Formal Verification of Graph Convolutional Networks with Uncertain Node Features and Uncertain Graph Structure

Tobias Ladner TOBIAS.LADNER@TUM.DE
Michael Eichelbeck MICHAEL.EICHELBECK@TUM.DE
Matthias Althoff ALTHOFF@TUM.DE

School of Computation, Information and Technology
Technical University of Munich, Germany

Abstract

Graph neural networks are becoming increasingly popular in the field of machine learning due to their unique ability to process data structured in graphs. They have also been applied in safety-critical environments where perturbations inherently occur. However, these perturbations require us to formally verify neural networks before their deployment in safety-critical environments as neural networks are prone to adversarial attacks. While there exists research on the formal verification of neural networks, there is no work verifying the robustness of generic graph convolutional network architectures with uncertainty in the node features and in the graph structure over multiple message-passing steps. This work addresses this research gap by explicitly preserving the non-convex dependencies of all elements in the underlying computations through reachability analysis with (matrix) polynomial zonotopes. We demonstrate our approach on three popular benchmark datasets.

Keywords: Graph neural network, formal verification, set-based computing, matrix polynomial zonotope, uncertain message passing.

1 Introduction

A graph neural network extends the typical notion of feedforward neural networks to graph inputs (Kipf and Welling, 2017). Each node in the graph is associated with a feature vector, which is iteratively updated by exchanging information with neighboring nodes using their feature vectors over multiple message-passing steps. They have shown to achieve state-of-the-art results in a variety of fields (Wu et al., 2020), including advances in drug discovery (Zhang et al., 2021), recommender systems in social networks (Ying et al., 2018), and have also been applied in safety-critical environments such as cooperative autonomous driving (Chen et al., 2021).

However, it is well known that neural networks are sensitive to adversarial attacks (Goodfellow et al., 2015), where minor perturbations to the input can lead to unexpected predictions. Adversarial examples have also extensively been studied for graph neural networks (Dai et al., 2018; Günnemann, 2022), where both the node features and the graph structure can be perturbed. As graph neural networks are a generalization of many other network architectures to non-Euclidean input data (Bronstein et al., 2017), the existence of adversarial examples is not surprising. Thus, neural networks need to be formally verified before they can be safely deployed (Brix et al., 2023; König et al., 2024).

1.1 Related Work

Most state-of-the-art verifiers only consider standard, feedforward neural networks (Brix et al., 2023; König et al., 2024): These can generally be categorized into complete and incomplete algorithms (König et al., 2024). Complete algorithms (Huang et al., 2017; Katz et al., 2017) compute the exact output of a neural network given perturbations on the input. This allows one to either verify given specifications or to extract a counterexample. However, it has been shown that verifying a neural network with ReLU activations requires solving an exponential number of linear subproblems as this problem is NP-hard (Katz et al., 2017). Thus, many existing verifiers use incomplete but sound algorithms (Brix et al., 2023), which can verify given specifications by relaxing the problem; however, this relaxation might prevent them from extracting a counterexample when the specification could not be verified. These verifiers can again be categorized into optimization-based approaches and approaches using reachability analysis.

Optimization-based approaches formulate relaxed constraints for the activation functions in a neural network. This relaxed problem is then solved using satisfiability modulo theories (SMT) or mixed integer programming (MIP) solvers (Zhang et al., 2018; Katz et al., 2019; Müller et al., 2022; Tjeng et al., 2019; Dutta et al., 2018), or symbolic interval propagation (Henriksen and Lomuscio, 2020; Singh et al., 2019; Brix and Noll, 2020). These algorithms can be improved using branch-and-bound strategies (Bunel et al., 2020), where the problem is divided into simpler subproblems. For example, one can split ReLU neurons into their linear parts (Botoeva et al., 2020; Singh et al., 2018b). Such branch-and-bound strategies (Wang et al., 2021; Ferrari et al., 2022; Shi et al., 2023) are currently the dominant strategies in state-of-the-art verifiers (Brix et al., 2023).

On the other hand, one can use reachability analysis to verify a neural network by computing an enclosure of the output set. This is realized by propagating the perturbed input set through each layer of the neural network and bounding all approximation errors. Early approaches propagate convex set representations through neural networks, such as intervals (Pulina and Tacchella, 2010) and zonotopes (Gehr et al., 2018; Singh et al., 2018a). Non-convex set representations can improve the verification results as the exact output set can be non-convex due to the nonlinearity within the network. These approaches use Taylor models (Ivanov et al., 2021; Bogomolov et al., 2019; Huang et al., 2022), star sets (Bak, 2021; Lopez et al., 2023), and polynomial zonotopes (Kochdumper et al., 2023; Ladner and Althoff, 2023) to verify neural networks. Branch-and-bound strategies are also used in approaches using reachability analysis (Xiang et al., 2018).

To the best of our knowledge, there exist only a few approaches considering the formal verification of graph neural networks. As with feedforward neural networks (Katz et al., 2017), the theoretical limits of the graph neural networks verification problem have been discussed (Sälzer and Lange, 2023). Thus, most existing methods for verifying graph neural networks again employ incomplete but sound algorithms: Some approaches (Zügner and Günnemann, 2019; Bojchevski and Günnemann, 2019) formulate uncertainty in the semi-supervised node classification setting as an optimization problem, where uncertain node features (Zügner and Günnemann, 2019) and uncertainty in the graph structure (Bojchevski and Günnemann, 2019) are considered separately. The network architecture in the latter approach only has a single, slightly altered message-passing step. This approach is extended

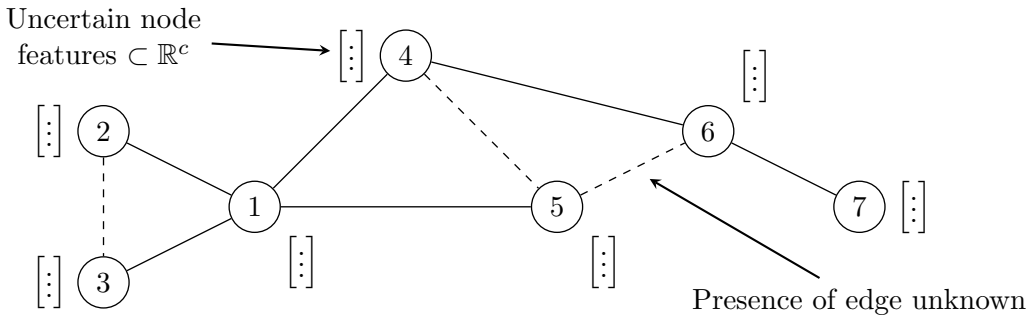

Figure 1: Graph $\mathcal{G}$ with uncertain node features and uncertain graph structure.

to restrict both the global and the local uncertainty of the graph (Jin et al., 2020a). Another approach (Wu et al., 2022) verifies uncertain node features in graph neural networks for job schedulers by unrolling them into feedforward neural networks and verifies them using reachability analysis. It is also worth mentioning that probabilistic guarantees can be achieved using randomized smoothing (Jia et al., 2020; Bojchevski et al., 2020), and one can try to defend adversarial attacks (Jin et al., 2020b); however, these approaches do not provide formal guarantees.

1.2 Contributions

Our contributions are as follows:

- We present the first approach to verify graph convolutional networks with uncertain node features and an uncertain graph structure as input (Fig. 1).
- The considered architecture of the graph convolutional network is generic and can have any element-wise activation function.
- Our approach allows us to verify the graph neural network over multiple message-passing steps given an uncertain graph input.
- We explicitly preserve the non-convex dependencies of all involved variables through all layers of the graph neural network using (matrix) polynomial zonotopes.
- Our verification algorithm has polynomial time complexity in the number of uncertain input features and in the number of uncertain edges.
- We demonstrate our approach on three popular benchmark datasets with added perturbations on the node features and the graph structure.
- Our approach will be made publicly available with the next release of CORA (Althoff, 2015).

This work is structured as follows: In Sec. 2, we introduce all required preliminaries and the problem statement, followed by defining the matrix variant of polynomial zonotopes in Sec. 3. Our verification approach is described in Sec. 4: We first show that graph-based

layers in neural networks can be computed exactly using matrix polynomial zonotopes with only uncertain input features. The required adaptations when also the graph structure is uncertain are described subsequently. Finally, we show experimental results in Sec. 5 and draw conclusions in Sec. 6.

2 Background

2.1 Notation

We denote scalars and vectors by lowercase letters, matrices by uppercase letters, and sets by calligraphic letters. The i-th element of a vector $v \in \mathbb{R}^n$ is written as $v_{(i)}$. The element in the i-th row and j-th column of a matrix $A \in \mathbb{R}^{n \times m}$ is written as $A_{(i,j)}$, the entire i-th row and j-th column are written as $A_{(i,\cdot)}$ and $A_{(\cdot,j)}$, respectively. The concatenation of A with a matrix $B \in \mathbb{R}^{n \times o}$ is denoted by $[A\ B] \in \mathbb{R}^{n \times (m+o)}$. The empty matrix is written as $[\,]$. We denote with I_n the identity matrix of dimension $n \in \mathbb{N}$. The symbols $\mathbf{0}$ and $\mathbf{1}$ refer to matrices with all zeros and ones of proper dimensions, respectively. Given $n \in \mathbb{N}$, we use the shorthand notation $[n] = \{1, \ldots, n\}$. The cardinality of a discrete set $\mathcal{D}$ is denoted by $|\mathcal{D}|$. Let $\mathcal{D} \subseteq [n]$, then $A_{(\mathcal{D},\cdot)}$ denotes all rows $i \in \mathcal{D}$ in lexicographic order; this is used analogously for columns. Let $\mathcal{S} \subset \mathbb{R}^n$ be a set and $f \colon \mathbb{R}^n \to \mathbb{R}^m$ be a function, then $f(\mathcal{S}) = \{f(x) \mid x \in \mathcal{S}\}$. An interval with bounds $a, b \in \mathbb{R}^n$ is denoted by $[a, b]$, where $a \leq b$ holds element-wise.

2.2 Neural Networks

Let us introduce the neural network architectures we consider in this work. We start by stating a general formalization of a neural network and, afterward, several types of layers.

Definition 1 (Neural Networks (Bishop and Nasrabadi, 2006, Sec. 5.1)) *Let $x \in \mathbb{R}^{n_0}$ be the input of a neural network Φ with κ layers, its output $y = \Phi(x) \in \mathbb{R}^{n_\kappa}$ is obtained as follows:*

$$\begin{aligned} h_0 &= x, \\ h_k &= L_k(h_{k-1}), \quad k \in [\kappa], \\ y &= h_\kappa, \end{aligned}$$

where $L_k \colon \mathbb{R}^{n_{k-1}} \to \mathbb{R}^{n_k}$ represents the operation of layer k.

Standard, non-graph-based neural networks are usually composed of alternating linear layers and nonlinear activation layers:

Definition 2 (Linear Layer) *A linear layer is defined by the operation*

$$h_k = L_k^{\mathrm{LIN}}(h_{k-1}) = W_k h_{k-1} + b_k,$$

with weight matrix $W_k \in \mathbb{R}^{n_k \times n_{k-1}}$, and bias vector $b_k \in \mathbb{R}^{n_k}$.

