Scalable Polyhedral Verification of Recurrent Neural Networks

Wonryong Ryou[1($\boxtimes$)], Jiayu Chen[1], Mislav Balunović[1],
Gagandeep Singh[2], Andrei Dan[3], and Martin Vechev[1]

[1] Department of Computer Science, ETH Zürich, Zurich, Switzerland
wryou@ethz.ch
[2] VMWare Research and Department of Computer Science, UIUC, Champaign, USA
[3] Hitachi Power Grids Research, Zurich, Switzerland

Abstract. We present a scalable and precise verifier for recurrent neural networks, called PROVER based on two novel ideas: (i) a method to compute a set of polyhedral abstractions for the non-convex and non-linear recurrent update functions by combining sampling, optimization, and Fermat's theorem, and (ii) a gradient descent based algorithm for abstraction refinement guided by the certification problem that combines multiple abstractions for each neuron. Using PROVER, we present the first study of certifying a non-trivial use case of recurrent neural networks, namely speech classification. To achieve this, we additionally develop custom abstractions for the non-linear speech preprocessing pipeline. Our evaluation shows that PROVER successfully verifies several challenging recurrent models in computer vision, speech, and motion sensor data classification beyond the reach of prior work.

Keywords: Robustness verification · Polyhedral abstraction · Recurrent neural networks · Long short-term memory · Abstraction refinement · Speech classifier verification

1 Introduction

Recurrent neural networks (RNNs) are widely used to model long-term dependencies in lengthy sequential signals [11,27,43]. Prior work has demonstrated the susceptibility of RNNs to adversarial perturbations of its inputs [28], exposing security vulnerabilities of state-of-the-art RNNs when used in domains such as speech recognition [8,22], malware detection [16], and others. Thus, verifying the robustness of recurrent architectures is critical for their safe deployment. While there has been considerable interest in certifying the robustness of feedforward image classifiers [4,12,13,23,32,37,39,47], less attention has been given to recurrent architectures. As a result, current certification solutions do not scale beyond simple models and datasets, which limits their practical applicability. Further, there has been no work on verifying real-world use cases of RNNs. In this paper, we address both of these challenges and present the first precise and scalable verifier for RNNs based
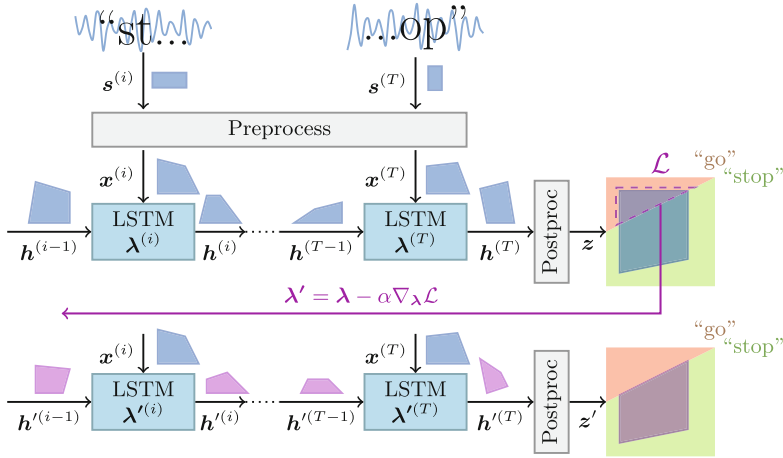
Fig. 1. Certification of recurrent architectures using PROVER: utterance "stop" with perturbations is correctly classified. Possible perturbations are captured and propagated through the system, then refined backward for improved precision. (Color figure online)

on abstract interpretation [10], which enables us to certify robustness of realistic speech recognition systems.

We illustrate the problem setting and overall flow in Fig. 1. Here, a speech recognition model based on the Long Short-Term Memory (LSTM) architecture [15] receives a signal encoding the utterance of "stop" by a human. As such models are usually employed in noisy environments, they must robustly classify variations (e.g., voice changes) to the utterance "stop". However, recent work [8] has shown the model may be fooled into classifying the utterance as "go". It is important to prove such mis-classifications are not possible, thus avoiding a potential exploitation by an adversary, for instance in automated traffic control settings (which can lead to accidents). Our goal is to design a verifier that can formally establish the robustness of such models against noise-induced perturbations. We focus on LSTMs, as they are the most widely used form of RNNs, but our methodology can be easily extended to other architectures (e.g., Gated Recurrent Unit (GRU) [9]). Figure 1 shows how our proposed verifier, called PROVER, (Polyhedral Robustness Verifier of RNNs) automatically verifies the robustness of the model. Here, the labeled rectangles represent operations in the network. The "Preprocess" box captures domain-specific pre-processing operations (typically present when using RNNs, e.g., speech processing). In our method, we first compute a polyhedral abstraction capturing all speech signals given as input to the model under the given perturbation budget. At each timestep i, the pre-processing operation receives a polyhedron $s^{(i)}$ and produces an output polyhedron $x^{(i)}$. This shape is then propagated symbolically through the LSTM and the post-processing stage, resulting in a polyhedral output shape, denoted as z (blue shape in Fig. 1).

Key Challenge: Polyhedral Abstractions for LSTMs. The main challenge in certifying LSTMs is the design of precise and scalable polyhedral abstract

transformers for the non-linear operations employed in LSTMs: given a polyhedral shape capturing hidden states $\boldsymbol{h}^{(i-1)}$, to produce the shape capturing the next set of hidden states $\boldsymbol{h}^{(i)}$. A recent method [21] computes this based on gradient-based optimization but suffers from two main limitations. First, the optimization procedure is computationally expensive and does not scale to realistic use cases. Second, the method lacks convergence and optimality guarantees. To address these issues, we introduce a novel technique based on a combination of sampling, linear programming, and Fermat's theorem [1], which significantly improves the precision and scalability compared to prior work [21], while offering asymptotic guarantees of convergence towards the optimal solution.

Refinement via Optimization. To certify robustness, we must verify that each concrete point in the output shape $\boldsymbol{z}$ corresponds to the correct label "stop". However, $\boldsymbol{z}$ can contain, due to over-approximation, spurious incorrect concrete points (it intersects the red region representing incorrect outputs). To address this issue, we form a loss based on the output shape, backpropagate the gradient of this loss through the timesteps and adjust the polyhedral abstractions in each LSTM unit to decrease the loss. The goal is to refine the abstraction, guided by the certification task. We illustrate this process in Fig. 1 using the purple backward arrow with the refined polyhedral abstraction shown in purple. Using the refined abstraction, the new output shape $\boldsymbol{z}'$ (purple polygon) lies completely inside the green region of the output space, meaning it provably contains only correct output vectors (corresponding to "stop"), and hence certification succeeds. Overall, our method significantly increases the precision of end-to-end RNN certification without introducing high runtime costs.

Key Contributions. Our main contributions are:

- A new and efficient method to certify the robustness of RNNs to adversarial perturbations. Our method relies on novel polyhedral abstractions for handling non-linear operations in these architectures.
- A novel method that automatically refines the abstraction for each input example being certified guided by the certification task.
- An implementation of the method in a system called PROVER and evaluation on several benchmarks and datasets. Our results show that PROVER is precise and scales to larger models than prior work. PROVER is also the first verifier able to certify realistic RNN-based speech classifiers. The code is available in https://github.com/eth-sri/prover.

2 Related Work

While the first adversarial examples for neural networks were found in computer vision [6,41], recent work also showed the vulnerability of RNNs [28]. Modern speech recognition systems, based on RNNs, were shown susceptible to small noise crafted by an adversary using white-box attacks [7,8], achieving a 100% success rate against DeepSpeech [14], a state-of-the-art speech-to-text engine. These were later followed by attacks based on universal perturbation [26] and temporal dependency [46]. Recent work [22,31] demonstrates that adversarial

examples for audio classifiers are realizable in the real-world. While giving an empirical estimate of the vulnerability of RNNs, these works do not provide any formal guarantees, which is the goal of our work.

