Pushmeet Kohli. On the effectiveness of interval bound propagation for training verifiably robust models. *CoRR*, abs/1810.12715, 2018.

[10] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. *CoRR*, abs/1607.00653, 2016.

[11] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. *CoRR*, abs/1706.02216, 2017.

[12] Guy Katz, Clark Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. *Lecture Notes in Computer Science*, page 97–117, 2017.

[13] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *CoRR*, abs/1609.02907, 2016.

[14] J. Zico Kolter and Eric Wong. Provable defenses against adversarial examples via the convex outer adversarial polytope. *CoRR*, abs/1711.00851, 2017.

[15] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. Adversarial examples in the physical world. *CoRR*, abs/1607.02533, 2016.

[16] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

[17] Nicolas Papernot, Patrick D. McDaniel, Ian J. Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. Practical black-box attacks against deep learning systems using adversarial examples. *CoRR*, abs/1602.02697, 2016.

[18] SungMin Rhee, Seokjun Seo, and Sun Kim. Hybrid approach of relation network and localized graph convolutional filtering for breast cancer subtype classification. *CoRR*, abs/1711.05859, 2017.

[19] Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin Vechev. Fast and effective robustness certification. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 10802–10813. Curran Associates, Inc., 2018.

[20] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. An abstract domain for certifying neural networks. *Proc. ACM Program. Lang.*, 3(POPL):41:1–41:30, January 2019.

[21] Aman Sinha, Hongseok Namkoong, and John Duchi. Certifying some distributional robustness with principled adversarial training. *arXiv preprint arXiv:1710.10571*, 2017.

[22] Dawn Song, Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Florian Tramer, Atul Prakash, and Tadayoshi Kohno. Physical adversarial examples for object detectors. In *12th {USENIX} Workshop on Offensive Technologies ({WOOT} 18)*, 2018.

[23] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[24] Shen Wang, Zhengzhang Chen, Jingchao Ni, Xiao Yu, Zhichun Li, Haifeng Chen, and Philip S. Yu. Adversarial Defense Framework for Graph Neural Network. *arXiv e-prints*, page arXiv:1905.03679, May 2019.

[25] Shiqi Wang, Yizheng Chen, Ahmed Abdou, and Suman Jana. Mixtrain: Scalable training of formally robust neural networks. *arXiv preprint arXiv:1811.02625*, 2018.

[26] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. Efficient formal safety analysis of neural networks. *CoRR*, abs/1809.08098, 2018.

[27] Tsui-Wei Weng, Huan Zhang, Hongge Chen, Zhao Song, Cho-Jui Hsieh, Duane Boning, Inderjit S Dhillon, and Luca Daniel. Towards fast computation of certified robustness for relu networks. *arXiv preprint arXiv:1804.09699*, 2018.

[28] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. *CoRR*, abs/1806.01973, 2018.

[29] Huan Zhang, Hongge Chen, Chaowei Xiao, Bo Li, Duane Boning, and Cho-Jui Hsieh. Towards stable and efficient training of verifiably robust neural networks. *arXiv preprint arXiv:1906.06316*, 2019.

[30] Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. Efficient neural network robustness certification with general activation functions. *CoRR*, abs/1811.00866, 2018.

[31] Daniel Zügner, Amir Akbarnejad, and Stephan Günnemann. Adversarial Attacks on Neural Networks for Graph Data. *arXiv e-prints*, page arXiv:1805.07984, May 2018.

Appendix

Proof of Lemma 3.2

Using the definition of the pre-activation, we have that:

$$y_{i,m}^{(1)} = \hat{A}_{i,:} X W_{:,m}^{(1)} = \sum_{j=1}^{N} \sum_{k=1}^{D_l} \hat{A}_{i,j} X_{j,k} W_{k,m}^{(1)} \tag{24}$$

$$= \sum_{W_{k,m}^{(1)} \geq 0} \sum_{j=1}^{N} \hat{A}_{i,j} X_{j,k} W_{k,m}^{(1)} + \sum_{W_{k,m}^{(1)} < 0} \sum_{j=1}^{N} \hat{A}_{i,j} X_{j,k} W_{k,m}^{(1)} \tag{25}$$