Definition 3 (Activation Layer) *An activation layer is defined by the operation*

$$h_k = L_k^{\mathrm{ACT}}(h_{k-1}) = \sigma_k(h_{k-1}),$$

where $\sigma_k(\cdot)$ is the respective element-wise nonlinear activation function, e.g., sigmoid or ReLU.

Graph neural networks generalize standard neural networks and additionally take a graph $\boldsymbol{\mathcal{G}} = (\mathcal{N}, \mathcal{E})$ as an input, where $\mathcal{N} \subset \mathbb{N}$ denotes the set of nodes and $\mathcal{E} \subseteq \mathcal{N} \times \mathcal{N}$ the set of edges of $\boldsymbol{\mathcal{G}}$. For each node $i \in \mathcal{N}$, we associate a feature vector $X_{(i,\cdot)} \in \mathbb{R}^{1 \times c_0}$ with c_0 input features, as illustrated in Fig. 1. These feature vectors of all $|\mathcal{N}|$ nodes are stacked vertically to obtain the input feature matrix $X \in \mathbb{R}^{|\mathcal{N}| \times c_0}$. Graph neural networks contain message-passing layers in which neighboring nodes exchange information. In this work, we consider the well-established graph convolutional layer (Kipf and Welling, 2017), which combines a node-level linear layer and a message-passing layer:

Definition 4 (Graph Convolutional Layer (Kipf and Welling, 2017, Eq. 2)) *Given are a weight matrix $W \in \mathbb{R}^{c_{k-1} \times c_k}$, an adjacency matrix $A \in \mathbb{R}^{|\mathcal{N}| \times |\mathcal{N}|}$ of a graph $\boldsymbol{\mathcal{G}}$, and an input $H_{k-1} \in \mathbb{R}^{|\mathcal{N}| \times c_{k-1}}$. Let $\tilde{A} = A + I_{|\mathcal{N}|}$ be the adjacency matrix with added self-loops and $\tilde{D} = \texttt{diag}(\mathbf{1}\tilde{A}) \in \mathbb{R}^{|\mathcal{N}| \times |\mathcal{N}|}$ be the diagonal degree matrix. The computation for a graph convolutional layer k is computed as*

$$H_k = L_k^{\mathrm{GC}}(H_{k-1}, \boldsymbol{\mathcal{G}}) = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} H_{k-1} W_k.$$

The term $P = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}$ computes the message passing between nodes. The adjacency matrix A can also be a weighted adjacency for graphs with scalar edge weights (Kipf and Welling, 2017, Sec. 7.2).

Please note that related verification approaches considering uncertainty in the graph structure (Bojchevski and Günnemann, 2019; Jin et al., 2020a) consider $\tilde{D}^{-1}\tilde{A}$ instead of $\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}$ in their message passing step. This is justified by the argument that it corresponds to the personalized page rank matrix, which has a similar spectrum. However, without appropriate approximation errors, how to verify the original graph neural network remains unknown using those approaches.

Depending on the use case, we let a graph neural network Φ return a node-level or graph-level output. For a node-level output, the output is simply the feature matrix of the last layer: $Y = \Phi(X, \boldsymbol{\mathcal{G}}) \in \mathbb{R}^{|\mathcal{N}| \times c_\kappa}$. For a graph-level output, we aggregate all node feature vectors into a single graph feature vector. Thus, $y = \Phi(X, \boldsymbol{\mathcal{G}}) \in \mathbb{R}^{n_\kappa}$. This is realized using a pooling layer, which is computed as follows:

Definition 5 (Global Pooling Layer) *A global pooling layer aggregates all node feature vectors $H_{k-1} \in \mathbb{R}^{|\mathcal{N}| \times c_{k-1}}$ within a graph $\boldsymbol{\mathcal{G}}$ into a single graph feature vector $h_k \in \mathbb{R}^{c_{k-1}}$ as follows:*

$$h_k = L_k^{\mathrm{GP}}(H_{k-1}, \boldsymbol{\mathcal{G}}) = \psi_k(H_{k-1}),$$

where $\psi_k(\cdot)$ denotes a permutation invariant aggregation function across all nodes, e.g., sum, mean, or maximum.

For example,

$$\psi_k(H_{k-1}) = (\mathbf{1}H_{k-1})^\top \tag{1}$$

computes a summation across all nodes in a global pooling layer k. For graph neural networks with a graph-level output, there can be regular linear and activation layers after the pooling layer.

2.3 Set-Based Computing

We verify neural networks using continuous sets. For an input set $\mathcal{X} \subset \mathbb{R}^{n_0}$ of a neural network Φ, the exact output set $\mathcal{Y}^* = \Phi(\mathcal{X})$ is computed by

$$\begin{aligned} \mathcal{H}_0^* &= \mathcal{X}, \\ \mathcal{H}_k^* &= L_k(\mathcal{H}_{k-1}^*), \quad k \in [\kappa], \\ \mathcal{Y}^* &= \mathcal{H}_\kappa^*. \end{aligned} \tag{2}$$

Polynomial zonotopes are a well-suited set representation to verify graph neural networks due to their polynomial computational complexity and precise outputs of the required operations. We briefly introduce polynomial zonotopes and all required operations, followed by an example.

Definition 6 (Polynomial Zonotope (Kochdumper and Althoff, 2020)) *Given are an offset $c \in \mathbb{R}^n$, a generator matrix of dependent generators $G \in \mathbb{R}^{n \times h}$, a generator matrix of independent generators $G_I \in \mathbb{R}^{n \times q}$, and an exponent matrix $E \in \mathbb{N}_0^{p \times h}$ with an identifier $\mathtt{id} \in \mathbb{N}^p$. A polynomial zonotope*[1] *$\mathcal{PZ} = \langle c, G, G_I, E \rangle_{PZ}$ is defined as*

$$\mathcal{PZ} \coloneqq \left\{ c + \sum_{i=1}^{h} \left(\prod_{k=1}^{p} \alpha_k^{E_{(k,i)}} \right) G_{(\cdot,i)} + \sum_{j=1}^{q} \beta_j G_{I(\cdot,j)} \;\middle|\; \alpha_k, \beta_j \in [-1, 1] \right\}.$$

The identifier $\mathtt{id}$ is used to keep track of the dependencies of the factors α_k between different polynomial zonotopes. Given two polynomial zonotopes $\mathcal{PZ}_1 = \langle c_1, G_1, G_{I,1}, E_1 \rangle_{PZ}, \mathcal{PZ}_2 = \langle c_2, G_2, G_{I,2}, E_2 \rangle_{PZ} \subset \mathbb{R}^n$, the Minkowski sum is computed by (Kochdumper and Althoff, 2020, Prop. 8)

$$\begin{aligned} \mathcal{PZ}_1 \oplus \mathcal{PZ}_2 &= \{ x_1 + x_2 \mid x_1 \in \mathcal{PZ}_1,\, x_2 \in \mathcal{PZ}_2 \} \\ &= \left\langle c_1 + c_2, \begin{bmatrix} G_1 & G_2 \end{bmatrix}, \begin{bmatrix} G_{I,1} & G_{I,2} \end{bmatrix}, \begin{bmatrix} E_1 & \mathbf{0} \\ \mathbf{0} & E_2 \end{bmatrix} \right\rangle_{PZ}, \end{aligned} \tag{3}$$

and given $A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m$, the affine map is computed by (Kochdumper and Althoff, 2020, Prop. 9)

$$A\mathcal{PZ}_1 + b = \{ Ax + b \mid x \in \mathcal{PZ}_1 \} = \langle Ac_1 + b, AG_1, AG_{I,1}, E_1 \rangle_{PZ}. \tag{4}$$

Crucially for our approach, the quadratic map can be evaluated exactly for polynomial zonotopes. The quadratic map is usually used to evaluate higher-order polynomials over polynomial zonotopes to enclose nonlinear functions.

Proposition 7 (Quadratic Map (Kochdumper, 2022, Prop. 3.1.30)) *Given two polynomial zonotopes $\mathcal{PZ}_1 = \langle c_1, G_1, [\,], E_1 \rangle_{PZ} \subset \mathbb{R}^{n_1}$, $\mathcal{PZ}_2 = \langle c_2, G_2, [\,], E_2 \rangle_{PZ} \subset \mathbb{R}^{n_2}$ with*

1. As in Kochdumper (2022), we adapt the definition from Kochdumper and Althoff (2020) and do not integrate the offset c into the generator matrix G and omit the identifier vector almost always for simplicity.

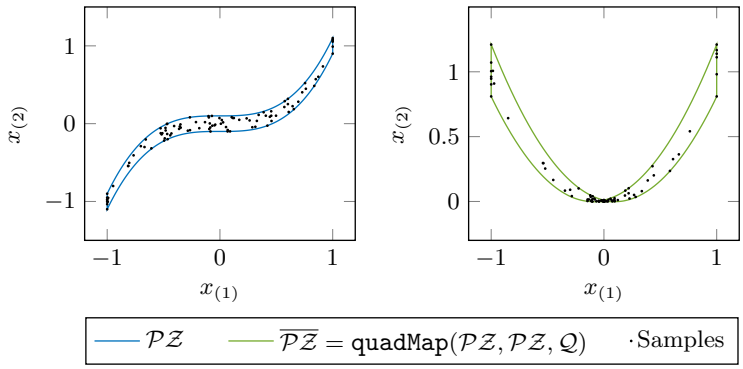

Figure 2: Visualization of the quadratic map using the polynomial zonotope $\mathcal{PZ}$ from Example 1.

h_1 and h_2 generators, respectively, a common identifier vector, and $\mathcal{Q} = \{Q_1, \ldots, Q_{\overline{n}}\}$, $Q_i \in \mathbb{R}^{n_1 \times n_2}$, then the quadratic map is computed as follows:

$$\overline{\mathcal{PZ}} = \texttt{quadMap}(\mathcal{PZ}_1, \mathcal{PZ}_2, \mathcal{Q}) = \left\{ \begin{bmatrix} x_1^\top Q_1 x_2 \\ \vdots \\ x_1^\top Q_{\overline{n}} x_2 \end{bmatrix} \;\middle|\; x_1 \in \mathcal{PZ}_1,\, x_2 \in \mathcal{PZ}_2 \right\}$$
$$= \left\langle \overline{c}, \begin{bmatrix} \widehat{G}_1 & \widehat{G}_2 & \overline{G}_1 & \ldots & \overline{G}_h \end{bmatrix}, [\,], \begin{bmatrix} E_1 & E_2 & \overline{E}_1 & \ldots & \overline{E}_h \end{bmatrix} \right\rangle_{PZ} \subset \mathbb{R}^{\overline{n}},$$

where

$$\overline{c} = \begin{bmatrix} c_1^\top Q_1 c_2 \\ \vdots \\ c_1^\top Q_{\overline{n}} c_2 \end{bmatrix},\; \widehat{G}_1 = \begin{bmatrix} c_2^\top Q_1^\top G_1 \\ \vdots \\ c_2^\top Q_{\overline{n}}^\top G_1 \end{bmatrix},\; \widehat{G}_2 = \begin{bmatrix} c_1^\top Q_1 G_2 \\ \vdots \\ c_1^\top Q_{\overline{n}} G_2 \end{bmatrix},\; \overline{G}_j = \begin{bmatrix} G_{1(\cdot,j)}^\top Q_1 G_2 \\ \vdots \\ G_{1(\cdot,j)}^\top Q_{\overline{n}} G_2 \end{bmatrix},$$

and $\overline{E}_j = E_2 + E_{1(\cdot,j)}\,\mathbf{1}$, $j \in [h_1]$. The output $\overline{\mathcal{PZ}}$ has $\mathcal{O}(h_1 h_2)$ generators.