There have also been recent works on the verification of RNNs. [2] propose the certification of RNNs based on mixed-integer linear programming, which only works for ReLU-based networks and does not consider LSTMs, which use sigmoid and tanh activations. [45] propose an input discretization method to certify video models that are a combination of CNNs and RNNs. However, discretization does not scale to the perturbations we consider in our work. [18] propose to verify RNNs by automatically inferring temporal homogeneous invariants using binary search. However, their approach is limited to vanilla RNNs and does not apply to the more commonly used LSTM networks considered in this work. [19] propose the statistical variant of Angulin's algorithm [3] for probabilistic verification and counterexample generation for RNNs, however they cannot provide deterministic guarantees as our work. The work most related to ours is POPQORN [21] which uses expensive gradient-based optimizations for every operation in the network. We experimentally show that it does not scale to practical applications such as speech classification.

3   Background

We first define the threat model and then present all operations that are part of the verification procedure, including speech preprocessing and LSTM updates.

3.1   Threat Model

We use a threat model based on the L_∞-norm, where an attacker can change each element of a correctly classified input vector s by an amount $\leq \epsilon \in \mathbb{R}$ [8]. Therefore, our input region can be represented as a conjunction of intervals $[s_i - \epsilon, s_i + \epsilon]$, where s_i is the i-th element of s. The measure of signal distortion in this setting are decibels (dB) defined as:

$$dB(s) = \max_i 20 \cdot \log_{10}(|s_i|); \; dB_s(\delta) = dB(\delta) - dB(s)$$

The quieter the perturbation is, the smaller $dB_s(\delta)$ is. We fix the $dB_s(\delta) =: \epsilon$ as *dB perturbation* and focus on verifying that the model classifies correctly all signals s' possible under our threat model.

3.2   Long Short-Term Memory (LSTM)

LSTM architectures [15] are popular for handling sequential data as they can utilize long-term dependencies. These dependencies are passed through time using two state vectors for the timestep t: cell state $c^{(t)}$ and hidden state $h^{(t)}$. These state vectors are updated using the following formulas:
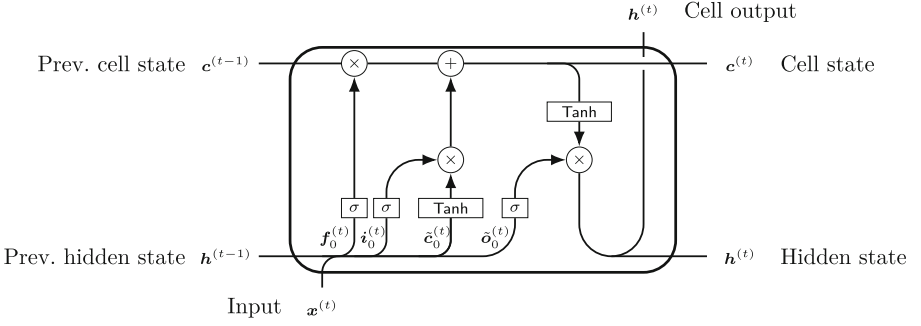
Fig. 2. LSTM cell: $\boldsymbol{f}_0^{(t)}, \boldsymbol{i}_0^{(t)}, \boldsymbol{o}_0^{(t)}$, and $\tilde{\boldsymbol{c}}_0^{(t)}$ represent the pre-activated gates.

$$\boldsymbol{f}_0^{(t)} = [\boldsymbol{x}^{(t)}, \boldsymbol{h}^{(t-1)}]\boldsymbol{W}_f + \boldsymbol{b}_f \qquad \boldsymbol{i}_0^{(t)} = [\boldsymbol{x}^{(t)}, \boldsymbol{h}^{(t-1)}]\boldsymbol{W}_i + \boldsymbol{b}_i$$

$$\boldsymbol{o}_0^{(t)} = [\boldsymbol{x}^{(t)}, \boldsymbol{h}^{(t-1)}]\boldsymbol{W}_o + \boldsymbol{b}_o \qquad \tilde{\boldsymbol{c}}_0^{(t)} = [\boldsymbol{x}^{(t)}, \boldsymbol{h}^{(t-1)}]\boldsymbol{W}_{\tilde{c}} + \boldsymbol{b}_{\tilde{c}}$$

$$\boldsymbol{c}^{(t)} = \sigma(\boldsymbol{f}_0^{(t)}) \odot \boldsymbol{c}^{(t-1)} + \sigma(\boldsymbol{i}_0^{(t)}) \odot \tanh(\tilde{\boldsymbol{c}}_0^{(t)}) \qquad \boldsymbol{h}^{(t)} = \sigma(\boldsymbol{o}_0^{(t)}) \odot \tanh(\boldsymbol{c}^{(t)})$$

where $[\cdot, \cdot]$ is the horizontal concatenation of two row vectors, $\boldsymbol{W}.$ and $\boldsymbol{b}.$ are the kernel and bias of the cell, respectively, and σ is the sigmoid function. At timestep t, vectors $\boldsymbol{f}_0^{(t)}, \boldsymbol{i}_0^{(t)}, \boldsymbol{o}_0^{(t)}, \tilde{\boldsymbol{c}}_0^{(t)}$ represent pre-activations of the forget gate, input gate, output gate and the candidate gate, respectively. We show an illustration of an LSTM cell in Fig. 2. We treat σ and tanh as forms of activation functions, which is why we define the LSTM using pre-activations.

Intuitively, the *input gate* transforms the input vector, the *forget gate* filters the information from the previous cell state, the *candidate gate* prepares the candidate cell state, and the *output gate* transforms the current hidden state. All of these gates receive as input the hidden state $\boldsymbol{h}^{(t-1)}$ of the previous cell and the input $\boldsymbol{x}^{(t)}$ representing the current frame. This recurrent architecture allows inputs with arbitrary length, enabling LSTMs to handle temporal data, e.g., speech processing.

3.3   Speech Preprocessing

Though there have been various works that operate directly on the raw signal [29,36], speech signals are commonly preprocessed using the *filterbank* or *log Mel-filterbank energy* methods. The result is a vector of coefficients whose elements contain log-scaled values of filtered spectra, one for every Mel-frequency. This method models the non-linear human acoustic perception as power spectrum filters based on Mel-frequencies. The input signal is split into several (possibly overlapping) frames for granular analysis, and the following steps are applied:

1. *Pre-emphasizing and windowing* are preprocessing stages on the raw signals. Speech signals tend to have larger and smoother low-frequency samples and smaller and fluctuating high-frequency samples. *Pre-emphasizing* is a process

of subtracting the adjacent sampled values multiplied by a scalar parameter $(\boldsymbol{s}_j^{(i)} - \alpha \boldsymbol{s}_{j-1}^{(i)}$, commonly $\alpha = 0.97$). This alleviates the unbalanced distribution of signal strength along with the frequency. *Windowing* involves multiplication of each sampled value and 'windows' according to their indices. The window here refers to a Hamming window, which is a bell-like curve with peak in the middle of the frame and drops at the side. It reduces the border effects on each frame by suppressing the values near the border with smaller values.

2. *Power spectrum of Fast Fourier transform (FFT)* performs the discrete Fourier transform (DFT) and obtains the squared norm of each element to obtain intensities in the frequency domain. FFT consists of matrix multiplications with complex entries. We modify it to use only real numbers by: (i) separating real and imaginary parts of the matrix and constructing two separate matrices, (ii) multiplying each matrix with the signal, (iii) squaring the entries, and (iv) adding the resulting matrices entry-wise.

3. *Mel-filter bank log energy:* The Mel-frequency filters are triangular, each emphasizing the power of the selected frequency and suppressing the adjacent ones. In our case, we (i) apply the Mel-filterbank to the power spectrum and (ii) take the log of the entries to adjust the level.

Following [35], each step can be represented as a distinct matrix operation. It allows us to decompose and rearrange the steps into slightly different stages:

1. *Pre-square stage:* $\boldsymbol{S} \rightarrow \boldsymbol{Y} = \boldsymbol{S}\boldsymbol{M}_1$. This stage contains pre-emphasizing, windowing (step 1), and FFT (until step 2-(ii)). All operations are representable as matrix multiplications, so we pre-calculate the product matrix.