Note that this pre-activation equation is linear. As such, if we have a lower and upper bound on each input element, we can easily derive bounds for this equation by just observing the sign of the elements multiplied with X. Given that we defined $X \in \mathbb{B}_p(X_0, S, \epsilon)$, we have that:

$$l^{(0)} \leq \mathbb{B}_p(X_0, S, \epsilon) \leq u^{(0)} \tag{26}$$

$$l_{j,:}^{(0)} = \begin{cases} X_{0[j,:]} - \epsilon \text{ if } n \in S \\ X_{0[j,:]} \text{ otherwise} \end{cases} \qquad u_{j,:}^{(0)} = \begin{cases} X_{0[j,:]} + \epsilon \text{ if } n \in S \\ X_{0[j,:]} \text{ otherwise} \end{cases} \tag{27}$$

Thus, if the associated weight $W_{k,m}^{(1)}$ to the $[i,m]$-th neuron is non-negative, we have

$$\sum_{j=1}^{N} \hat{A}_{i,j} l_{j,k}^{(0)} W_{k,m}^{(1)} \leq \sum_{j=1}^{N} \hat{A}_{i,j} X_{j,k} W_{k,m}^{(1)} \leq \sum_{j=1}^{N} \hat{A}_{i,j} u_{j,k}^{(0)} W_{k,m}^{(1)} \tag{28}$$

Otherwise, we have

$$\sum_{j=1}^{N} \hat{A}_{i,j} u_{j,k}^{(0)} W_{k,m}^{(1)} \leq \sum_{j=1}^{N} \hat{A}_{i,j} X_{j,k} W_{k,m}^{(1)} \leq \sum_{j=1}^{N} \hat{A}_{i,j} l_{j,k}^{(0)} W_{k,m}^{(1)} \tag{29}$$

Note that here we are effectively ignoring the effects of $\hat{A}$ on the sign of each element. We are allowed to do this because $\hat{A}$ is a non-negative matrix. As such, it's multiplication with X will not affect the total sign of the values multiplied with each element of X. (Intuitively applying $\hat{A}$ to X can be thought of as averaging together some rows. Because the weights multiply with X on a per-row basis, the selection of $l^{(0)}$ or $u^{(0)}$ along each entire column of X should remain consistent.)

Thus, we can obtain the upper bound as follows:

$$u_{i,m}^{(1)} = \sum_{W_{k,m}^{(1)} \geq 0} \sum_{j=1}^{N} \hat{A}_{i,j} u_{j,k}^{(0)} W_{k,m}^{(1)} + \sum_{W_{k,m}^{(1)} < 0} \sum_{j=1}^{N} \hat{A}_{i,j} l_{j,k}^{(0)} W_{k,m}^{(1)} \tag{30}$$

$$= \hat{A}_{i,:} \tilde{u}_{:,:,m}^{(0)} W_{:,m}^{(1)} \tag{31}$$

where

$$\tilde{u}_{j,k,m}^{(0)} = \begin{cases} u_{j,k}^{(0)} & \text{if } W_{k,m}^{(1)} \geq 0; \\ l_{j,k}^{(0)} & \text{if } W_{k,m}^{(1)} < 0; \end{cases} \tag{32}$$

We can obtain the lower bound with a similar procedure, but instead swapping whether we use the lower or upper bounds based on the weight term sign. Therefore:

$$l_{i,m}^{(1)} = \hat{A}_{i,:} \tilde{l}_{:,:,m}^{(0)} W_{:,m}^{(1)} \qquad \tilde{l}_{j,k,m}^{(0)} = \begin{cases} l_{j,k}^{(0)} & \text{if } W_{k,m}^{(1)} \geq 0; \\ u_{j,k}^{(0)} & \text{if } W_{k,m}^{(1)} < 0; \end{cases} \tag{33}$$

Proof of Theorem 3.3

Using the equation of a two-layer GCN:

$$f_{i,m}(X) = \hat{A}H^{(1)}W^{(2)} \tag{34}$$

$$= \sum_{W^{(2)}_{k,m}\geq 0}\sum_{j=1}^{N}\hat{A}_{i,j}H^{(1)}_{j,k}W^{(2)}_{k,m} + \sum_{W^{(2)}_{k,m}<0}\sum_{j=1}^{N}\hat{A}_{i,j}H^{(1)}_{j,k}W^{(2)}_{k,m} \tag{35}$$