In this work, we only use matrices Q_i with entries consisting of zeros and ones, which effectively selects which dimensions of the polynomial zonotopes are multiplied as part of a quadratic map. We illustrate this by an example:

Example 1 *Let us consider the set $\{\begin{bmatrix}\alpha_1 & \alpha_1^3{+}0.1\alpha_2 & \alpha_1^2\end{bmatrix}^\top \mid \alpha_1, \alpha_2 \in [-1, 1]\} \subset \mathbb{R}^3$. This set can be represented as a polynomial zonotope as follows:*

$$\mathcal{PZ} = \left\langle \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0.1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix}\,\end{bmatrix}, \begin{bmatrix} 1 & 3 & 0 & 2 \\ 0 & 0 & 1 & 0 \end{bmatrix} \right\rangle_{PZ},$$

which we visualize in Fig. 2 (left). The set $\{\begin{bmatrix}\alpha_1^3 & (\alpha_1^3{+}0.1\alpha_2)^2\end{bmatrix}^\top \mid \alpha_1, \alpha_2 \in [-1, 1]\} \subset \mathbb{R}^2$ can be computed from $\mathcal{PZ}$ using the quadratic map with $\mathcal{Q} = \{Q_1, Q_2\}$, where

$$Q_1 = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix},\; Q_2 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}.$$

Thus,

$$\overline{\mathcal{PZ}} = \texttt{quadMap}(\mathcal{PZ}, \mathcal{PZ}, \mathcal{Q}) = \left\{ \begin{bmatrix} x^\top Q_1 x \\ x^\top Q_2 x \end{bmatrix} \;\middle|\; x \in \mathcal{PZ} \right\}$$
$$= \left\langle \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0.2 & 0.01 \end{bmatrix}, \begin{bmatrix} \; \end{bmatrix}, \begin{bmatrix} 3 & 6 & 3 & 0 \\ 0 & 0 & 1 & 2 \end{bmatrix} \right\rangle_{PZ},$$

where we compacted $\overline{\mathcal{PZ}}$ by removing zero-length generators and adding generators whose dependent factors have equal exponents. The resulting $\overline{\mathcal{PZ}}$ is visualized in Fig. 2 (right).

Please note that the quadratic map in Prop. 7 is defined for polynomial zonotopes with a common identifier vector. We can adjust two polynomial zonotopes with different identifiers by extending the exponent matrix accordingly (Kochdumper, 2022, Prop. 3.1.5).

2.4 Verification of Feedforward Neural Networks

Finally, we briefly introduce the main steps to propagate a polynomial zonotope through a standard, non-graph-based neural network (Def. 1). Since the set propagation through a neural network (2) cannot be computed exactly in general, we have to enclose the output of each layer:

Proposition 8 (Image Enclosure (Kochdumper et al., 2023, Sec. 3)) *Let $\mathcal{H}_{k-1} \supseteq \mathcal{H}_{k-1}^* \subset \mathbb{R}^{n_{k-1}}$ be an input set to layer k, then*

$$\mathcal{H}_k = \texttt{enclose}(L_k, \mathcal{H}_{k-1}) \supseteq \mathcal{H}_k^* \subset \mathbb{R}^{n_k}$$

computes an outer-approximative output set. If the layer k is nonlinear (Def. 3), the output $\mathcal{H}_k$ has at most n_k more generators than $\mathcal{H}_{k-1}$.

Using polynomial zonotopes, the output of a linear layer can be computed exactly as stated in (4). However, the output of activation layers needs to be enclosed to obtain a sound outer approximation. We summarize the six main steps to enclose a nonlinear layer next and visualize them in Fig. 3: As we only consider element-wise activation functions, we can enclose each neuron individually (step 1). In step 2, we find bounds for our input set. Next, we approximate the activation function using a polynomial (step 3) and find an appropriate approximation error (step 4). Finally, the chosen polynomial is evaluated over the input set (step 5), and the output is enclosed using the approximation error (step 6), where one generator for each neuron of the current layer is added using (3). While we have depicted the steps in Fig. 3 using a polynomial of order one, higher-order polynomials can be used to obtain a tighter enclosure (Ladner and Althoff, 2023). The higher-order polynomials are evaluated over the input polynomial zonotopes using multiple applications of Prop. 7.

2.5 Problem Statement

Given an uncertain graph $\boldsymbol{\mathcal{G}} = (\mathcal{N}, \mathcal{E})$ with nodes $\mathcal{N} \subset \mathbb{N}$ and edges $\mathcal{E} = \mathcal{E}^* \cup \widetilde{\mathcal{E}} \subseteq \mathcal{N} \times \mathcal{N}$ consisting of fixed edges $\mathcal{E}^*$ and uncertain edges $\widetilde{\mathcal{E}}$, an uncertain input feature matrix $\mathcal{X} \subset$

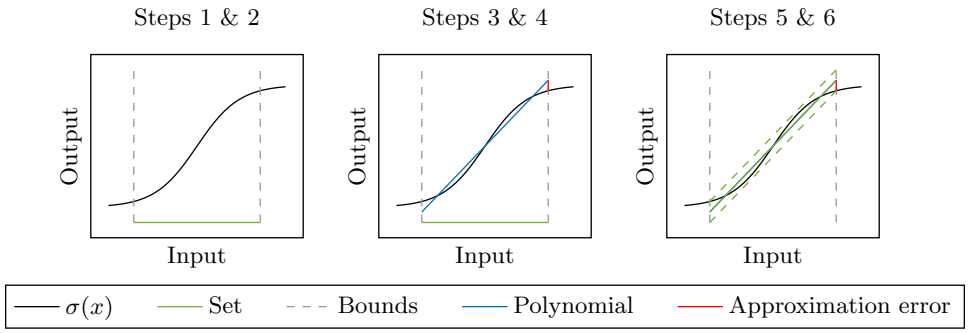

Figure 3: Main steps of enclosing a nonlinear layer. Step 1: Evaluate nonlinear function element-wise. Step 2: Find bounds of the input set. Step 3: Find an approximating polynomial. Step 4: Find the approximation error. Step 5: Evaluate polynomial over the input set. Step 6: Add the approximation error.

$\mathbb{R}^{|\mathcal{N}|\times c_0}$, a graph neural network Φ, and an unsafe set $\mathcal{S} \subset \mathbb{R}^{n_\kappa}$ where n_κ denotes the dimension of the output of Φ, we want to compute an outer-approximative output set $\mathcal{Y}$ such that it encloses the output for all possible graph inputs:

$$\forall \overline{\mathcal{E}} \subseteq \widetilde{\mathcal{E}}\colon \quad \Phi(\mathcal{X}, (\mathcal{N}, \mathcal{E}^* \cup \overline{\mathcal{E}})) \subseteq \mathcal{Y}.$$

We can then verify the given specification by showing that:

$$\mathcal{Y} \cap \mathcal{S} = \emptyset.$$

3 Matrix Polynomial Zonotopes

Before we present our approach, we introduce an extension to polynomial zonotopes, namely matrix polynomial zonotopes. Graph convolutional layers require a matrix $H_{k-1} \in \mathbb{R}^{|\mathcal{N}|\times c_{k-1}}$ as input (Def. 4) so that a set-based evaluation requires propagating uncertain matrices $\mathcal{H}_{k-1} \subset \mathbb{R}^{|\mathcal{N}|\times c_{k-1}}$ through all layers, which we want to represent as polynomial zonotopes; however, a (standard) polynomial zonotope is not defined for matrices (Def. 6). Thus, we define its matrix variant and a few required operations on them in this section. These operations are specifically tailored to facilitate the verification of graph neural networks; however, the concepts are generic and have applications elsewhere.

Definition 9 (Matrix Polynomial Zonotope) *Given are an offset $C \in \mathbb{R}^{n\times m}$, dependent generators $G \in \mathbb{R}^{n\times m\times h}$, independent generators $G_I \in \mathbb{R}^{n\times m\times q}$, and an exponent matrix $E \in \mathbb{N}_0^{p\times h}$ with an identifier $\mathtt{id} \in \mathbb{N}^p$. A matrix polynomial zonotope $\mathcal{PZ} = \langle C, G, G_I, E\rangle_{PZ}$ is defined as*

$$\mathcal{PZ} \coloneqq \left\{ C + \sum_{i=1}^{h} \left(\prod_{k=1}^{p} \alpha_k^{E_{(k,i)}} \right) G_{(\cdot,\cdot,i)} + \sum_{j=1}^{q} \beta_j G_{I(\cdot,\cdot,j)} \;\middle|\; \alpha_k, \beta_j \in [-1, 1] \right\}.$$

The Minkowski sum of two matrix polynomial zonotopes $\mathcal{PZ}_1 = \langle C_1, G_1, G_{I,1}, E_1 \rangle_{PZ}, \mathcal{PZ}_2 = \langle C_2, G_2, G_{I,2}, E_2 \rangle_{PZ} \subset \mathbb{R}^{n \times m}$, is computed analogously to (3):

$$\begin{aligned} \mathcal{PZ}_1 \oplus \mathcal{PZ}_2 &= \{x_1 + x_2 \mid x_1 \in \mathcal{PZ}_1,\, x_2 \in \mathcal{PZ}_2\} \\ &= \left\langle C_1 + C_2, \begin{bmatrix} G_1 & G_2 \end{bmatrix}, \begin{bmatrix} G_{I,1} & G_{I,2} \end{bmatrix}, \begin{bmatrix} E_1 & \mathbf{0} \\ \mathbf{0} & E_2 \end{bmatrix} \right\rangle_{PZ}, \end{aligned} \tag{5}$$

where the concatenation of the generators is along the last dimension. Given the matrices $A_1 \in \mathbb{R}^{k \times n}$, $A_2 \in \mathbb{R}^{m \times k}$, and the vectors $b_1 \in \mathbb{R}^{k \times m}$, $b_2 \in \mathbb{R}^{n \times k}$, an affine map is computed analogously to (4):

$$\begin{aligned} A_1 \mathcal{PZ}_1 + b_1 &= \{A_1 x + b_1 \mid x \in \mathcal{PZ}_1\} = \langle A_1 C_1 + b_1, A_1 G_1, A_1 G_{I,1}, E_1 \rangle_{PZ}, \\ \mathcal{PZ}_1 A_2 + b_2 &= \{x A_2 + b_2 \mid x \in \mathcal{PZ}_1\} = \langle C_1 A_2 + b_2, G_1 A_2, G_{I,1} A_2, E_1 \rangle_{PZ}, \end{aligned} \tag{6}$$

where the matrix multiplications are broadcast across all generators. Reshaping and transposing a matrix polynomial zonotope are computed by applying the respective operation on the center matrix and each generator matrix. In particular, reshaping a matrix polynomial zonotope into a vector by stacking it column-wise results in a standard polynomial zonotope, which we indicate by a vector decoration ($\vec{\square}$). This allows us, for example, to seamlessly use a matrix polynomial zonotope $\mathcal{H}_{k-1} \subset \mathbb{R}^{|\mathcal{N}| \times c_{k-1}}$ during the enclosure of an activation layer k by first reshaping it: $\vec{\mathcal{H}}_{k-1} \subset \mathbb{R}^{|\mathcal{N}| \cdot c_{k-1}}$, then obtain $\vec{\mathcal{H}}_k \subset \mathbb{R}^{|\mathcal{N}| \cdot c_k}$ using Prop. 8, and finally reshape it back to its original shape: $\mathcal{H}_k \subset \mathbb{R}^{|\mathcal{N}| \times c_k}$. During the verification of graph neural networks, we also require the computation of matrix multiplication on two matrix polynomial zonotopes. This operation can be computed using Prop. 7 without inducing additional outer approximations:

Lemma 10 (Matrix Multiplication on Matrix Polynomial Zonotopes) *Given two matrix polynomial zonotopes $\mathcal{M}_1 \subset \mathbb{R}^{n \times k}$, $\mathcal{M}_2 \subset \mathbb{R}^{k \times m}$ with h_1 and h_2 generators, respectively, then the matrix multiplication*

$$\mathcal{M}_3 = \mathcal{M}_1 \boxdot \mathcal{M}_2 = \{(M_1 M_2) \mid M_1 \in \mathcal{M}_1, M_2 \in \mathcal{M}_2\},$$

is obtained by

$$\vec{\mathcal{M}}_3 = \texttt{quadMap}\Big(\vec{\mathcal{M}}_1, \vec{\mathcal{M}}_2, \mathcal{Q}\Big) \subset \mathbb{R}^{n \cdot m},$$

where $\mathcal{Q} = \{Q_{1,1}, Q_{2,1}, \ldots, Q_{n,1}, Q_{1,2}, \ldots, Q_{n,m}\}$. Let $v_i = [i \ \ldots \ n(k-1)+i]$ and $w_j = [k(j-1)+1 \ \ldots \ k(j-1)+k]$ be the respective indices involved to compute the (i,j)-th entry, then

$$Q_{i,j} = \texttt{sparse}(v_i, w_j, nk, km) \in \mathbb{R}^{(nk) \times (km)}$$

with ones in positions $(v_{i(l)}, w_{j(l)})$, $\forall l \in [k]$, and zeros otherwise. The output $\mathcal{M}_3$ has $\mathcal{O}(h_1 h_2)$ generators.

Proof The statement follows directly from Prop. 7 and the construction of $Q_{i,j}$. The number of generators also directly follows from Prop. 7. ■

We want to stress that Lemma 10 can be efficiently computed using matrix broadcasting, as effectively the center matrix and each generator matrix from one set is multiplied with the center matrix and each generator matrix of the other set. This broadcasting is also parallelizable and efficiently computed on a GPU, which makes our approach scalable.

4 Formal Verification of Graph Convolutional Networks

In this section, we demonstrate how to generalize the verification of standard neural networks (Kochdumper et al., 2023; Ladner and Althoff, 2023) to graph convolutional networks. We start by (i) explaining how to verify graph neural networks that have only uncertain node features, and then (ii) describe the adaptations where, additionally, the graph structure is unknown. Moreover, we show (iii) how a subgraph can be efficiently extracted in cases where not the entire graph is relevant to verify the specification.

4.1 Verification with Uncertain Node Features

Uncertainty in the node features requires us to define how the graph-specific layers can be enclosed for an uncertain input. Using matrix polynomial zonotopes, the enclosure of a graph convolutional layer (Def. 4) does not induce any additional outer approximation.

Proposition 11 (Enclosure of Graph Convolutional Layer) *Given are a weight matrix $W_k \in \mathbb{R}^{c_{k-1} \times c_k}$, a graph $\boldsymbol{\mathcal{G}} = (\mathcal{N}, \mathcal{E})$, and an input $\mathcal{H}_{k-1} \subset \mathbb{R}^{|\mathcal{N}| \times c_{k-1}}$ represented as a matrix polynomial zonotope. Let $A \in \mathbb{R}^{|\mathcal{N}| \times |\mathcal{N}|}$ be the adjacency matrix of $\boldsymbol{\mathcal{G}}$, $\tilde{A} = A + I_{|\mathcal{N}|}$, and let $\tilde{D} = \texttt{diag}(\mathbf{1}\tilde{A}) \in \mathbb{R}^{|\mathcal{N}| \times |\mathcal{N}|}$ be the diagonal degree matrix. The exact output of a graph convolutional layer k in Def. 4 is computed by*

$$\mathcal{H}_k = L_k^{\mathrm{GC}}(\mathcal{H}_{k-1}) = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} \mathcal{H}_{k-1} W_k.$$

Proof As the graph convolutional layer is composed of a left and a right matrix multiplication, the computation is exact using (6). ■

The enclosure of a pooling layer (Def. 5) with a summation as aggregation function as in (1) is obtained analogously.

Proposition 12 (Enclosure of Summation Pooling Layer) *Given are a graph $\boldsymbol{\mathcal{G}}$ and an input $\mathcal{H}_{k-1} \subset \mathbb{R}^{|\mathcal{N}| \times c_{k-1}}$ represented as a matrix polynomial zonotope. The exact output of a pooling across all nodes via summation is computed by*

$$\mathcal{H}_k = L_k^{\mathrm{GP}}(\mathcal{H}_{k-1}, \boldsymbol{\mathcal{G}}) = (\mathbf{1}\mathcal{H}_{k-1})^\top.$$

Proof As the pooling layer is computed by a left matrix multiplication, the computation is exact using (6). ■

Thus, the graph-based layers can be computed without inducing additional outer approximations using matrix polynomial zonotopes when we only have uncertain node features.

4.2 Verification with Uncertain Graph Structure

Verifying graph neural networks becomes more difficult if the presence of some edges is unknown in an uncertain graph $\boldsymbol{\mathcal{G}}$. This case requires us to enclose the outputs of all possible graph inputs (Sec. 2.5). We enclose these outputs by computing an outer-approximative output set of an equivalent graph with uncertain edge weights: Let $\boldsymbol{\mathcal{G}}$ have fixed edges $\mathcal{E}^*$

and uncertain edges $\widetilde{\mathcal{E}}$. Then, we set the edge weight to 1 for edges in $\mathcal{E}^*$ and to $[0,1]$ for edges in $\widetilde{\mathcal{E}}$. This uncertainty requires a set-based evaluation of the message passing $P = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$ in graph convolutional layers (Def. 4). In particular, we now have an uncertain (weighted) adjacency matrix $\mathcal{A} \subset \mathbb{R}^{|\mathcal{N}|\times|\mathcal{N}|}$ containing the respective edge weights, which in turn leads to an uncertain degree matrix $\tilde{\mathcal{D}} \subset \mathbb{R}^{|\mathcal{N}|\times|\mathcal{N}|}$, and eventually, an uncertain message passing

$$\mathcal{P}^* = \tilde{\mathcal{D}}^{-\frac{1}{2}}\tilde{\mathcal{A}}\tilde{\mathcal{D}}^{-\frac{1}{2}}. \tag{7}$$

Please note that analogous holds if the graph has uncertain scalar edge weights. Subsequently, we detail the required steps to compute an enclosure of the message passing $\mathcal{P} \supseteq \mathcal{P}^*$ using matrix polynomial zonotopes (Def. 9). Please compare these steps with the definition of a graph convolutional layer (Def. 4). We construct the uncertain adjacency matrix as a matrix polynomial zonotope $\mathcal{A} \subset \mathbb{R}^{|\mathcal{N}|\times|\mathcal{N}|}$, where each generator of $\mathcal{A}$ corresponds to one uncertain edge. Then,

$$\tilde{\mathcal{A}} = \mathcal{A} + I_{|\mathcal{N}|} \tag{8}$$

adds self-loops to the adjacency matrix. Analogously to Prop. 12, we compute the diagonal entries of the degree matrix $\tilde{\mathcal{D}}$ by summing across all rows of $\tilde{\mathcal{A}}$ using (6):

$$\tilde{\mathcal{D}}_{\text{diag}} = (\mathbf{1}\tilde{\mathcal{A}})^\top. \tag{9}$$

To obtain $\tilde{\mathcal{D}}^{-\frac{1}{2}}$, we note that the inverse of a diagonal matrix is given by the inverse of each entry on the main diagonal. Additionally, we are required to compute the square root of each entry individually. However, polynomial zonotopes are not closed under these operations. Thus, we enclose the output of the inverse square root function using Prop. 8. The function is already applied element-wise, hence it suffices to provide an appropriate approximation error:

Lemma 13 (Approximation Error of Inverse Square Root) *Given a polynomial $p(x) = ax + b$ and bounds $[l, u] \subset \mathbb{R}_+$, then the maximum approximation error*

$$d = \max_{x\in[l,u]} |f(x) - p(x)| = |f(x^*) - p(x^*)|,$$

where

$$x^* \in \left\{ l, \sqrt[3]{(1/2a)^2}, u \right\} \cap [l, u].$$

Proof The approximation error d has to lie on the extreme point:

$$\begin{aligned}
& \frac{d}{dx}\,(f(x) - p(x)) \stackrel{!}{=} 0 \\
\iff \quad & -\frac{1}{2}x^{-\frac{3}{2}} - a = 0 \\
\iff \quad & x^{-\frac{3}{2}} = -2a \\
\implies \quad & x = \sqrt[3]{(1/2a)^2},
\end{aligned}$$

or on a boundary point l, u if the extreme point lies outside $[l, u]$ due to monotonicity. ■

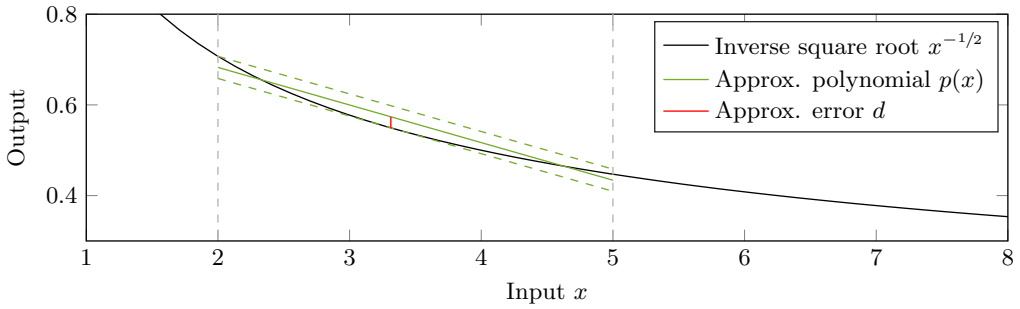

Figure 4: Enclosure of the inverse square root function. The x-axis corresponds to the degree of a node in $\tilde{D}$, and the y-axis to the respective entry in $\tilde{D}^{-\frac{1}{2}}$.