2. *Square stage:* $\boldsymbol{Y} \rightarrow \boldsymbol{\theta} = \boldsymbol{Y} \odot \boldsymbol{Y}$. This is step 2-(iii). Entry-wise square operations cannot be combined with other matrix multiplications.

3. *Pre-log stage:* $\boldsymbol{\theta} \rightarrow \tilde{\boldsymbol{X}} = \boldsymbol{\theta}\boldsymbol{M}_2$. From step 2-(iv) through step 3-(i). We combine the operations into a single matrix.

4. *Log stage:* $\tilde{\boldsymbol{X}} \rightarrow \boldsymbol{X} = \log \tilde{\boldsymbol{X}}$. Applying entry-wise logarithm (step 3-(ii)).

We use the resulting $\boldsymbol{X} = [\boldsymbol{x}^{(1)} \cdots \boldsymbol{x}^{(T)}]^{\mathsf{T}}$ as the input to the neural network.

3.4 Verification Using DeepPoly Abstract Domain

DeepPoly [39] is a sub-polyhedral abstract domain that associates a lower and an upper polyhedral bound and interval bounds per neuron. It is faster than Polyhedra [40] and more precise than other weakly relational domains such as Octagons [25], Zones [24], and Zonotopes [38] when analyzing neural networks. Previously, it has been suceesfully applied for verifying feedforward networks in [4,39]. Formally, let $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$ be an ordered set of neurons such that the neurons in layer l appear before the neurons of layer $l' > l$. DeepPoly associates with each neuron x_j, both interval $l_j \leq x_j \leq u_j$ and polyhedral bounds $\sum_{i<j} a_i \cdot x_i + b \leq x_j \leq \sum_{i<j} a_i' \cdot x_i + b'$ where $l_j, u_j, a_i, a_i', b, b' \in \mathbb{R} \cup \{\infty\}$. DeepPoly is exact for affine transformations which are frequently applied both in the speech preprocessing pipeline and the LSTM unit. DeepPoly loses precision

for the non-linear operations in LSTMs. We note that computing polyhedral bounds on their output is more challenging than for feedforward networks.

The precision of the DeepPoly approximation for the non-linear operations depends on the tightness of the interval bounds of the neurons that are input to the non-linear operations. DeepPoly provides a scalable and precise method called *backsubstitution* for optimizing a linear expression within a region defined by the set of DeepPoly constraints. It does so by recursively substituting the bounding linear expressions of target neurons with the polyhedral bounds of previous layers' neurons until reaching the input neurons. It then uses the concrete bounds of the input neurons for computing the result. Backsubstitution is used for computing the interval bounds of neurons input to the non-linear operations as well as for bounding the difference between the neurons in the output layer needed to prove robustness. We refer the reader to [39] for details of the backsubstitution.

4   Overview of *PROVER*

This section illustrates the workings of PROVER on a small example. Our goal is to certify the robustness of a single LSTM cell on the input $x \in [-1.2, 1.2]$. For this example, we assume that there are two output classes and all intermediate LSTM gates $\{i, f, \tilde{c}, o\}$ share the same weights and biases:

$$\{i, f, \tilde{c}, o\} = \begin{bmatrix} 1 \\ 0.5 \end{bmatrix} x + \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \qquad c = \sigma(i) \odot \tanh(\tilde{c}), \qquad h = \sigma(o) \odot \tanh(c).$$

The correct output here is h_2 and to certify robustness we need to prove that $h_2 - h_1 > 0$ holds for all inputs x. In other words, $\min h_2 - h_1 > 0$.

Polyhedral Abstraction. We build our verifier based on the DeepPoly [39] abstraction since DeepPoly outperforms the interval analysis and other competitive domains, as Sect. 3.4 states.

Challenges in Computing Polyhedral Bounds for LSTMs. The composed binary non-linear operations applied in LSTMs such as $\sigma(x) \tanh(y)$ and $\sigma(x)y$ are significantly more complex to handle than the ReLU, Sigmoid, and Tanh activations originally handled by [39]. This is because the non-linear operations in LSTMs mentioned above involve transcendental functions yielding non-linear 3D curves that are neither convex nor concave. The optimal polyhedral bounds for these operations have no closed-form solution and cannot be calculated by simple geometry or algebra. Further, obtaining such bounds is computationally expensive [21]. For example, obtaining the lower linear plane for bounding $\sigma(x) \tanh(y)$ is equivalent to solving a Lagrangian with 6 variables - 3 linear coefficients, 2 interval-bounded coordinates and 1 Lagrange multiplier for the constraint. In contrast, the optimal polyhedral bounds for ReLU, Sigmoid, and Tanh have closed form solutions, easily visualized in 2D.

Precise Polyhedral Bounds via LP. To overcome these challenges, we propose a generic approach based on linear programming (LP) to compute precise

polyhedral bounds. We illustrate our approach for calculating a lower polyhedral bound of $h_2 = \sigma(o_2)\tanh(c_2)$. First, we calculate the concrete intervals for the two target variables via *backsubstitution* [39], briefly described in Sect. 3.4. In our case, the target variables are o_2 and c_2 and the backsubstitution yields $o_2 \in [0.4, 1.6]$ and $c_2 \in [-0.79, 0.62]$. Our abstraction can represent the affine transformations exactly. Therefore, we obtain the exact interval for $o_2 = 0.5 \cdot x + 1$ via the backsubstitution whereas the obtained interval for c_2 is an overapproximate one. Then, we uniformly sample a set of points $\{(x_1, y_1), ..., (x_n, y_n)\}$ from the input domain $[0.4, 1.6] \times [-0.79, 0.62]$. We solve the following optimization problem to calculate the lower polyhedral bound of h_2:

$$\min_{A_l, B_l, C_l \in \mathbb{R}} \sum_{i=1}^{n} \left(\sigma(x_i)\tanh(y_i) - (A_l \cdot x_i + B_l \cdot y_i + C_l) \right),$$

subject to the constraint that $A_l \cdot x_i + B_l \cdot y_i + C_l \leq \sigma(x_i)\tanh(y_i)$ for each i. This is a linear program over three variables (A_l, B_l, C_l) that can be solved efficiently in polynomial time. However, the obtained bound may not be sound as the sampled points do not fully cover the continuous input domain. To address this, we shift the plane downwards by an offset (decreasing C_l) equal to the maximum violation between $A_l \cdot x + B_l \cdot y + C_l$ and h_2 based on Fermat's theorem. After solving the linear program and the adjustment, we obtain $A_l = 0.04, B_l = 0.46, C_l = 0.01$ which results in the following lower polyhedral bound to h_2: $h_2 \geq LB_{h_2} = 0.04 \cdot o_2 + 0.46 \cdot c_2 + 0.01$. We compute the upper bound to $h_2 : h_2 \leq UB_{h_2}$ analogously. After computing a polyhedral abstraction of each neuron, we calculate the lower bound of $h_2 - h_1$ via backsubstitution as follows:

$$\begin{aligned}
\min h_2 - h_1 &\geq LB_{h_2} - UB_{h_1} \\
&\geq (0.04 \cdot o_2 + 0.46 \cdot c_2 + 0.01) - (-0.09 \cdot o_1 + 0.66 \cdot c_1 + 0.14) \\
&\geq 0.04 \cdot o_2 + 0.46 \cdot (0.07 \cdot i_2 + 0.27 \cdot g_2 + 0.09) \\
&\quad + 0.09 \cdot o_1 - 0.66 \cdot (-0.04 \cdot i_1 + 0.38 \cdot g_1 + 0.25) - 0.14 \\
&\geq 0.20 \cdot (0.5 \cdot x + 1) - 0.13 \cdot x - 0.10 \geq -0.03 \cdot x - 0.08 \geq -0.11.
\end{aligned}$$

The precision of the bounds generated by our LP-based method increases with the number of samples yielding optimal bounds (in the sense of small gap) asymptotically. For our example, the computed bounds are optimal.