Notice that this is of identical notation as equation 25, so we can use identical reasoning as that in the proof of lemma 3.2. As such, using our first-layer bounds and definition 3.1 to obtain $\alpha^{(1)}_U$ and $\beta^{(1)}_U$, we can say that $f(X)$ is upper bounded by:

$$f^U_{i,m}(X) = \sum_{W^{(2)}_{k,m}\geq 0}\sum_{j=1}^{N}\hat{A}_{i,j}\alpha^{(1)}_{U,j,k}(y^{(1)}_{j,k}+\beta^{(1)}_{U,j,k})W^{(2)}_{k,m} + \sum_{W^{(2)}_{k,m}<0}\sum_{j=1}^{N}\hat{A}_{i,j}\alpha^{(1)}_{L,j,k}(y^{(1)}_{j,k}+\beta^{(1)}_{L,j,k})W^{(2)}_{k,m} \tag{36}$$

$$= \sum_{k=1}^{D_2}\sum_{j=1}^{N}\hat{A}_{i,j}\lambda^{(1)}_{j,k,m}(y^{(1)}_{j,k}+\Delta^{(1)}_{j,k,m})W^{(2)}_{k,m} \tag{37}$$

$$= \hat{A}_{i,:}(\lambda^{(1)}_{:,:,m}\odot(y^{(1)}+\Delta^{(1)}_{:,:,m}))W^{(2)}_{:,m} \tag{38}$$

$$= \hat{A}_{i,:}(\lambda^{(1)}_{:,:,m}\odot(\hat{A}XW^{(1)}))W^{(2)}_{:,m} + \hat{A}_{i,:}(\lambda^{(1)}_{:,:,m}\odot\Delta^{(1)}_{:,:,m})W^{(2)}_{:,m} \tag{39}$$

where $\odot$ is the Hadamard product, and

$$\lambda^{(1)}_{j,k,m} = \begin{cases}\alpha^{(1)}_{U,j,k} & \text{if } W^{(2)}_{k,m}\geq 0;\\ \alpha^{(1)}_{L,j,k} & \text{if } W^{(2)}_{k,m}<0.\end{cases} \qquad \Delta^{(1)}_{j,k,m} = \begin{cases}\beta^{(1)}_{U,j,k} & \text{if } W^{(2)}_{k,m}\geq 0;\\ \beta^{(1)}_{L,j,k} & \text{if } W^{(2)}_{k,m}<0.\end{cases} \tag{40}$$

Similarly, we can obtain the lower bound via an identical procedure but by swapping our selection of the upper and lower bounds with respect to the weights. Therefore:

$$f^L_{i,m}(X) = \hat{A}_{i,:}(\omega^{(1)}_{:,:,m}\odot(\hat{A}XW^{(1)}))W^{(2)}_{:,m} + \hat{A}_{i,:}(\omega^{(1)}_{:,:,m}\odot\Theta^{(1)}_{:,:,m})W^{(2)}_{:,m} \tag{41}$$

$$\omega^{(1)}_{j,k,m} = \begin{cases}\alpha^{(1)}_{L,j,k} & \text{if } W^{(2)}_{k,m}\geq 0;\\ \alpha^{(1)}_{U,j,k} & \text{if } W^{(2)}_{k,m}<0.\end{cases} \qquad \Theta^{(1)}_{j,k,m} = \begin{cases}\beta^{(1)}_{L,j,k} & \text{if } W^{(2)}_{k,m}\geq 0;\\ \beta^{(1)}_{U,j,k} & \text{if } W^{(2)}_{k,m}<0.\end{cases} \tag{42}$$

Proof of Corollary 3.4

Using the explicit bounds from theorem 3.3, we can say that:

$$u^{(2)}_{i,m} = \max_{X\in\mathbb{B}_p(X_0,S,\epsilon)} f^U_{i,m}(X) \tag{43}$$

$$= \max_{X\in\mathbb{B}_p(X_0,S,\epsilon)}\left[\hat{A}_{i,:}\lambda^{(1)}_{:,:,m}\odot(\hat{A}XW^{(1)})W^{(2)}_{:,m} + \hat{A}_{i,:}(\lambda^{(1)}_{:,:,m}\odot\Delta^{(1)}_{:,:,m})W^{(2)}_{:,m}\right] \tag{44}$$

$$= \epsilon\max_{z\in\mathbb{B}_p(0,S,1)}\left[\hat{A}_{i,:}\lambda^{(1)}_{:,:,m}\odot(\hat{A}zW^{(1)})W^{(2)}_{:,m}\right] + \hat{A}_{i,:}\lambda^{(1)}_{:,:,m}\odot(\hat{A}X_0W^{(1)})W^{(2)}_{:,m} + \hat{A}_{i,:}(\lambda^{(1)}_{:,:,m}\odot\Delta^{(1)}_{:,:,m})W^{(2)}_{:,m} \tag{45}$$