An example of the enclosure of the inverse square root function for a polynomial found via regression is shown in Fig. 4. A tighter enclosure can be obtained using higher-order polynomials (Ladner and Althoff, 2023). Thus, we can enclose the diagonal entries of the degree matrix using Prop. 8:

$$\widehat{\mathcal{D}}_{\text{diag}}^{-\frac{1}{2}} = \texttt{enclose}\Big(x \mapsto x^{-\frac{1}{2}}, \tilde{\mathcal{D}}_{\text{diag}}\Big) \supseteq \tilde{\mathcal{D}}_{\text{diag}}^{-\frac{1}{2}}, \tag{10}$$

and place the entries $\widehat{\mathcal{D}}_{\text{diag}}^{-\frac{1}{2}}$ on the main diagonal of

$$\widehat{\mathcal{D}}^{-\frac{1}{2}} = \texttt{diag}\left(\widehat{\mathcal{D}}_{\text{diag}}^{-\frac{1}{2}}\right) \supseteq \tilde{\mathcal{D}}^{-\frac{1}{2}}. \tag{11}$$

This is computed by first projecting $\widehat{\mathcal{D}}_{\text{diag}}^{-\frac{1}{2}} \subset \mathbb{R}^{|\mathcal{N}|}$ into a higher-dimensional space with zeros in the new dimensions:

$$\vec{\widehat{\mathcal{D}}}^{-\frac{1}{2}} = I_{|\mathcal{N}|^2(\cdot,\mathcal{K})} \widehat{\mathcal{D}}_{\text{diag}}^{-\frac{1}{2}} \subset \mathbb{R}^{|\mathcal{N}|^2}, \tag{12}$$

where $\mathcal{K} = \{1, |\mathcal{N}| + 2, \ldots, |\mathcal{N}|^2\}$ contains the indices of the diagonal entries of a diagonal matrix, and then reshaping the polynomial zonotope to obtain $\widehat{\mathcal{D}}^{-\frac{1}{2}} \subset \mathbb{R}^{|\mathcal{N}| \times |\mathcal{N}|}$. To obtain the entire uncertain message passing $\mathcal{P}$, we compute the matrix multiplication on the involved matrix polynomial zonotopes $\widehat{\mathcal{D}}^{-\frac{1}{2}}$ and $\tilde{\mathcal{A}}$ using Lemma 10:

Proposition 14 (Enclosure of Uncertain Message Passing) *Given an uncertain adjacency matrix $\mathcal{A}$ with h generators, then*

$$\mathcal{P} = \widehat{\mathcal{D}}^{-\frac{1}{2}} \boxdot \tilde{\mathcal{A}} \boxdot \widehat{\mathcal{D}}^{-\frac{1}{2}} \supseteq \mathcal{P}^*,$$

encloses the message passing with $\mathcal{O}(h^3)$ generators.

Proof The enclosure is computed using a set-based evaluation of the message passing in Def. 4 using (8)–(12) and Lemma 10. These steps are computed using affine maps (6) and matrix multiplications of polynomial zonotopes (Lemma 10), which are exact, and

the enclosure of $\tilde{\mathcal{D}}_{\text{diag}}$ using Prop. 8 with the approximation error in Lemma 13, which is outer-approximative. Thus, the enclosure of the message passing is sound.

Number of generators: Affine maps do not increase the number of generators (6). The enclosure of $\tilde{\mathcal{D}}_{\text{diag}}$ in (10) adds one generator for each node with an uncertain degree (Prop. 8), which are at most $2h$ as each uncertain edge in $\mathcal{A}$ has two adjacent nodes. Finally, two applications of the matrix multiplication on matrix polynomial zonotopes (Lemma 10) obtains the $\mathcal{O}(h^3)$ generators of $\mathcal{P}$. ■

After obtaining the uncertain message passing $\mathcal{P}$, we can enclose the output set of a graph convolutional layer as follows:

Proposition 15 (Enclosure of Graph Convolutional Layer) *Given are a weight matrix $W_k \in \mathbb{R}^{c_{k-1} \times c_k}$, an uncertain graph $\boldsymbol{\mathcal{G}}$, and an uncertain input $\mathcal{H}_{k-1} \in \mathbb{R}^{|\mathcal{N}| \times c_{k-1}}$ with h_1 generators. Let $\mathcal{P} \subset \mathbb{R}^{|\mathcal{N}| \times |\mathcal{N}|}$ be the uncertain message passing according to Prop. 14 with $\mathcal{O}(h_2^3)$ generators. The output for a graph convolutional layer k (Def. 4) is enclosed by*

$$\mathcal{H}_k = \texttt{enclose}\big(L_k^{\text{GC}}, \mathcal{H}_{k-1}, \mathcal{P}\big) = (\mathcal{P} \boxdot \mathcal{H}_{k-1}) W_k \ \subseteq \ L_k^{\text{GC}}(\mathcal{H}_{k-1}, \boldsymbol{\mathcal{G}}),$$

with $\mathcal{O}(h_1 h_2^3)$ generators.

Proof The enclosure follows directly from the enclosure of the message passing (Prop. 14), the matrix multiplication on polynomial zonotopes (Lemma 10), and the affine map (6). Given the number of generators of $\mathcal{H}_{k-1}$ and $\mathcal{P}$, the number of generators of $\mathcal{H}_k$ follows from Lemma 10. ■

Our approach defines the enclosure layer-wise and thus realizes an arbitrary concatenation of the considered layers. To demonstrate the polynomial time complexity in the number of uncertain edges and input features for an entire graph neural network with multiple message-passing steps, let us consider Alg. 1: The graph neural network has κ' message-passing steps, each consisting of one graph convolutional layer and one activation layer (lines 3 to 6). For networks with a node-level output, the output of the last message-passing step is directly the output of the network. For networks with a graph-level output, the output is passed to a global pooling layer and optionally followed by standard, non-graph-based layers (lines 11 to 14). With this algorithm, we can state the main theorem of this work:

Theorem 16 *Given a neural network Φ with κ layers and κ' message passing steps, an uncertain graph $\boldsymbol{\mathcal{G}} = (\mathcal{N}, \mathcal{E})$ with $|\mathcal{N}|$ nodes and h_e uncertain edges, and an uncertain input $\mathcal{X} \subset \mathbb{R}^{|\mathcal{N}| \times c_0}$ with h_x generators, then Alg. 1 satisfies the problem statement in Sec. 2.5. More specifically, the number of generators of the computed output $\mathcal{Y}$ is given by:*

$$h_y \in \mathcal{O}\Big(h_e^{3\kappa'}(h_x + |\mathcal{N}| c_{\max}) + (\kappa - 2\kappa') n_{\max}\Big),$$

where $c_{\max} \coloneqq \max_{k' \in [\kappa']} c_{2k'}$ denotes the maximum number of features within the graph layers and $n_{\max} \coloneqq \max_{k \in \{2\kappa'+2, \dots, \kappa\}} n_k$ denote the maximum number of output neurons of the non-graph-based layers after the global pooling layer.

Algorithm 1 Enclosing the Output of a Graph Neural Network

Require: Neural network Φ, number of layers κ, number of message passing steps κ', input set $\mathcal{X}$, graph $\boldsymbol{\mathcal{G}}$.

1: $\mathcal{H}_0 \leftarrow \mathcal{X}$
2: $\mathcal{P} \leftarrow$ Compute message passing based on $\boldsymbol{\mathcal{G}}$ ▷ Prop. 14
3: **for** $k' = 2, \ldots, 2\kappa'$ **do** ▷ Graph-based layers
4: $\quad \mathcal{H}_{k'-1} \leftarrow \texttt{enclose}\big(L^{\text{GC}}_{k'-1}, \mathcal{H}_{k'-2}, \mathcal{P}\big)$ ▷ Prop. 15
5: $\quad \mathcal{H}_{k'} \leftarrow \texttt{enclose}\big(L^{\text{ACT}}_{k'}, \mathcal{H}_{k'-1}\big)$ ▷ Prop. 8
6: **end for**
7: **if** $\kappa = 2\kappa'$ **then** ▷ Graph-level output
8: $\quad \mathcal{Y} \leftarrow \mathcal{H}_\kappa$
9: **else** ▷ Node-level output
10: $\quad \mathcal{H}_{2\kappa'+1} \leftarrow L^{\text{GP}}_{2\kappa'+1}(\mathcal{H}_{2\kappa'}, \boldsymbol{\mathcal{G}})$ ▷ Global pooling layer, Prop. 12
11: $\quad$ **for** $k = 2\kappa' + 2, 2\kappa' + 4, \ldots, \kappa$ **do** ▷ Standard, non-graph-based layers
12: $\quad\quad \mathcal{H}_k \leftarrow L^{\text{LIN}}_k(\mathcal{H}_{k-1})$ ▷ Def. 2
13: $\quad\quad \mathcal{H}_{k+1} \leftarrow \texttt{enclose}\big(L^{\text{ACT}}_{k+1}, \mathcal{H}_k\big)$ ▷ Prop. 8
14: $\quad$ **end for**
15: $\quad \mathcal{Y} \leftarrow \mathcal{H}_\kappa$
16: **end if**
17: **return** Enclosure of output set $\mathcal{Y} \supseteq \mathcal{Y}^*$

Proof The problem statement is satisfied as each step to compute $\mathcal{Y}$ is either exact (Prop. 12, (4)) or outer-approximative (Prop. 14, Prop. 15, and Prop. 8), and the specification can be checked as in previous approaches using polynomial zonotopes (Kochdumper et al., 2023; Ladner and Althoff, 2023). The message passing $\mathcal{P}$ has $\mathcal{O}(h_e^3)$ generators (Prop. 14). The enclosure of a nonlinear layer adds at most one generator for each output neuron (Prop. 8). The global pooling layer (Prop. 12) and linear layers (4) do not change the number of generators. Thus, the number of generators of $\mathcal{Y}$ in Alg. 1 is:

$$
\begin{aligned}
h_y \in \mathcal{O}\Bigg(& \overbrace{h_e^3 \cdot \big(h_e^3 \cdot (\cdots \underbrace{h_e^3 \cdot h_x}_{\text{(Prop. 15)}} + \underbrace{|\mathcal{N}|c_2}_{\text{(Prop. 8)}} \cdots) + |\mathcal{N}|c_{2\kappa'-2}\big) + |\mathcal{N}|c_{2\kappa'}}^{\kappa' \text{ message passing steps (lines 3 to 6)}} + \overbrace{\frac{1}{2}\sum_{k=2\kappa'+2}^{\kappa} \underbrace{n_k}_{\text{(Prop. 8)}}}^{\text{(lines 11 to 14)}} \Bigg) \\
= \mathcal{O}\Bigg(& (h_e^3)^{\kappa'} h_x + \underbrace{(h_e^3)^{\kappa'-1}|\mathcal{N}|c_2 + \cdots + (h_e^3)^1 |\mathcal{N}|c_{2\kappa'-2} + |\mathcal{N}|c_{2\kappa'}}_{\text{Polynomial of order } \kappa'-1} + \frac{1}{2}\sum_{k=2\kappa'+2}^{\kappa} n_k \Bigg) \\
\subseteq \mathcal{O}\Bigg(& (h_e^3)^{\kappa'} h_x + (h_e^3)^{\kappa'-1} \max_{k' \in [2\kappa']} |\mathcal{N}|c_{k'} + \frac{1}{2}\sum_{k=2\kappa'+2}^{\kappa} n_k \Bigg) \\
\subseteq \mathcal{O}\Bigg(& h_e^{3\kappa'} \Bigg(h_x + \max_{k \in [2\kappa']} |\mathcal{N}|c_{2k'} \Bigg) + \frac{1}{2}\sum_{k=2\kappa'+2}^{\kappa} n_k \Bigg) = \widetilde{\mathcal{H}}_y.
\end{aligned}
$$