While our optimal bounds significantly improve precision compared to intervals, they are not sufficient to certify robustness. Prior work for ReLU networks [5,12,23] showed that the greedy approach of always selecting the optimal bounds minimizing the gap can yield less precise results than an adaptive strategy which computes bounds guided by the certification problem. Based on this observation, we introduce a novel approach based on splitting and gradient descent that computes polyhedral abstractions for non-linearities employed in LSTMs informed by the certification problem and proves that $\min h_2 - h_1 > 0$ actually holds.

Abstraction Refinement via Splitting and Gradient Descent. While our method based on LP offers an efficient way to compute polyhedral abstraction of

activation functions, its main limitation is that the abstraction cannot be refined based on the certification goal. In this work, we introduce a novel method where we first compute mutiple sound bounds for the neuron using our LP method and then automatically obtain a combination of the computed bounds that improves the lower bound of our certification objective $h_2 - h_1$ for each input example. As before, we use the backsubstitution to obtain the interval bounds for the input variables. Since the output of our LP method is sensitive to the choice of the sampled points, we split the original input region $[l_x, u_x] \times [l_y, u_y]$ to sample more effectively from smaller sub-regions thereby reducing the chances of missing an outlier. We found that splitting along the two diagonals of $[l_x, u_x] \times [l_y, u_y]$ into four triangular zones, denoted as $\mathcal{T}_k$, $k \in \{1, 2, 3, 4\}$, performs the best in our evaluation. We use $\mathcal{T}_0$ to denote the original input region. Next, we calculate four additional planes, for both the upper and lower bounds, by sampling each subregion $\mathcal{T}_k$ and then applying our LP method as before. We refer to each plane as a *candidate bound*:

$$\min_{A_l, B_l, C_l \in \mathbb{R}} \sum_{i=1}^{n} (\sigma(x_i) \tanh(y_i) - (A_l \cdot x_i + B_l \cdot y_i + C_l))$$

$$\text{subject to } \bigwedge_{i=1}^{n} A_l \cdot x_i + B_l \cdot y_i + C_l \leq \sigma(x_i) \tanh(y_i) \text{ where } (x_i, y_i) \sim \mathcal{T}_k$$

Using our LP based method, we obtain the following corresponding candidate polyhedral abstraction for h_2, $LB_{h_2}^k$ for each $\mathcal{T}_k$ in our example:

$$h_2 \geq LB_{h_2}^0 = 0.04 \cdot o_1 + 0.46 \cdot c_1 + 0.01, h_2 \geq LB_{h_2}^1 = 0.04 \cdot o_1 + 0.46 \cdot c_1 + 0.01$$
$$h_2 \geq LB_{h_2}^2 = 0.13 \cdot o_1 + 0.63 \cdot c_1 - 0.17, h_2 \geq LB_{h_2}^3 = 0.04 \cdot o_1 + 0.46 \cdot c_1 + 0.01$$
$$h_2 \geq LB_{h_2}^4 = 0.13 \cdot o_1 + 0.63 \cdot c_1 - 0.17$$

Note that $LB_{h_2}^0$ denotes the polyhedral abstraction calculated for the whole region, and there might be duplicate LB's when the curve in the given subregion is concave. The final bound LB_{h_2} is a linear combination of $LB_{h_2}^k$:

$$LB_{h_2} = \sum_{k=0}^{4} \lambda_i \cdot LB_{h_2}^k, \qquad \sum_{k=0}^{4} \lambda_i = 1.$$

Our optimization algorithm, explained in Sect. 5.2, learns the values of λ_i via gradient descent that maximizes $\min h_2 - h_1$. For our example, we obtain $\lambda = (0.09, 0.13, 0.34, 0.09, 0.35)$ as the set of coefficients which results in a new lower bound of $h_2 \geq 0.10 \cdot o_2 + 0.58 \cdot c_2 - 0.11$ for the neuron h_2. We improve the bounds for other neurons in a similar fashion. Using the new bounds, we obtain $h_2 - h_1 \geq 0.01 > 0$ which enables us to correctly certify the predicate of interest. If the certification still fails, it is possible to further refine the abstraction by increasing the number of splits and repeating the procedure above.

Compared to [21], which uses a single bound, our method is more flexible and can tune $\boldsymbol{\lambda}$ parameters to find a combination of different bounds for each neuron

that yields the most precise certification result for each certification instance. Our method is also faster as it performs expensive gradient-based optimization for only the output layer whereas [21] performs this step for each neuron in the LSTM twice. [5,12,23] also suggest a similar idea of bounding ReLU's lower bound using gradient descent, but their approach is limited to unary functions with trivial candidates, not applicable to our setting which requires handling complex binary operations with non-trivial initial bounds.

Generality of Our Method. Our method is generic and can be easily extended to obtain polyhedral bounds for the non-linear operations in other architectures such as transformers [42] and capsule networks [34].

5   Scalable Certification of LSTMs

Next, we formally describe our scalable verifier for LSTM networks. As mentioned in Sect. 4, we build our verifier based on the DeepPoly abstract domain [39] introduced in Sect. 3.4. For simplicity, we focus on computing the polyhedral bounds for the output of non-linear operations. Note that the computed polyhedral bounds contain only the neurons from the previous layers. This restriction is required for backsubstitution used for computing the interval bounds of the inputs, which is an approximate algorithm for solving an LP (e.g. maximize or minimize x_j) within a polyhedral region defined by DeepPoly constraints. In Sect. 5.1, we show how to obtain tight, asymptotically optimal polyhedral bounds on key operations in the LSTM unit: $\sigma(x)\tanh(y)$ and $\sigma(x)y$. Section 5.2 describes a novel method to dynamically choose between different polyhedral bounds for increasing verifier precision.

5.1   Computing Polyhedral Abstractions of LSTM Operations

Our goal is to bound the products of *sigmoid and tanh* and *sigmoid and identity*, using lower and upper polyhedral planes parameterized by coefficients A_l, B_l, C_l and A_u, B_u, C_u, respectively. Let $f(x,y) = \sigma(x)\tanh(y)$ and $g(x,y) = \sigma(x)y$. For $h \in \{f,g\}$ we describe how to obtain the lower and upper bounds of h:

$$A_l \cdot x + B_l \cdot y + C_l \leq h(x,y) \leq A_u \cdot x + B_u \cdot y + C_u$$

We formulate the search for a lower bound of $h(x,y)$ as an optimization problem that minimizes the volume between the bound and the function, subject to the (soundness) constraint that the lower bound is below the function value:

$$\min_{A_l,B_l,C_l} \int_{(x,y)\in B} (h(x,y) - (A_l \cdot x + B_l \cdot y + C_l))$$

$$\text{subject to } A_l \cdot x + B_l \cdot y + C_l \leq h(x,y), \forall (x,y) \in B. \tag{1}$$

We denote $B = [l_x, u_x] \times [l_y, u_y]$ as the boundaries of input neurons x and y obtained using backsubstitution. We next describe our method to solve Eq. (1).

Step 1: Approximation via LP. We solve an approximation of the intractable optimization problem from Eq. (1), obtaining potentially unsound constraints. Unsoundness implies that there can be points in region B which violate the bounds. We build on the approach from [4], which proposes to approximate the objective in Eq. (1) using Monte Carlo sampling. Let $D = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ be a set of points from B sampled uniformly at random. We phrase the following optimization problem:

$$\min_{A_l, B_l, C_l \in \mathbb{R}} \sum_{i=1}^{n} (h(x_i, y_i) - (A_l \cdot x_i + B_l \cdot y_i + C_l))$$
$$\text{subject to } \bigwedge_{i=1}^{n} A_l \cdot x_i + B_l \cdot y_i + C_l \leq h(x_i, y_i). \tag{2}$$

Figure 3 shows an input region with Monte Carlo samples as red circles and summands in the LP objective as vertical lines. As this is a low-dimensional linear program (LP), we can solve it exactly in polynomial time using off-the-shelf LP solvers. We compute a candidate upper bound analogously.