$$= \epsilon\max_{z\in\mathbb{B}_p(0,S,1)}\left[\hat{A}_{i,:}\otimes W^{(2)\top}_{:,m}\Lambda^{(1)}_{:,:,m}\hat{A}\otimes W^{(1)\top}\operatorname{vec}(z)\right] + \hat{A}_{i,:}\lambda^{(1)}_{:,:,m}\odot(\hat{A}X_0W^{(1)})W^{(2)}_{:,m} + \hat{A}_{i,:}(\lambda^{(1)}_{:,:,m}\odot\Delta^{(1)}_{:,:,m})W^{(2)}_{:,m} \tag{46}$$

$$= \epsilon\|(\hat{A}_{i,:}\otimes W^{(2)\top}_{:,m})\Lambda^{(1)}_{:,:,m}(\hat{A}_S\otimes W^{(1)\top})\|_q + \hat{A}_{i,:}\lambda^{(1)}_{:,:,m}\odot(\hat{A}X_0W^{(1)})W^{(2)}_{:,m} + \hat{A}_{i,:}(\lambda^{(1)}_{:,:,m}\odot\Delta^{(1)}_{:,:,m})W^{(2)}_{:,m} \tag{47}$$

In equation 46, we utilize the fact that $\text{vec}(AXB) = (A \otimes B^\top)\,\text{vec}(X)$, where vec is the row-major vector operator $\text{vec}(H) \coloneqq [H_{1,1}, \cdots, H_{1,D_l}, H_{2,1}, \cdots, H_{2,D_l}, \cdots, H_{N,1}, \cdots, H_{N,D_l}]^T$. Additionally, the Hadamard product is removed by noting that $\text{vec}(C \odot AXB) = \tilde{C}(A \otimes B^\top)\,\text{vec}(X)$, where $\tilde{C}$ is a diagonal vector such that $\text{diag}(\tilde{C}) = \text{vec}(C)$.

A near-identical derivation for the lower bound can also be done, but where the min is taken instead of the max. This gives us:

$$\begin{aligned} l^{(2)}_{i,m} = &-\epsilon \|(\hat{A}_{i,:} \otimes W^{(2)\top}_{:,m})\Omega^{(1)}_{:,:,m}(\hat{A}_S \otimes W^{(1)\top})\|_q \\ &+ \hat{A}_{i,:}\omega^{(1)}_{:,:,m} \odot (\hat{A}X_0W^{(1)})W^{(2)}_{:,m} + \hat{A}_{i,:}(\omega^{(1)}_{:,:,m} \odot \Theta^{(1)}_{:,:,m})W^{(2)}_{:,m} \end{aligned} \tag{48}$$

Proof of Theorem 3.5

The procedure used during the proof of theorem 3.3 can be repeated iteratively for any arbitrary number of layers. For instance, if we assume instead that f is instead a bilinear neural network with L layers, it follows from equation 39:

$$f^U_{i,m}(X) = A^{(L)}_{i,:}(\lambda^{(L-1)}_{:,:,i,m} \odot (A^{(L-1)}H^{(L-2)}W^{(L-1)}))W^{(L)}_{:,m} + A^{(L)}_{i,:}(\lambda^{(L-1)}_{:,:,i,m} \odot \Delta^{(L-1)}_{:,:,i,m})W^{(L)}_{:,m} \tag{49}$$

$$\begin{aligned} &= A^{(L)}_{i,:}(\lambda^{(L-1)}_{:,:,i,m} \odot (A^{(L-1)}(\lambda^{(L-2)}_{:,:,i,m} \odot (A^{(L-2)}H^{(L-3)}W^{(L-2)} + \Delta^{(L-2)}_{:,:,i,m}))W^{(L-1)}))W^{(L)}_{:,m} \\ &\quad + A^{(L)}_{i,:}(\lambda^{(L-1)}_{:,:,i,m} \odot \Delta^{(L-1)}_{:,:,i,m})W^{(L)}_{:,m} \end{aligned} \tag{50}$$

$$\begin{aligned} &= A^{(L)}_{i,:}(\lambda^{(L-1)}_{:,:,i,m} \odot (A^{(L-1)}(\lambda^{(L-2)}_{:,:,i,m} \odot (A^{(L-2)}H^{(L-3)}W^{(L-2)}))W^{(L-1)}))W^{(L)}_{:,m} \\ &\quad + A^{(L)}_{i,:}(\lambda^{(L-1)}_{:,:,i,m} \odot (A^{(L-1)}(\lambda^{(L-2)}_{:,:,i,m} \odot \Delta^{(L-2)}_{:,:,i,m})W^{(L-1)}))W^{(L)}_{:,m} \\ &\quad + A^{(L)}_{i,:}(\lambda^{(L-1)}_{:,:,i,m} \odot \Delta^{(L-1)}_{:,:,i,m})W^{(L)}_{:,m} \end{aligned} \tag{51}$$