Next, we simplify the term by bounding the number of output neurons with their maximum:

$$\begin{aligned}\widetilde{\mathcal{H}}_y &\subseteq \mathcal{O}\Big(h_e^{3\kappa'}\Big(h_x + |\mathcal{N}|c_{\max}\Big) + \frac{1}{2}\sum_{k=2\kappa'+2}^{\kappa} n_{\max}\Big)\\ &\subseteq \mathcal{O}\Big(h_e^{3\kappa'}(h_x + |\mathcal{N}|c_{\max}) + (\kappa - 2\kappa')n_{\max}\Big),\end{aligned}$$

which shows that $h_y \in \widetilde{\mathcal{H}}_y \subseteq \mathcal{O}\Big(h_e^{3\kappa'}(h_x + |\mathcal{N}|c_{\max}) + (\kappa - 2\kappa')n_{\max}\Big)$. ■

Please note that all involved operations on polynomial zonotopes to compute the output set $\mathcal{Y}$ (affine map, Minkowski sum, and quadratic map) have polynomial time complexity (Kochdumper, 2022, Tab. 3.2), and that the time complexity is dominated by the number of generators resulting from the applied quadratic map operations. Thus, it follows directly from Thm. 16 that Alg. 1 has polynomial time complexity in the number of uncertain input features h_x and uncertain edges h_e compared to an exponential complexity when all 2^{h_e} possible graphs need to be verified individually. While our approach is exponential in the number of message-passing steps κ', we want to stress that κ' is usually small to avoid over-smoothing (Chen et al., 2020). To further improve the scalability of our approach, the number of generators can be limited using order reduction methods (Ladner and Althoff, 2024; Kochdumper, 2022, Prop. 3.1.39) at the cost of additional outer approximations. Additionally, we want to stress that many involved operations can be parallelized and efficiently be computed on a GPU, in particular the matrix multiplication on matrix polynomial zonotopes (Lemma 10).

Let us demonstrate our approach for verifying graph neural networks by a small example:

Example 2 *Let Φ be a neural network with input X, graph $\boldsymbol{\mathcal{G}}$, and output Y computed by two layers:*

$$\begin{aligned}H_1 &= L_1^{\mathrm{GC}}(X, \boldsymbol{\mathcal{G}}),\\ Y &= L_2^{\mathrm{GC}}(H_1, \boldsymbol{\mathcal{G}}),\end{aligned} \qquad \text{with } W_1 = W_2 = I_2.$$

The input graph $\boldsymbol{\mathcal{G}} = (\mathcal{N}, \mathcal{E})$ is chosen as

$$\mathcal{N} = \Big\{①, ②, ③\Big\}, \qquad \mathcal{E} = \Big\{①\text{–}②, ①\text{–}③, ②\text{–}③\Big\},$$

and the input features for each node are

$$\mathcal{X}_{(1,\cdot)} = \begin{bmatrix}[0.9, 1.1]\\ [0.9, 1.1]\end{bmatrix}^\top,\ X_{(2,\cdot)} = X_{(3,\cdot)} = \begin{bmatrix}1\\ 1\end{bmatrix}^\top.\ \text{Thus, } \mathcal{X} = \begin{bmatrix}\mathcal{X}_{(1,\cdot)}\\ X_{(2,\cdot)}\\ X_{(3,\cdot)}\end{bmatrix}.$$

Let us now consider the presence of the edge ①–③ as unknown during the evaluation of $\mathcal{Y}^* = \Phi(\mathcal{X}, \boldsymbol{\mathcal{G}})$. Thus, the uncertainty of the features of node ① is passed to node ③ after one message passing step if the edge ①–③ is present (in $\mathcal{H}_1^* = L_1^{\mathrm{GC}}(X, \boldsymbol{\mathcal{G}})$), and after two steps otherwise (in $\mathcal{Y}^* = L_2^{\mathrm{GC}}(H_1, \boldsymbol{\mathcal{G}})$ via ②).

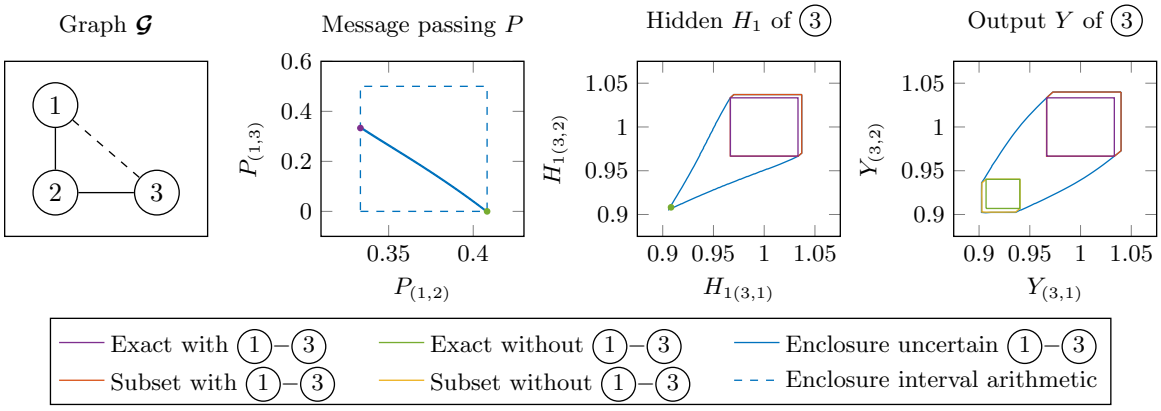

Figure 5: Visualization of Example 2. Our approach allows a tight enclosure of the output with uncertain input graph $\mathcal{G}$.

Example 2 is visualized in Fig. 5: The input set $\mathcal{X}$ given as an interval is converted to a (matrix) polynomial zonotope (Kochdumper, 2022, Prop. 3.1.10). We can obtain the exact output set for either case by propagating the respective graph through the network (purple and green) as well as their enclosure using our approach (Thm. 16, blue). Please note that we explicitly preserve the dependencies between the considered sets via the identifier vector of a matrix polynomial zonotope (Def. 9). We can visualize the preserved dependencies in the enclosure of the uncertain edge: By plugging -1 and 1 into the dependent factor α_k corresponding to the uncertain edge, we obtain the subset (Kochdumper, 2022, Prop. 3.1.43) corresponding to the respective case (orange and yellow). This demonstrates the tightness of our approach.

Additionally, we show the respective message passing P from node ① to the nodes ② and ③ for each case (purple and green) as well as their enclosure (blue), where we use a polynomial of order 2 to enclose $\widehat{\mathcal{D}}_{\text{diag}}^{-\frac{1}{2}}$ in (10). While the message passing from node ① to ③ trivially becomes 0 if we remove that edge, the message passing from node ① to ② also changes due to the normalization during the computation of P through the degree matrix. Moreover, we want to point out that the enclosure $\mathcal{P}$ is a non-convex, slightly bent stripe. Please note that the enclosure of the output $\mathcal{Y}$ can also be non-convex in general. For comparison, we include an enclosure of the uncertain message passing $\mathcal{P}^*$ using interval arithmetic (Jaulin et al., 2001) in Fig. 5. We omit the enclosure of $\mathcal{H}_1^*$ and $\mathcal{Y}^*$ using interval arithmetic as the obtained intervals are so large that the results using our approach described above would be barely visible, even for this small example. This large outer approximation comes from the lost dependencies between all involved variables.

4.3 Subgraph Verification

For a graph neural network with node-level output, we are not always required to propagate the entire graph through all layers of the network. Given a node of interest and a network with κ' message passing steps, we are only required to verify the subgraph within the $(\kappa'+1)$-hop neighborhood as all other nodes do not influence the considered node (Zügner

and Günnemann, 2019). We require $(\kappa' + 1)$ hops due to the normalization through the degree matrix in the message passing (Def. 4). The $(\kappa' + 1)$-hop neighborhood can easily be found using a breadth-first search on the given graph with the considered node as the root node. The graph and the respective feature matrix can be reduced as follows:

Corollary 17 (Subgraph Selection) *Given an input $H_{k-1} \in \mathbb{R}^{|\mathcal{N}| \times c_{k-1}}$ to a layer k, the message passing $P = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} \in \mathbb{R}^{|\mathcal{N}| \times |\mathcal{N}|}$ of a graph $\boldsymbol{\mathcal{G}}$, and the node indices $\mathcal{K}$ of a subgraph $\boldsymbol{\mathcal{G}}'$, we can construct a projection matrix $M = I_{|\mathcal{N}|(\mathcal{K},\cdot)}$ such that*

$$H'_{k-1} = MH_{k-1}, \qquad P' = MPM^\top,$$

contain the input and the message passing corresponding to the subgraph.

Proof The statement follows directly from the construction of the projection matrix M, where nodes that are not in $\boldsymbol{\mathcal{G}}'$ are removed. ■

After each graph convolutional layer (Def. 4), we can further reduce the graph as the number of remaining message-passing steps decreases. This can be achieved by implicitly adding projection layers computing Cor. 17 after each graph convolutional layer. After the last graph convolutional layer, we can remove all nodes except for the considered node, as no information is exchanged between nodes from that point onward. The selection of the subgraph only requires left and right matrix multiplications, thus, Cor. 17 can also be computed if the input $\mathcal{H}_{k-1} \subset \mathbb{R}^{|\mathcal{N}| \times c_{k-1}}$ or the message passing $\mathcal{P} \subset \mathbb{R}^{|\mathcal{N}| \times |\mathcal{N}|}$ are uncertain and represented by a matrix polynomial zonotope using (6).

5 Experimental Results

Our approach is implemented in the MATLAB toolbox CORA (Althoff, 2015), where we extend the existing approach of verifying neural networks using polynomial zonotopes (Kochdumper et al., 2023; Ladner and Althoff, 2023). Our implementation will be made publicly available with the next release of CORA. All computations were performed on an Intel Core Gen. 11 i7-11800H CPU @2.30GHz with 64GB memory.

We demonstrate our approach on three benchmark graph datasets: The first two, *Enzymes* and *Proteins*, represent protein structures tailored for the task of protein function classification (Borgwardt et al., 2005; Schomburg et al., 2004). The third dataset, *Cora*[2], represents a citation network with several classes of publications (Yang et al., 2016; McCallum et al., 2000). The main properties of each dataset are summarized in Tab. 1. All graph neural networks considered here are as described in Alg. 1, where we have three message-passing steps ($\kappa' = 3$) and tanh activation unless stated otherwise. The number of input and output neurons depends on the number of node features and classes of the dataset (Tab. 1), respectively, and the networks have 64 neurons per node in hidden layers.