Step 2: Adjusting the Offset to Guarantee Soundness. Since we compute the lower bound from a subset of points in B, there can be a point in B where the value of $h(x, y)$ is less than our computed lower bound. To ensure soundness, we compute $\Delta_l = \min_{(x,y) \in B} h(x, y) - (A_l \cdot x + B_l \cdot y + C_l)$ and then adjust the lower bound by updating the offset $C_l \leftarrow C_l + \Delta_l$, resulting in a sound lower bound plane. While the
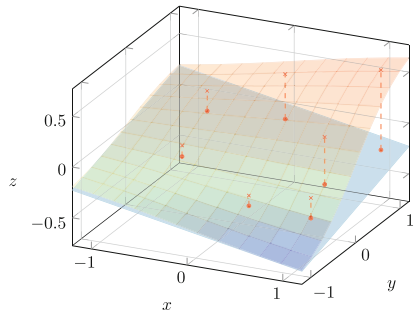

Fig. 3. Visualization of the $z = \sigma(x) \tanh(y)$ curve and the lower bound computed by linear programming. Red crosses represent the sampled points and dashed lines show the difference between the curve and the plane (summands in the optimization). (Color figure online)

method of [4] also performs offset calculation for obtaining sound bounds, they perform certification of image classifiers against geometric perturbations using expensive branch and bound for calculating the offset. In contrast, we exploit the structure of non-linearities used in LSTMs obtaining a closed-form formula for the offset yielding an exact solution. We now provide details of our offset adjustment method for $f(x, y) = \sigma(x) \tanh(y)$ and $g(x, y) = \sigma(x)y$.

Offset Calculation for $f(x, y) = \sigma(x) \tanh(y)$: Let $A_l \cdot x + B_l \cdot y + C_l$ be the initial lower bounding plane obtained from LP in region B. We define $F(x, y)$:

$$F(x, y) = \sigma(x) \tanh(y) - (A_l \cdot x + B_l \cdot y + C_l)$$

To find $\Delta_l = \min_{(x,y)\in B} F(x,y)$, we first find the extreme points by computing partial derivatives.

$$\frac{\partial F}{\partial x} = \sigma(x)\tanh(y)(1 - \sigma(x)) - A_l \tag{3}$$

$$\frac{\partial F}{\partial y} = \sigma(x)(1 - \tanh^2(y)) - B_l \tag{4}$$

We consider three cases:

- *Case 1: $x \in \{l_x, u_x\}$ and $y \in [l_y, u_y]$* Under this condition, we denote $S_x := \sigma(x)$ as a constant. To ease notation, let $t = \tanh(y)$ where $t \in [\tanh(l_y), \tanh(u_y)]$. Then $\frac{\partial F}{\partial y} \stackrel{!}{=} 0$ can be rewritten as:

$$1 - t^2 = B_l/S_x \tag{5}$$

- *Case 2: $y \in \{l_y, u_y\}$ and $x \in [l_x, u_x]$* Here we set $T_y := \tanh(y)$ and $s = \sigma(x)$, $x \in [\sigma(l_x), \sigma(u_x)]$ analogously. $\frac{\partial F}{\partial x} \stackrel{!}{=} 0$ is rewritten to:

$$s(1 - s) = A_l/T_y \tag{6}$$

- *Case 3: otherwise* Otherwise, we consider both $\frac{\partial F}{\partial x} \stackrel{!}{=} 0$ and $\frac{\partial F}{\partial y} \stackrel{!}{=} 0$. By combining Eq. (3) and Eq. (4), we reduce $\tanh(y)$ and obtain:

$$s^4 + (-2 - B_l)s^3 + (1 + 2B_l)s^2 + (-B_l)s - A_l^2 \stackrel{!}{=} 0 \tag{7}$$

Given that $F(x,y)$ is differentiable and the region B is compact, Fermat's theorem (stationary points) [1] states that F achieves its extremum at either the roots of Eq. (5), Eq. (6), and Eq. (7), or at the 4 corners of B. We evaluate F at these points to get Δ_l. We adjust the offset by replacing $C_l \leftarrow C_l + \Delta_l$. The adjusted F is no less than 0 on any point in B, which means that the plane with adjusted C_l becomes a sound lower bound of the $\sigma(x)\tanh(y)$ curve.

Offset Calculation for $g(x,y) = \sigma(x)y$: We next calculate the offset for $\sigma(x)y$. We define the differentiable function $G(x,y) = \sigma(x)y - (A_l \cdot x + B_l \cdot y + C_l)$ over the compact set B and compute:

$$\frac{\partial G}{\partial x} = \sigma(x)y(1 - \sigma(x)) - A_l \tag{8}$$

$$\frac{\partial G}{\partial y} = \sigma(x) - B_l \tag{9}$$

We use Fermat's theorem and consider three cases:

- *Case 1: $x \in \{l_x, u_x\}$ and $y \in [l_y, u_y]$* When $\sigma(x)$ is fixed, Eq. (9) is constant, which means G is monotonous in this case.

- *Case 2:* $y \in \{l_y, u_y\}$ and $x \in [l_x, u_x]$ Denote $s = \sigma(x)$ where $s \in [\sigma(l_x), \sigma(u_x)]$, then setting Eq. (8) $\stackrel{!}{=} 0$ becomes

$$s(1 - s) = A_l/y \tag{10}$$

- *Case 3: otherwise* If there is a local extremum in the region, the Hessian of G must be either positive-definite or negative-definite.

$$\frac{\partial^2 G}{\partial x^2} = \sigma(x)y(1 - \sigma(x))(1 - 2\sigma(x)), \frac{\partial^2 G}{\partial y^2} = 0, \frac{\partial^2 G}{\partial x \partial y} = \sigma(x)(1 - \sigma(x))$$

$$\frac{\partial^2 G}{\partial x^2} \cdot \frac{\partial^2 G}{\partial y^2} - \left(\frac{\partial^2 G}{\partial x \partial y}\right)^2 = -\left(\sigma(x)(1 - \sigma(x))\right)^2 < 0$$

Hence, there is no local extremum inside the boundaries.

To summarize, we only need to consider the roots of Eq. (10) to calculate the minimum of G to obtain Δ_l for $\sigma(x)y$. Figure 3 shows the lower bound plane obtained after solving the LP and adjusting the offset. We update the upper bound analogously.

Asymptotic Optimality. We can prove that, similarly to [4], as we increase the number of samples n, the solution of the LP asymptotically approaches the solution of the original problem from Eq. (1). Rephrasing and simplifying the theorem from [4]:

Theorem 1. *Let N be the number of points sampled in the algorithm. Let (ω_l, b_l) be our lower constraint (linear constraints and bias, respectively) and let $L(\omega^*, b^*)$ be the true minimum of function L. For every $\delta > 0$ there exists N_δ such that $|L(\omega_l, b_l) - L(\omega^*, b^*)| < \delta$ for every $N > N_\delta$, with high probability. Analogous result holds for upper constraint (ω_u, b_u) and function U.*

We denote $L = \int_{(x,y)} F(x, y)$ and (ω_l, b_l) are our A_l, B_l, C_l. Following the theorem, our sampling method guarantees the asymptotic optimality of our bounds. The theorem can be extended analogously for the upper bound.

5.2   Abstraction Refinement via Optimization

While our approach based on sampling, linear programming, and Fermat's theorem allows us to obtain (asymptotically) optimal bounds, it still has a fundamental limitation that it produces a *single* bound. Further, this approach is, in a sense, greedy: when considering the *entire* network, it is possible that selecting non-optimal planes for each neuron yields more precise results at the end. Neither the method from [21] nor the method in Sect. 5.1 achieves this. We present the first approach to learn an abstraction refinement that increases the end-to-end precision of certification.