If we continue this until we reach an expression with respect to X, we arrive at the following recursive equation:

$$f^U_{i,m}(X) = \Gamma^{(L)}_{i,m,i,m}(X) + \sum_{k=1}^{L-1} \Psi^{(k,k)}_{i,m,i,m} \tag{52}$$

where

$$\Gamma^{(k)}_{:,:,i,m}(X) = \begin{cases} A^{(k)}(\lambda^{(k-1)}_{:,:,i,m} \odot \Gamma^{(k-1)}_{:,:,i,m}(X))W^{(k)} & \text{if } k = [2, L] \\ A^{(k)}XW^{(k)} & \text{if } k = 1 \end{cases} \tag{53}$$

$$\Psi^{(k,\bar{k})}_{:,:,i,m} = \begin{cases} A^{(L-k+\bar{k})}(\lambda^{(L-k+\bar{k}-1)}_{:,:,i,m} \odot \Psi^{(k,\bar{k}-1)}_{:,:,i,m})W^{(L-k+\bar{k})} & \text{if } \bar{k} = [2, L-1] \\ A^{(L-k+\bar{k})}(\lambda^{(L-k+\bar{k}-1)}_{:,:,i,m} \odot \Delta^{(L-k)}_{:,:,i,m})W^{(L-k+\bar{k})} & \text{if } \bar{k} = 1 \end{cases} \tag{54}$$

However, we still need a method with which to select the elements of each $\lambda^{(k)}$ and $\Delta^{(k)}$. Recall from the proof of theorm 3.3 that we select the elements from $\alpha^{(k)}_U$ and $\beta^{(k)}_U$ by observing the signs of the elements multiplied with the values of layer k. We can do this by decomposing the above equation in Kronecker products.

Recall that $\text{vec}(AXB) = (A \otimes B^\top)\,\text{vec}(X)$, where vec is the row-major vector operator. Similarly, $\text{vec}(C \odot AXB) = \tilde{C}(A \otimes B^\top)\,\text{vec}(X)$, where $\tilde{C}$ is a diagonal vector such that $\text{diag}(\tilde{C}) = \text{vec}(C)$ Therefore, if we unroll $\Gamma^{(L)}_{i,m,i,m}$ until we reach the $\lambda^{(k)}$ term:

$$\Gamma^{(L)}_{i,m,i,m}(X) = A^{(L)}_{i,:}(\lambda^{(L-1)}_{:,:,i,m} \odot \Gamma^{(L-1)}_{:,:,i,m}(X))W^{(L)}_{:,m} \tag{55}$$

$$= (A^{(L)}_{i,:} \otimes W^{(L)\top}_{:,m}) \operatorname{vec}(\lambda^{(L-1)}_{:,:,i,m} \odot (A^{(L-1)}(\lambda^{(L-2)}_{:,:,i,m} \odot \Gamma^{(L-2)}_{:,:,i,m}(X))W^{(L-1)})) \tag{56}$$

$$= (A^{(L)}_{i,:} \otimes W^{(L)\top}_{:,m})\Lambda^{(L-1)}_{:,:,i,m}(A^{(L-1)} \otimes W^{(L-1)\top}) \operatorname{vec}(\lambda^{(L-2)}_{:,:,i,m} \odot \Gamma^{(L-2)}_{:,:,i,m}(X)) \tag{57}$$

$$= \operatorname{vec}(\lambda^{(k)}_{:,:,i,m} \odot \Gamma^{(k)}_{:,:,i,m}(X))^\top \left[\prod_{\bar{k}=k+1}^{L-1} (A^{(\bar{k})\top} \otimes W^{(\bar{k})})\Lambda^{(\bar{k})}_{:,:,i,m}\right] (A^{(L)\top}_{i,:} \otimes W^{(L)}_{:,m}) \tag{58}$$