To evaluate our approach on the datasets, we perturb the node features and graph structure as follows: We normalize all node features and perturb them using the same

2. The identical names of the toolbox CORA and the dataset Cora are coincidental, with no relation between the two.

Table 1: Properties of the benchmark datasets.

Name	Classification	#Graphs	#Nodes min/max	#Edges min/max	#Node features c_0	#Classes
Enzymes	graph-level	600	11/66	34/186	21	6
Proteins	graph-level	1,113	4/238	10/869	4	2
Cora	node-level	1	2,708	10,556	1,433	7

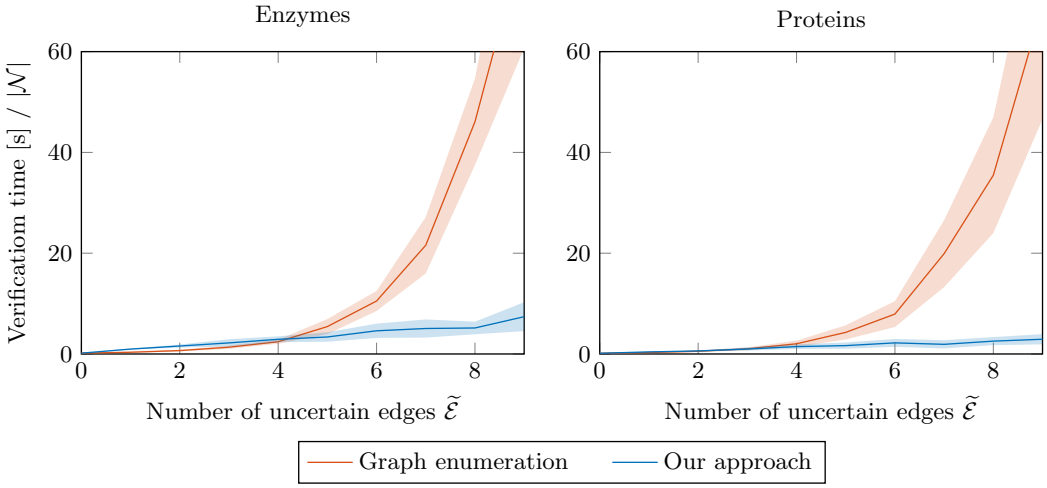

Figure 6: Time comparison of our approach with computing all possible graphs individually, where we normalized the verification time by the number of nodes $|\mathcal{N}|$ of the verified graphs.

perturbation radius $\delta \in \mathbb{R}_+$ on all features. Given a flattened input $\vec{X} \in \mathbb{R}^{|\mathcal{N}| \cdot c_0}$, our input set then becomes

$$\vec{\mathcal{X}} = \left\langle \vec{X}, \delta I_{|\mathcal{N}| \cdot c_0}, [\,], I_{|\mathcal{N}| \cdot c_0} \right\rangle_{PZ} \subset \mathbb{R}^{|\mathcal{N}| \cdot c_0}, \tag{13}$$

which we can reshape to a matrix polynomial zonotope $\mathcal{X} \subset \mathbb{R}^{|\mathcal{N}| \times c_0}$. For the perturbation of the graph structure, please recall that the uncertain graph $\boldsymbol{\mathcal{G}} = (\mathcal{N}, \mathcal{E})$ has nodes $\mathcal{N} \subset \mathbb{N}$ and edges $\mathcal{E} = \mathcal{E}^* \cup \widetilde{\mathcal{E}} \subseteq \mathcal{N} \times \mathcal{N}$ consisting of fixed edges $\mathcal{E}^*$ and uncertain edges $\widetilde{\mathcal{E}}$. The uncertain edges $\widetilde{\mathcal{E}}$ can be seen as a budget an attacker has to perturb the graph structure for the graph neural network to misclassify the input (Günnemann, 2022), and we verify that no possible configuration results in a misclassification (Sec. 2.5) using our approach (Thm. 16). The partitioning of the edges depends on the experiment: To preserve the structure of the input graphs, the set of fixed edges $\mathcal{E}^*$ always contains a spanning tree of the graph, and we make the presence of some remaining edges unknown and, thus, part of the uncertain edges $\widetilde{\mathcal{E}}$ depending on the experiment. The spanning tree is constructed using a breadth-first search, with the root node being the one with the highest degree (e.g., (1) in Fig. 1). We repeat each experiment 50 times with different graphs sampled from the respective dataset.

In our first experiment, we evaluate the polynomial time complexity (Thm. 16) on graphs with uncertain node features and uncertain graph structure. For this experiment, we iteratively increase the number of uncertain edges $\widetilde{\mathcal{E}}$, and compare it to enumerating all

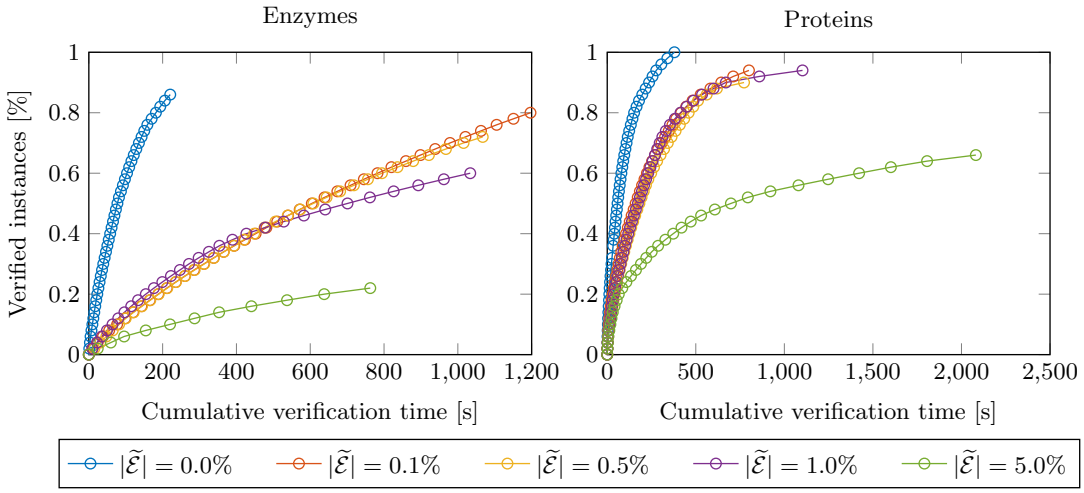

Figure 7: Verified instances of the Enzymes dataset and the Proteins dataset, where the number of uncertain edges $|\widetilde{\mathcal{E}}|$ is relative to the total number of edges $|\mathcal{E}|$ in the graph.

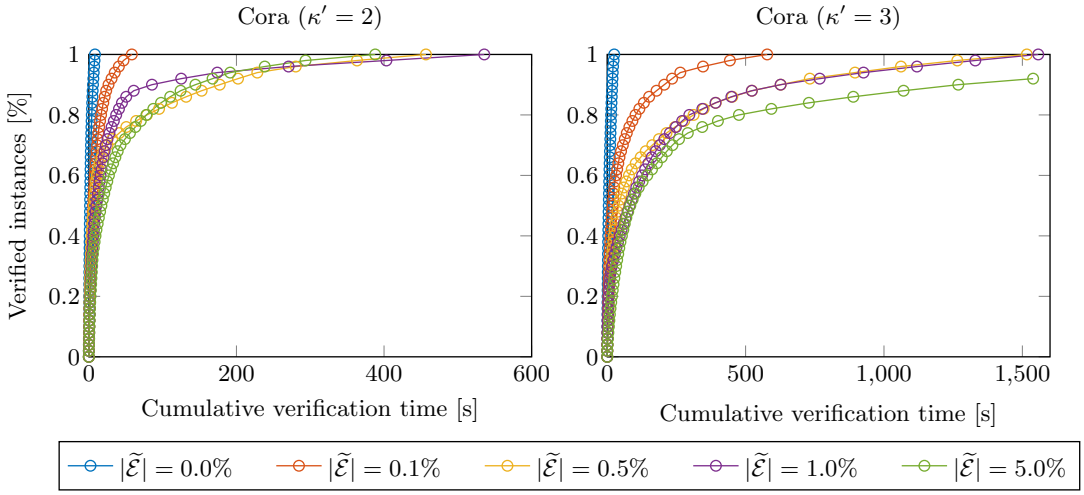

Figure 8: Verified instances of the Cora dataset with different numbers of message-passing steps, where the number of uncertain edges $|\widetilde{\mathcal{E}}|$ is relative to the total number of edges $|\mathcal{E}|$ in the graph.

possible graphs based on the uncertain edges and verifying them individually. As illustrated in Fig. 6, the verification time using enumeration quickly explodes due to its exponential time complexity, whereas the verification time of our approach remains low. We repeated this experiment only 20 times due to this reason.

In our second experiment, we examine the number of graphs verified by our approach (Fig. 7). The graphs are sorted by their size in ascending order, and we state the number of uncertain edges $\widetilde{\mathcal{E}}$ relative to the total number of edges of a graph for better comparability across differently sized graphs. We use a rather small perturbation radius $\delta = 0.001$ on the

Enzymes and Proteins dataset as we have found that the graph neural networks are not robust for larger radii, and counterexamples can easily be found. While we were able to verify almost all instances taken from the Proteins dataset, the verification rate drops on the Enzymes dataset. We think this is due to the smaller graphs in the Enzymes dataset, which appear to be less robust to graph structure perturbations using our networks, and smaller graphs have nodes with smaller degrees, which can result in a larger approximation error during the enclosure of the inverse square root function (Lemma 13) using linear polynomials (Fig. 4).

In our third experiment, we demonstrate the scalability of our approach by applying it on the Cora dataset. For this dataset, we do not use a perturbation radius ($\delta = 0$) as the input data is binary, and thus perturbations do not have an intuitive justification. As this dataset has a node-level output, we can also dynamically remove nodes that do not influence a considered node throughout the verification process (Sec. 4.3). However, we want to stress that, on average, about half of the nodes have to be considered initially, as the graph is highly connected. The verification results for two graph neural networks with different numbers of message-passing steps ($\kappa' = 2$ and $\kappa' = 3$) are shown in Fig. 8. We obtain high verification rates despite the large size of the graph of the Cora dataset (Tab. 1). Please note that for a fixed number of perturbed edges, the verification time varies significantly despite always verifying a node on the same graph. This is primarily due to the dynamic subgraph extraction being able to remove many nodes and, thus, significantly speeding up computation time.

6 Conclusion

In this work, we present the first formal verification approach for graph convolutional networks, where both the node features and the graph structure can be uncertain. The considered network architecture is a) generic, b) may have arbitrary element-wise activation functions, and, c) for the first time, can be verified over multiple message-passing steps. This is realized by generalizing existing verification approaches using polynomial zonotopes to graph neural networks. The use of (matrix) polynomial zonotopes allows us to preserve the non-convex dependencies of the involved variables and efficiently compute all underlying operations, resulting in an overall polynomial time complexity in the number of uncertain edges and uncertain input features. We demonstrate our approach using illustrative examples and an experimental evaluation on three popular datasets: The evaluation shows that our approach quickly outperformed verifying all individual graphs and produced tight output sets, thus making it possible to verify large graph neural networks.