Step 1: Compute a Set of Candidate Bounds. We adapt our approach from Sect. 5.1 to compute a set of candidate planes, instead of a single plane. We run

the sampling procedure multiple times, each time on a different subregion of the original region $B = [l_x, u_x] \times [l_y, u_y]$, with the constraints still enforced over the entire region B. We define four different triangular subdomains: $\mathcal{T}_1$ and $\mathcal{T}_2$ are triangles resulting from splitting B along the main diagonal, while $\mathcal{T}_3$ and $\mathcal{T}_4$ are triangles resulting from splitting B along the other diagonal. We additionally define $\mathcal{T}_0 = B$. For each $k \in \{0, 1, 2, 3, 4\}$, we perform sampling and optimization as in Eq. (2), this time sampling from $\mathcal{T}_k$ to obtain candidate lower bounds:

$$\min_{A_l, B_l, C_l \in \mathbb{R}} \sum_{i=1}^{n} (\sigma(x_i) \tanh(y_i) - (A_l \cdot x_i + B_l \cdot y_i + C_l))$$

$$\text{subject to } \bigwedge_{i=1}^{n} A_l \cdot x_i + B_l \cdot y_i + C_l \leq \sigma(x_i) \tanh(y_i) \text{ where } (x_i, y_i) \sim \mathcal{T}_k$$

For each neuron i, this yields 5 candidate lower bound and upper bound planes, LB_i^k and UB_i^k for $k \in \{0, 1, 2, 3, 4\}$. These five candidate planes for each of the N neurons are shown in Fig. 4.
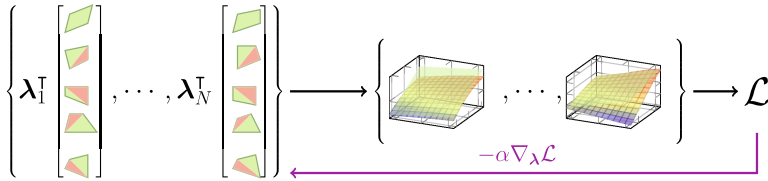

Fig. 4. Learning to combine linear bounds via gradient descent. Here the five candidate planes multiplied by $\boldsymbol{\lambda}$ are depicted either in green or red, or both. Green represents the sampled domain, $\mathcal{T}_k$, and red is the extension of the obtained green plane out of the domain. With the linear combination of the planes, we compute the bound, calculate the loss, and backpropagate. (Color figure online)

Step 2: Find the Optimal Combinations of the Bounds. Next, our goal is to learn a linear combination of the computed candidate bounds which yields the highest end-to-end certification precision for the given input region. To do this, we define the lower and upper bound of neuron i as a linear combination of the proposed five bounds:

$$LB_i = \sum_{k=0}^{4} \lambda_i^{LB} \cdot LB_i^k, \quad \sum_{k=0}^{4} \lambda_i^{LB} = 1, UB_i = \sum_{k=0}^{4} \lambda_i^{UB} \cdot UB_i^k, \quad \sum_{k=0}^{4} \lambda_i^{UB} = 1.$$

Recall that we formulate robustness certification as proving that for all labels i different from the ground truth label t: $z_t - z_i > 0$. The lower bound on $z_t - z_i$ is computed using backsubstitution [39], as shown in our overview example in Sect. 4. However, this lower bound now depends on the coefficients $\boldsymbol{\lambda}$, so we define the function $f(x, \epsilon, i, \boldsymbol{\lambda})$ which computes the lower bound of the expression $z_t - z_i$ when using $\boldsymbol{\lambda}$ to combine the neuron bounds.

We describe our approach to find the best coefficients $\boldsymbol{\lambda}$ in Algorithm 1. Consider the number of possible labels m and the number of binary operations of interest N_{ops}. To find $\boldsymbol{\lambda}$, we solve the optimization problem for each label i:

$$z_t - z_i > \max_{\boldsymbol{\lambda}} f(x, \epsilon, i, \boldsymbol{\lambda})$$

If the solution to the above optimization problem is positive, then we proved robustness with respect to class i. In the algorithm, we initialize $\tilde{\boldsymbol{\lambda}}$, a pre-normalized vector of $\boldsymbol{\lambda}$, for each neuron, uniformly between –1 and 1. Then, in each epoch, we compute the normalized $\boldsymbol{\lambda}$ by applying softmax to $\tilde{\boldsymbol{\lambda}}$ and run certification using $\boldsymbol{\lambda}$, obtaining a loss $\mathcal{L}$ equal to the value $-f(x, \epsilon, i, \boldsymbol{\lambda})$. We perform gradient descent update on $\tilde{\boldsymbol{\lambda}}$ based on the loss. If the loss is negative, we have found $\boldsymbol{\lambda}$ which proves the robustness and the algorithm terminates. The core updating flow is shown in Fig. 4.

Algorithm 1. Learning $\boldsymbol{\lambda}$ via gradient descent

Given input $\boldsymbol{x}$, label y, model $\mathcal{M}$, perturbation ϵ
Initialize the polyhedral abstractions and candidate bounds based on $\boldsymbol{x}$, $\mathcal{M}$ and ϵ.
for $i \leftarrow 1$ **to** m **where** $i \neq y$ **do**
    Initialize $\tilde{\boldsymbol{\lambda}} \sim [-1, 1]^{N_{ops} \times 5}$, $epoch \leftarrow 0$
    repeat
        $\boldsymbol{\lambda} \leftarrow \text{SoftMax}(\tilde{\boldsymbol{\lambda}})$, $\mathcal{L} \leftarrow -f(x, \epsilon, i, \boldsymbol{\lambda})$, $\tilde{\boldsymbol{\lambda}} \leftarrow \tilde{\boldsymbol{\lambda}} - \alpha \nabla_{\tilde{\boldsymbol{\lambda}}} \mathcal{L}$, $epoch \leftarrow epoch + 1$
    until $epoch = max\_epoch$ or $\mathcal{L} < 0$
    if $\mathcal{L} \geq 0$ **then**
        return not certified
    end if
end for
return certified

---

6   Certification of Speech Preprocessing

Speech preprocessing transforms the original set of perturbed speech signals, represented via intervals, through complex pipeline operations, into a non-linear and non-convex set. Propagating this set through the network is computationally expensive (infeasible for large models). To address this issue, we define precise overapproximations of key non-linear operations found in the speech preprocessing pipeline, such as Square and Log, expressed in the DeepPoly abstraction. These approximate bounds are computed via constant time closed form formulas based on concrete bounds of the inputs. We note that the first and third stages of the pipeline described in Sect. 3.3 involve an affine transformation, captured exactly using DeepPoly. Overall, when combined with our LSTM verifier, this method yields more precise end-to-end certification results than using intervals for approximating speech preprocessing.

Square. The lower and upper polyhedral bounds of the output of the square function $y = x^2$ where $x \in [l_x, u_x]$ are shown in Fig. 5a. We first consider the

bounds for y which minimize the area in the xy-plane. The upper bound UB_y is obtained by computing the chord joining the end points (l_x, l_x^2) and (u_x, u_x^2). The lower bound is a line parallel to UB_y passing through a point $((u_x + l_x)/2, ((u_x + l_x)/2)^2)$ in the middle of the curve.

$$LB_y = (u_x + l_x) \cdot x - ((u_x + l_x)/2)^2, UB_y = (u_x + l_x) \cdot x - u_x \cdot l_x.$$

While the above bounds would be sufficient in any other domain, they do not work for the speech domain as the subsequent Log requires that the input is strictly non-negative, as it is not defined for negative inputs. Also, we should carefully consider the floating point errors during calculations. Hence, we introduce the additional parameter $\delta \in \mathbb{R}$, a small threshold value to ensure the lower bound stays non-negative. In our experiments, we set $\delta = 1 \times 10^{-5}$. Upper and lower bounds for $y = x^2$ are computed as $UB_y = (u_x + l_x) \cdot x - u_x \cdot l_x$ and $LB_y =$
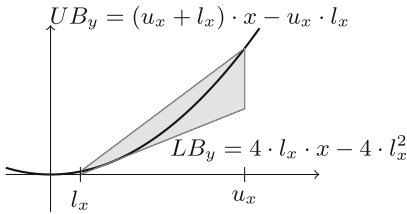
(a) Abstraction for square function with threshold $\delta = 0$
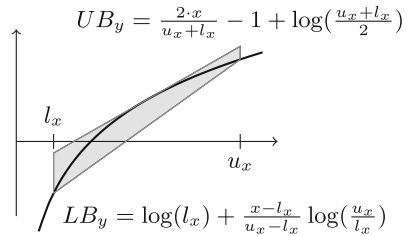
(b) Abstraction for log function.