We obtain equation 58 by transposing the entire equation and continuing to unroll identically until layer k. Therefore:

$$\xi^{(k)}_{j,n,i,m} = \left[\left[\prod_{\bar{k}=k+1}^{L-1} (A^{(\bar{k})\top} \otimes W^{(\bar{k})})\Lambda^{(\bar{k})}_{:,:,i,m}\right] (\hat{A}^\top_{i,:} \otimes W^{(L)}_{:,m})\right]_{j \cdot D_k + n} \tag{59}$$

$$\lambda^{(k)}_{j,n,i,m} = \begin{cases} \alpha^{(k)}_{U,j,n} & \text{if } \xi^{(k)}_{j,n,i,m} \geq 0; \\ \alpha^{(k)}_{L,j,n} & \text{if } \xi^{(k)}_{j,n,i,m} < 0. \end{cases} \qquad \Delta^{(k)}_{j,n,i,m} = \begin{cases} \beta^{(k)}_{U,j,n} & \text{if } \xi^{(k)}_{j,n,i,m} \geq 0; \\ \beta^{(k)}_{L,j,n} & \text{if } \xi^{(k)}_{j,n,i,m} < 0. \end{cases} \tag{60}$$

Because the right-most Kronecker product is a column vector, this matrix product leaves us with a vector who's components are the terms that are multiplied by the vectorized elements of the k-th layer, which determines how we map the j, n-th coordinate to the $\xi^{(k)}$ term. Note that we don't consider the Ψ terms in this selection because they do not contain any elements of X, and as such they will not affect the sign of the multiplied elements.

A near-identical proof can be applied to the lower bound as well, where we instead just flip whether we are selecting for the lower or upper bound of the previous layer in all the terms. As such, we can say that:

$$f^L_{i,m}(X) = \Sigma^{(L)}_{i,m,i,m}(X) + \sum_{k=1}^{L-1} \Phi^{(k,k)}_{i,m,i,m} \tag{61}$$

$$\Sigma^{(k)}_{:,:,i,m}(X) = \begin{cases} A^{(k)}(\omega^{(k-1)}_{:,:,i,m} \odot \Sigma^{(k-1)}_{:,:,i,m}(X))W^{(k)} & \text{if } k = [2, L] \\ A^{(k)}XW^{(k)} & \text{if } k = 1 \end{cases} \tag{62}$$

$$\Phi^{(k,\bar{k})}_{:,:,i,m} = \begin{cases} A^{(L-k+\bar{k})}(\omega^{(L-k+\bar{k}-1)}_{:,:,i,m} \odot \Phi^{(k,\bar{k}-1)}_{:,:,i,m})W^{(L-k+\bar{k})} & \text{if } \bar{k} = [2, L-1] \\ A^{(L-k+\bar{k})}(\omega^{(L-k+\bar{k}-1)}_{:,:,i,m} \odot \Theta^{(L-k)}_{:,:,i,m})W^{(L-k+\bar{k})} & \text{if } \bar{k} = 1 \end{cases} \tag{63}$$

$$\omega^{(k)}_{j,n,i,m} = \begin{cases} \alpha^{(k)}_{L,j,n} & \text{if } \xi^{(k)}_{j,n,i,m} \geq 0; \\ \alpha^{(k)}_{U,j,n} & \text{if } \xi^{(k)}_{j,n,i,m} < 0. \end{cases} \qquad \Theta^{(k)}_{j,n,i,m} = \begin{cases} \beta^{(k)}_{L,j,n} & \text{if } \xi^{(k)}_{j,n,i,m} \geq 0; \\ \beta^{(k)}_{U,j,n} & \text{if } \xi^{(k)}_{j,n,i,m} < 0. \end{cases} \tag{64}$$