Acknowledgments and Disclosure of Funding

This work was partially supported by the project FAI (No. 286525601) and the project SAFARI (No. 458030766), both funded by the German Research Foundation (DFG). We also want to thank our colleagues Florian Finkeldei, Lukas Koller, and Mark Wetzlinger for their revisions of the manuscript.

References

Matthias Althoff. An introduction to CORA 2015. In *Proceedings of the Workshop on Applied Verification for Continuous and Hybrid Systems*, pages 120–151, 2015.

Stanley Bak. nnenum: Verification of relu neural networks with optimized abstraction refinement. In *NASA Formal Methods Symposium*, pages 19–36, 2021.

Christopher M. Bishop and Nasser M. Nasrabadi. *Pattern recognition and machine learning*, volume 4. Springer, 2006.

Sergiy Bogomolov, Marcelo Forets, Goran Frehse, Kostiantyn Potomkin, and Christian Schilling. JuliaReach: A toolbox for set-based reachability. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, pages 39–44, 2019.

Aleksandar Bojchevski and Stephan Günnemann. Certifiable robustness to graph perturbations. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Aleksandar Bojchevski, Johannes Gasteiger, and Stephan Günnemann. Efficient robustness certificates for discrete data: Sparsity-aware randomized smoothing for graphs, images and more. In *International Conference on Machine Learning*, volume 119, pages 1003–1013, 2020.

Karsten M. Borgwardt, Cheng S. Ong, Stefan Schönauer, S. V. N. Vishwanathan, Alex J. Smola, and Hans-Peter Kriegel. Protein function prediction via graph kernels. In *Bioinformatics*, volume 21, pages i47–i56, 2005.

Elena Botoeva, Panagiotis Kouvaros, Jan Kronqvist, Alessio Lomuscio, and Ruth Misener. Efficient verification of relu-based neural networks via dependency analysis. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 3291–3299, 2020.

Christopher Brix and Thomas Noll. Debona: Decoupled boundary network analysis for tighter bounds and faster adversarial robustness proofs. In *arXiv preprint arXiv:2006.09040*, 2020.

Christopher Brix, Stanley Bak, Changliu Liu, and Taylor T. Johnson. The fourth international verification of neural networks competition (VNN-COMP 2023): Summary and results. In *arXiv preprint arXiv:2312.16760*, 2023.

Michael M. Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: Going beyond Euclidean data. In *IEEE Signal Processing Magazine*, volume 34, pages 18–42, 2017.

Rudy Bunel, Ilker Turkaslan, Philip Torr, Mudigonda P. Kumar, Jingyue Lu, and Pushmeet Kohli. Branch and bound for piecewise linear neural network verification. In *Journal of Machine Learning Research*, volume 21, pages 1–39, 2020.

Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, and Xu Sun. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 3438–3445, 2020.

Sikai Chen, Jiqian Dong, Paul Ha, Yujie Li, and Samuel Labi. Graph neural network and reinforcement learning for multi-agent cooperative control of connected autonomous vehicles. volume 36, pages 838–857, 2021.

Hanjun Dai, Hui Li, Tian Tian, Xin Huang, Lin Wang, Jun Zhu, and Le Song. Adversarial attack on graph structured data. In *International Conference on Machine Learning*, volume 80, pages 1115–1124, 2018.

Souradeep Dutta, Susmit Jha, Sriram Sankaranarayanan, and Ashish Tiwari. Output range analysis for deep feedforward neural networks. In *NASA Formal Methods Symposium*, pages 121–138, 2018.

Claudio Ferrari, Mark N. Mueller, Nikola Jovanović, and Martin Vechev. Complete verification via multi-neuron relaxation guided branch-and-bound. In *International Conference on Learning Representations*, 2022.

Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin Vechev. AI2: Safety and robustness certification of neural networks with abstract interpretation. In *IEEE Symposium on Security and Privacy*, pages 3–18, 2018.

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

Stephan Günnemann. Graph neural networks: Adversarial robustness. In *Graph Neural Networks: Foundations, Frontiers, and Applications*, pages 149–176, 2022.

Patrick Henriksen and Alessio Lomuscio. Efficient neural network verification via adaptive refinement and adversarial search. In *European Conference on Artificial Intelligence*, volume 325, pages 2513–2520. 2020.

Chao Huang, Jiameng Fan, Xin Chen, Wenchao Li, and Qi Zhu. POLAR: A polynomial arithmetic framework for verifying neural-betwork controlled systems. In *Automated Technology for Verification and Analysis*, pages 414–430, 2022.

Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. Safety verification of deep neural networks. In *International Conference on Computer Aided Verification*, pages 3–29, 2017.

Radoslav Ivanov, Taylor Carpenter, James Weimer, Rajeev Alur, George Pappas, and Insup Lee. Verisig 2.0: Verification of neural network controllers using Taylor model preconditioning. In *International Conference on Computer Aided Verification*, pages 249–262, 2021.

Luc Jaulin, Michel Kieffer, Olivier Didrit, and Éric Walter. *Interval analysis*. 2001.

Jinyuan Jia, Binghui Wang, Xiaoyu Cao, and Neil Z. Gong. Certified robustness of community detection against adversarial structural perturbation via randomized smoothing. In *Proceedings of The Web Conference*, pages 2718–2724, 2020.

Hongwei Jin, Zhan Shi, Venkata J. S. A. Peruri, and Xinhua Zhang. Certified robustness of graph convolution networks for graph classification under topological attacks. In *Advances in Neural Information Processing Systems*, volume 33, pages 8463–8474, 2020a.

Wei Jin, Yao Ma, Xiaorui Liu, Xianfeng Tang, Suhang Wang, and Jiliang Tang. Graph structure learning for robust graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 66–74, 2020b.

Guy Katz, Clark Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. In *International Conference on Computer Aided Verification*, pages 97–117, 2017.

Guy Katz, Derek A. Huang, Duligur Ibeling, Kyle Julian, Christopher Lazarus, Rachel Lim, Parth Shah, Shantanu Thakoor, Haoze Wu, Aleksandar Zeljić, David L. Dill, Mykel J. Kochenderfer, and Clark Barret. The Marabou framework for verification and analysis of deep neural networks. In *International Conference on Computer Aided Verification*, pages 443–452, 2019.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017.

Niklas Kochdumper. *Extensions of polynomial zonotopes and their application to verification of cyber-physical systems*. PhD thesis, Technische Universität München, 2022.

Niklas Kochdumper and Matthias Althoff. Sparse polynomial zonotopes: A novel set representation for reachability analysis. In *IEEE Transactions on Automatic Control*, volume 66, pages 4043–4058, 2020.

Niklas Kochdumper, Christian Schilling, Matthias Althoff, and Stanley Bak. Open- and closed-loop neural network verification using polynomial zonotopes. In *NASA Formal Methods Symposium*, pages 16–36, 2023.

Matthias König, Annelot W. Bosman, Holger H. Hoos, and Jan N. van Rijn. Critically assessing the state of the art in neural network verification. In *Journal of Machine Learning Research*, volume 25, pages 1–53, 2024.

Tobias Ladner and Matthias Althoff. Automatic abstraction refinement in neural network verification using sensitivity analysis. In *Proceedings of the 26th ACM International Conference on Hybrid Systems: Computation and Control*, pages 1–13, 2023.

Tobias Ladner and Matthias Althoff. Exponent relaxation of polynomial zonotopes and its applications in formal neural network verification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 21304–21311, 2024.

Diego Manzanas Lopez, Sung Woo Choi, Hoang-Dung Tran, and Taylor T. Johnson. NNV 2.0: The neural network verification tool. In *International Conference on Computer Aided Verification*, pages 397–412, 2023.

Andrew K. McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. Automating the construction of internet portals with machine learning. In *Information Retrieval*, volume 3, pages 127–163, 2000.

Mark N. Müller, Gleb Makarchuk, Gagandeep Singh, Markus Püschel, and Martin Vechev. PRIMA: General and precise neural network certification via scalable convex hull approximations. In *Proceedings of the ACM on Programming Languages*, volume 6, pages 1–33, 2022.

Luca Pulina and Armando Tacchella. An abstraction-refinement approach to verification of artificial neural networks. In *International Conference on Computer Aided Verification*, pages 243–257, 2010.

Marco Sälzer and Martin Lange. Fundamental limits in formal verification of message-passing neural networks. In *International Conference on Learning Representations*, 2023.

Ida Schomburg, Antje Chang, Christian Ebeling, Marion Gremse, Christian Heldt, Gregor Huhn, and Dietmar Schomburg. BRENDA, the enzyme database: Updates and major new developments. In *Nucleic Acids Research*, volume 32D, pages 431–433, 2004.

Zhouxing Shi, Qirui Jin, Jeremy Z. Kolter, Suman Jana, Cho-Jui Hsieh, and Huan Zhang. Formal verification for neural networks with general nonlinearities via branch-and-bound. In *2nd Workshop on Formal Verification of Machine Learning*, 2023.

Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin Vechev. Fast and effective robustness certification. In *Advances in Neural Information Processing Systems*, volume 31, 2018a.

Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. Boosting robustness certification of neural networks. In *International Conference on Learning Representations*, 2018b.

Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. An abstract domain for certifying neural networks. In *Proceedings of the ACM on Programming Languages*, volume 3, pages 1–30, 2019.

Vincent Tjeng, Kai Y. Xiao, and Russ Tedrake. Evaluating robustness of neural networks with mixed integer programming. In *International Conference on Learning Representations*, 2019.

Shiqi Wang, Huan Zhang, Kaidi Xu, Xue Lin, Suman Jana, Cho-Jui Hsieh, and Jeremy Z. Kolter. Beta-CROWN: Efficient bound propagation with per-neuron split constraints for complete and incomplete neural network verification. In *Advances in Neural Information Processing Systems*, volume 34, 2021.

Haoze Wu, Clark Barrett, Mahmood Sharif, Nina Narodytska, and Gagandeep Singh. Scalable verification of GNN-based job schedulers. In *Proceedings of the ACM on Programming Languages*, volume 6, pages 1036–1065, 2022.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. In *IEEE Transactions on Neural Networks and Learning Systems*, volume 32, pages 4–24, 2020.

Weiming Xiang, Hoang-Dung Tran, and Taylor T. Johnson. Output reachable set estimation and verification for multilayer neural networks. In *IEEE Transactions on Neural Networks and Learning Systems*, volume 29, pages 5777–5783, 2018.

Zhilin Yang, William Cohen, and Ruslan Salakhudinov. Revisiting semi-supervised learning with graph embeddings. In *International Conference on Machine Learning*, volume 48, pages 40–48, 2016.

Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 974–983, 2018.

Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. Efficient neural network robustness certification with general activation functions. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

Xiao-Meng Zhang, Li Liang, and Lin Liu. Graph neural networks and their current applications in bioinformatics. In *Frontiers in Genetics*, volume 12, 2021.

Daniel Zügner and Stephan Günnemann. Certifiable robustness and robust training for graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 246–256, 2019.