Fig. 5. Two polyhedral abstractions for the speech preprocessing stage.

$$\begin{cases} 2 \cdot (l_x + \sqrt{l_x^2 - \delta}) \cdot x - (l_x + \sqrt{l_x^2 - \delta})^2 & 3 \cdot l_x^2 + 2 \cdot l_x.u_x - u_x^2 \le 4 \cdot \delta, \sqrt{\delta} \le l_x \\ 2 \cdot (u_x - \sqrt{u_x^2 - \delta}) \cdot x - (u_x - \sqrt{u_x^2 - \delta})^2 & 3 \cdot u_x^2 + 2.u_x.l_x - l_x^2 \le 4 \cdot \delta, u_x \le -\delta \\ 0 & l_x \le \sqrt{\delta}, -\sqrt{\delta} \le u_x \\ (u_x + l_x) \cdot x - ((u_x + l_x)/2)^2 & o.w. \end{cases}$$

Log. We define the polyhedral abstraction of the output $y = \log(x)$ of the log operation where $x \in [l_x, u_x]$, as shown in Fig. 5b. Our abstractions are optimal and minimize the area in the xy-plane. The lower bound LB_y is the chord joining the end points $(l_x, \log(l_x))$ and $(u_x, \log(u_x))$. The upper bound UB_y is obtained by computing a line parallel to LB_y passing through the middle of the curve at $((u_x + l_x)/2, \log((u_x + l_x)/2))$. Our final abstractions are:

$$LB_y = \log(l_x) + \frac{x - l_x}{u_x - l_x} \log(\frac{u_x}{l_x}), UB_y = \frac{2 \cdot x}{u_x + l_x} - 1 + \log(\frac{u_x + l_x}{2}).$$

7   Experimental Evaluation

We implemented our approach in a verifier called PROVER, using PyTorch [30] and Gurobi 9.0 for solving linear programs. The code is available in https://github.com/eth-sri/prover. We evaluate PROVER on speech classifiers for FSDD [17] and GSC v2 [44] datasets. Then, we compare PROVER against POPQORN [21] on the MNIST image classification task proposed by it. We note that POPQORN does not scale to the speech classifiers considered in our work. We demonstrate further scalability by verifying large motion sensor sequence classifier trained on HAPT [33] dataset containing 256 hidden dimensional 4 layered LSTM units.

Setup. GSC dataset experiments ran on an Nvidia GeForce RTX 2080, while the rest ran on a single Tesla V100. Following convention from prior work [39], we consider only those inputs that are classified correctly without perturbation. We use the same set of hyperparameters for the experiments unless specifically mentioned. We use 100 sampling points for constructing the linear program and optimize λ parameters using Adam [20] for 100 epochs. During optimization, we initialize the learning rate to 100 and multiply it by 0.98 after every epoch.

7.1   Speech Classification

We certify the robustness of two speech classifiers for the FSDD and GSC v2 datasets. FSDD consists of recordings of digits spoken by six different speakers, recorded at 8 kHz. GSC has 35 distinct labels of single word utterances at 16 kHz. We compare our base method based on sampling and linear programming (Sect. 5.1), denoted as PROVER (LP), and our method using abstraction refinement via optimization (Sect. 5.2), denoted as PROVER (OPT).

Preprocessing. A key challenge in speech classification, not encountered in the image domain, is the complex preprocessing stage before the LSTM network. The preprocessing stage in this experiment consists of FFT and Mel-filter transformations. Preprocessed input then passes through the fully connected layer with ReLU activation followed by the LSTM unit.

FSDD Certification. We used the following parameters for the preprocessing: we slice the raw wave signal with length 256 using a stride of 200 with 10 Mel-frequencies. For this experiment, we trained an LSTM network with two LSTM layers and 32 hidden units each, preceded by a 40 ReLU-activated fully-connected layer. This network achieves an accuracy of 83.6% on the FSDD task. The average number of frames was 14.7. We verify the first 100 correctly classified inputs for each perturbation. Our perturbation metric on speech classification tasks is described in Sect. 3.1. Our results are shown in Fig. 6a and Fig. 6b. We vary the decibel perturbation between –90 dB and –70 dB and evaluate the precision and runtime of PROVER. Figure 6a shows the percentage of certified samples: our method based on optimizing the bounds (OPT) performs best, e.g., certifying twice as many samples compared to LP, for a significant perturbation of –70
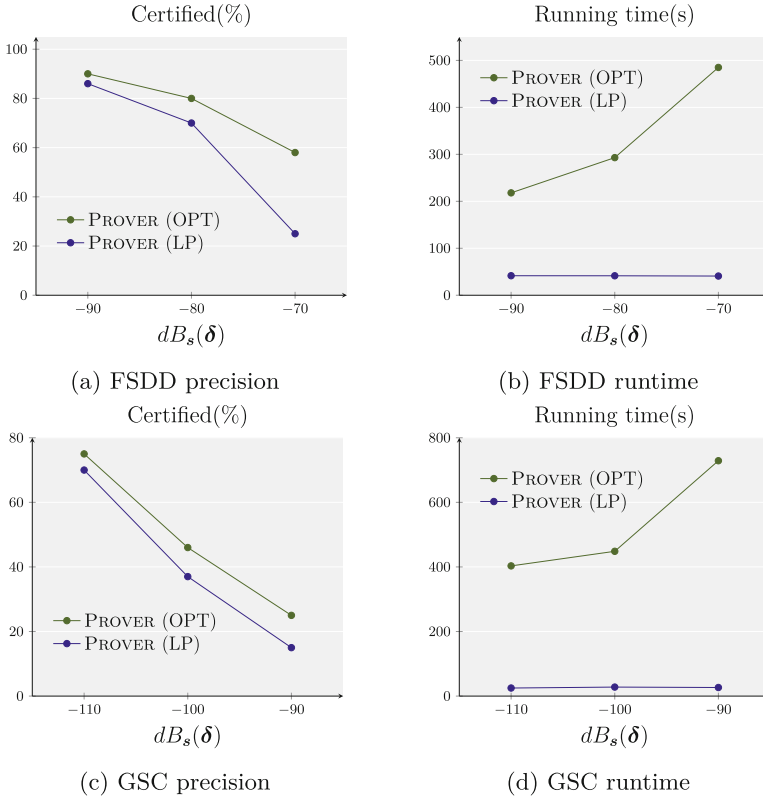
Fig. 6. Performance plots for the FSDD and GSC datasets with different perturbations. All tests are done with the same architecture described in the text.

dB. In terms of runtime, Fig. 6b shows that the OPT runtime increases with the perturbation magnitude, meaning that the optimizer needs more iterations to converge to the resulting bounds.

Interval vs. Polyhedral Abstraction for Speech Preprocessing. We studied experimentally the importance of designing precise polyhedral abstractions of the speech preprocessing pipeline. If we replace the polyhedral bounds for the square and logarithm operations with interval constraints, the precision of PROVER (LP) drops from 86% to 61% on –90 dB and from 70% to 20% on –80 dB. This shows the importance of keeping relational information while overapproximating the speech preprocessing pipeline.

GSC Certification. We used the following parameters for the preprocessing: we downsample the raw input to 8 kHz, sliced the signal in length 1024, followed by 10 Mel-frequency filterbanks. As with the FSDD architecture, we used two layers of LSTM and 50 hidden units each, preceded by a 50 ReLU-activated fully-connected layer. This network achieves accuracy of 80% on the GSC task.

Certifying the GSC classifier is more challenging than FSDD: this dataset has 35 labels, compared to 10 in FSDD. The larger label set size requires PROVER to compare 34 output differences - acquiring the lower bounds of $l_{GT} - l_{FL}$ where each term stands for the final output score for the ground truth and false label, respectively. Figure 6c shows the percentage of certified samples: 75% on –110 dB and 46% on –100 dB with PROVER (OPT), again higher precision than PROVER (LP). Figure 6d shows the longer running time for PROVER (OPT) than on FSDD, due to its larger label set size.

7.2   Image Classification

Based on the setup from [21], we flatten each image into a vector of dimension 784. This vector is partitioned into a sequence of f frames (f depends on the experiment). Next, the LSTM uses this frame as an input.