Proof of Corollary 3.6

Using the explicit bounds from theorem 3.5, and re-using equation 58 where $k = 1$, we can say that:

$$u_{i,m}^{(L)} = \max_{X \in \mathbb{B}_p(X_0,\epsilon)} f_{i,m}^{U}(X) \tag{65}$$

$$= \max_{X \in \mathbb{B}_p(X_0,\epsilon)} \left[\Gamma_{i,m,i,m}^{(L)}(X) + \sum_{k=1}^{L-1} \Psi_{i,m,i,m}^{(k,k)} \right] \tag{66}$$

$$= \epsilon \max_{z \in \mathbb{B}_p(0,1)} \left[\Gamma_{i,m,i,m}^{(L)}(z) \right] + \Gamma_{i,m,i,m}^{(L)}(X_0) + \sum_{k=1}^{L-1} \Psi_{i,m,i,m}^{(k,k)} \tag{67}$$

$$= \epsilon \max_{z \in \mathbb{B}_p(0,1)} \left[\mathrm{vec}(\lambda_{:,:,i,m}^{(1)} \odot \Gamma_{:,:,i,m}^{(1)}(z))^\top \left[\prod_{\bar{k}=2}^{L-1} (A^{(\bar{k})\top} \otimes W^{(\bar{k})}) \Lambda_{:,:,i,m}^{(\bar{k})} \right] (A_{i,:}^{(L)\top} \otimes W_{:,m}^{(L)}) \right]$$
$$+ \Gamma_{i,m,i,m}^{(L)}(X_0) + \sum_{k=1}^{L-1} \Psi_{i,m,i,m}^{(k,k)} \tag{68}$$

$$= \epsilon \max_{z \in \mathbb{B}_p(0,1)} \left[\mathrm{vec}(\lambda_{:,:,i,m}^{(1)} \odot A^{(1)} z W^{(1)})^\top \left[\prod_{\bar{k}=2}^{L-1} (A^{(\bar{k})\top} \otimes W^{(\bar{k})}) \Lambda_{:,:,i,m}^{(\bar{k})} \right] (A_{i,:}^{(L)\top} \otimes W_{:,m}^{(L)}) \right]$$
$$+ \Gamma_{i,m,i,m}^{(L)}(X_0) + \sum_{k=1}^{L-1} \Psi_{i,m,i,m}^{(k,k)} \tag{69}$$

$$= \epsilon \max_{z \in \mathbb{B}_p(0,1)} \left[\left[\mathrm{vec}(z) \prod_{\bar{k}=1}^{L-1} (A^{(\bar{k})\top} \otimes W^{(\bar{k})}) \Lambda_{:,:,i,m}^{(\bar{k})} \right] (A_{i,:}^{(L)\top} \otimes W_{:,m}^{(L)}) \right]$$
$$+ \Gamma_{i,m,i,m}^{(L)}(X_0) + \sum_{k=1}^{L-1} \Psi_{i,m,i,m}^{(k,k)} \tag{70}$$

$$= \epsilon \left\| \left[\prod_{\bar{k}=1}^{L-1} (A^{(\bar{k})\top} \otimes W^{(\bar{k})}) \Lambda_{:,:,i,m}^{(\bar{k})} \right] (A_{i,:}^{(L)\top} \otimes W_{:,m}^{(L)}) \right\|_q$$
$$+ \Gamma_{i,m,i,m}^{(L)}(X_0) + \sum_{k=1}^{L-1} \Psi_{i,m,i,m}^{(k,k)} \tag{71}$$

A near-identical derivation for the lower bound can also be done, where instead we take the min of the $\mathbb{B}_p(X_0, \epsilon)$ instead of the max. This gives us:

$$l_{i,m}^{(L)} = -\epsilon \left\| \left[\prod_{\bar{k}=1}^{L-1} (A^{(\bar{k})\top} \otimes W^{(\bar{k})}) \Omega_{:,:,i,m}^{(\bar{k})} \right] (A_{i,:}^{(L)\top} \otimes W_{:,m}^{(L)}) \right\|_q + \Sigma_{i,m,i,m}^{(L)}(X_0) + \sum_{k=1}^{L-1} \Phi_{i,m,i,m}^{(k,k)} \tag{72}$$

Adaption of GCN Bilinear Form to Previous Works

When trying to compute a lower bound on the minimum adversarial distortion, it is certainly possible to simply alter the GCN equation into a form accepted by previous works [30, 14]. Specifically, using the notation from section 2.1, we note that:

$$H_{i,j}^{(l)} = \hat{A}_{i,:} H^{(l-1)} W_{:,j}^{(l)} = \sum_{k=1}^{N} \hat{A}_{i,k} \sum_{m=1}^{D_l} H_{k,m}^{(l-1)} W_{m,j}^{(l)} \tag{73}$$

$$= \sum_{k=1}^{N} \sum_{m=1}^{D_l} \hat{A}_{i,k} W_{m,j}^{(l)} H_{k,m}^{(l-1)} = \mathrm{vec}(\hat{A}_{i,:} \otimes W_{:,j}^{(l)})^\top \mathrm{vec}(H^{(l-1)}) \tag{74}$$

where $\otimes$ denotes outer product of two vectors and the row-major vectorization operator $\text{vec}(H) \coloneqq [H_{1,1}, \cdots, H_{1,D_l}, H_{2,1}, \cdots, H_{2,D_l}, \cdots, H_{N,1}, \cdots, H_{N,D_l}]^T$. Thus, it is possible to convert (1) to a feed forward network, with the following definitions:

$$\begin{aligned} \text{vec}(f(X)) = y^{(L)} \qquad & y^{(l)} = (\hat{A}^{(l)} \otimes W^{(l)\top}) H^{(l-1)} \\ H^{(l)} = \sigma(y^{(l)}) \qquad & H^{(0)} = \text{vec}(X) \end{aligned} \tag{75}$$

where $\otimes$ is the Kronecker product (generalized outer product). Note that the dimension of the matrix $\hat{A}^{(l)} \otimes W^{(l)\top} \in \mathbb{R}^{ND_l \times ND_{l-1}}$ is very large. Although previous methods can solve it directly [30, 14], it is very inefficient. For instance, [30] can compute the output bounds of a network in $O(L^2M^3)$ time, where M is the number of neurons per layer and L is the number of layers in the network. With this modification, this method would run in $O(L^2N^3D^3)$, where N is the number of nodes in a network and D is the number of features each layer per node.

Computational Improvements via Relaxation of Bounds

Relaxation on the bound derived in theorem 3.5 can occur when we say we can find a way to decompose the Hadamard product into ordinary matrix products. For instance, let us consider $\Gamma^{(k)}$ from theorem 3.5. Assuming that we can define a $D_L^{(k)}$ and $D_R^{(k)}$ such that $\lambda^{(k)} \odot C = D_L^{(k)} C D_R^{(k)}$:

$$\Gamma^{(k)}_{:,:,i,m}(X) = A^{(k)}(\lambda^{(k-1)}_{:,:,i,m} \odot \Gamma^{(k-1)}_{:,:,i,m}(X))W^{(k)} \tag{76}$$

$$= A^{(k)}(D_L^{(k-1)} \Gamma^{(k-1)}_{:,:,i,m}(X) D_R^{(k-1)})W^{(k)} \tag{77}$$

$$= A^{(k-1)'} \Gamma^{(k-1)}_{:,:,i,m}(X) W^{(k-1)'} \tag{78}$$

Where $A^{(k-1)'} = A^{(k)} D_L^{(k-1)}$ and $W^{(k-1)'} = D_R^{(k-1)} W^{(k)}$. Notice that now we've simply composed $\Gamma^{(k)}_{:,:,i,m}(X)$ into standard bilinear form. This process can be repeated until we have a bilinear expression with respect to X, which is straightforward to bound via lemma 3.2 or by using the dual norm. Additionally, this technique can be repeated for all other terms involving the Hadamard product in theorem 3.5. If we assume the complexity of finding such $D_L^{(k)}$ and $D_R^{(k)}$ matrices is negligible, bounding this network would essentially be a series of matrix multiplications that would result in a $O(N^2D + ND^2)$ complexity, where N is the number of nodes and D is the number of features per node at each layer. If we wish to iteratively compute these bounds for each layer as done by [30] and theorem 3.5, we would likely require a multiplicative factor of L^2 to the complexity. This would leave us with $O(L^2N^2D + L^2ND^2)$. Although this isn't a significant improvement over our two-layer bound if we assume $D, S, T \ll N$, it is a significant improvement in the many-layer case.

However, in the general case, finding a $D_L^{(k)}$ and $D_R^{(k)}$ such that $\lambda^{(k)} \odot C = D_L^{(k)} C D_R^{(k)}$ is not always possible. Therefore, we must settle for $\lambda^{(k)} \odot C \leq D_L^{(k)} C D_R^{(k)}$ when we are deriving an upper bound on the output, as this will only make the upper bound higher. (Similarly, for variables related to the lower bound, we'd allow decompositions that made the bound value lower.) For instance, we attempted one such relaxation that corresponded to simply taking the loosest $\lambda^{(k)}$ term across all nodes, and using the same values for each node in the network. However, this resulted in significantly larger output bounds. In future work, we suspect that some minimization problem could be formulated to efficiently find such $D_L^{(k)}$ and $D_R^{(k)}$ matrices that keep the bound relatively tight.