Comparison with POPQORN. We compare the precision and scalability of PROVER against POPQORN [21]. We trained an LSTM network containing 1 layer with 32 hidden units using standard training, achieving an accuracy of 96.5%. The network receives a sequence of $f = 7$ image slices as input and predicts a digit corresponding to the image.

As POPQORN is slow, we used only ten correctly classified images randomly sampled from the test set. For each frame index i and each method, we compute the maximum perturbation bound ϵ such that the method can certify that the LSTM classifier is robust to perturbations up to ϵ in the L_∞-norm of the i-th slice of the image. We determine the maximum ϵ using the same binary search procedure as in [21].

Table 1. Certification of several LSTM models using PROVER with $\epsilon = 0.01$. F, H, and L denote the number of frames, LSTM hidden units and layers respectively.

F	H	L	Accuracy (%)	Certified (%) by OPT/LP	by OPT Running time (s)
4	**32**	**1**	96.1	91/89	14.5
4	32	**2**	96.7	92/73	29.1
4	32	**3**	95.8	95/65	43.1
4	**64**	1	97.3	93/92	27.0
4	**128**	1	97.1	95/95	52.4
7	32	1	96.5	63/56	32.1

Figure 7 presents the results of this experiment. We observe that, for all three methods, early frames have smaller ϵ certified perturbation bounds than the later frames. The reason is that the approximation error on frame 1 propagates through the later frames to the classifying layer, while the error on frame 7 only affects the last layer. Across all frames, both our LP and OPT methods significantly outperform POPQORN, meaning that PROVER enables a more precise abstraction than POPQORN. As for speech classifiers, OPT is more precise than LP. We compare running times of the three methods on perturbations in the first frame – most challenging as it requires propagating through all timesteps. Here, PROVER (LP), PROVER (OPT), and POPQORN

Fig. 7. Results for the comparison between PROVER and POPQORN. Plotted points represent the maximum L_∞ norm perturbation for each frame index 1 through 7.

take 65, 348, and 2,160 s respectively per example on average. We conclude that both variants of PROVER are more precise than POPQORN while being 33.2× and 6.21× more scalable for LP and OPT respectively.

Effect of Model Size. We evaluate the scalability of PROVER by certifying several recurrent architectures, with varying number of frames F, hidden units H and LSTM layers L. For each network, we certify the first 100 correctly classified images using the same perturbation $\epsilon = 0.01$ for all frames, with 3 repetitions. While in the previous experiment we certified each frame separately to closely follow the setup from [21], it is more natural to assume the adversary is able to perturb the entire input. The results are shown in Table 1. We observe that the precision of PROVER is affected mostly by the number of frames, as the precision loss accumulates along the frames. Naturally, the running time increases with the number of neurons and frames, as PROVER is optimizing the bounds for each $\sigma(x)\tanh(y)$ operation. However, we also observe a counter-intuitive phenomenon that PROVER (OPT) performs better with multi-layer models than with the single-layer model. The precision from PROVER (LP) drops with the number of LSTM layers unlike those from PROVER (OPT). We hypothesize that an increased number of trainable parameters enhances the flexibility of the bounds for the optimization, allowing us to find more combinations of the bounds that certify the input. PROVER (LP) has non-flexible bounds, so the error propagates.

Effect of Perturbation Budget. We certify the robustness of the MNIST classifier for different ϵ values. We again evaluated 100 correctly classified samples from the test set. Figure 8 shows the experimental results. The OPT version has
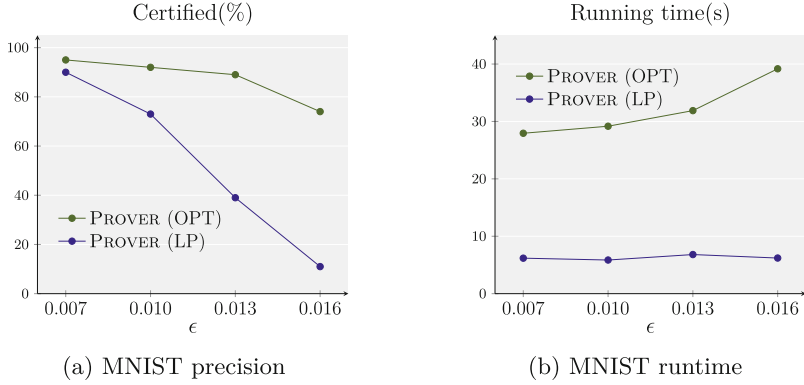
Fig. 8. Results on MNIST with different epsilons and $F = 4, H = 32, L = 2$.

significantly higher precision than LP: i.e., for $\epsilon = 0.013$ in Fig. 8a, LP proves 39% while OPT certifies 89% of samples with a higher runtime in Fig. 8b.

7.3 Motion Sensor Data Classification

We further demonstrate the scalability of PROVER by considering a large classifier containing 4 LSTM layers with 256 hidden units each for the human activity recognition dataset HAPT [33]. Each input in the dataset consists of recorded triaxial linear accelerations and angular velocities, sampled 50 Hz. Here, we restricted HAPT to six activity classes and we trimmed angular velocities to at most 6 s after the point of prediction. Each input sequence is sliced into sliding windows of 0.5 s, which are then passed as an input to the classifier. The trained classifier achieves 88% test accuracy. Identical to the other experiments, we run PROVER on the first 100 correctly classified inputs.
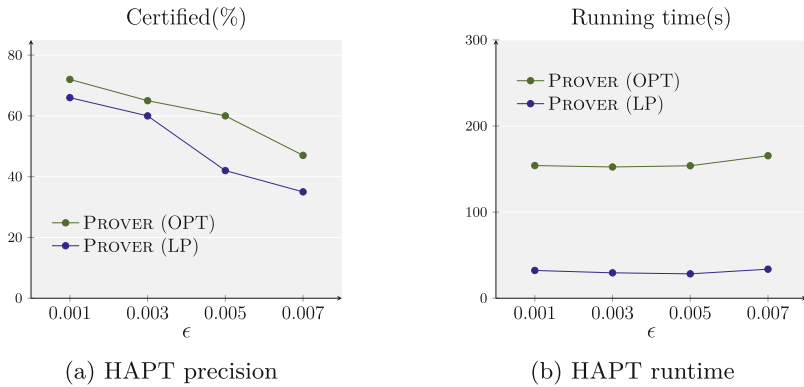

Fig. 9. Results on HAPT with different epsilons and $H = 256, L = 4$.

Results, shown in Fig. 9, indicate that PROVER (OPT) verifies more inputs than PROVER (LP), for all perturbation budgets. Although the number of parameters has increased, Fig. 9b shows smaller running times compared to Fig. 6b and Fig. 6d. This is because of the smaller number of classes in HAPT, as the verification needs to perform the backsubstitution for each incorrect class. This result shows that PROVER (i) is applicable to LSTM classifiers in various domains, and (ii) scales to the large models.

8   Conclusion

We introduced a novel approach for certifying RNNs based on a combination of linear programming and abstraction refinement. The key idea is to compute a polyhedral abstraction of the non-linear operations found in the recurrent cells and to dynamically adjust this abstraction according to each input example being certified. Our experimental results show that PROVER is more precise and scalable than prior work. These advances enable PROVER to certify, for the first time, the robustness of LSTM-based speech classifiers.

References

1. Fermat's theorem. https://planetmath.org/fermatstheoremstationarypoints
2. Akintunde, M.E., Kevorchian, A., Lomuscio, A., Pirovano, E.: Verification of RNN-based neural agent-environment systems. In: Proceedings of the AAAI Conference on Artificial Intelligence (2019)
3. Angluin, D.: Learning regular sets from queries and counterexamples. Inf. Comput. **75**(2), 87–106 (1987)
4. Balunovic, M., Baader, M., Singh, G., Gehr, T., Vechev, M.: Certifying geometric robustness of neural networks. In: Advances in Neural Information Processing Systems (2019)
5. Balunovic, M., Vechev, M.: Adversarial training and provable defenses: Bridging the gap. In: International Conference on Learning Representations (2020)
6. Biggio, B., et al.: Evasion attacks against machine learning at test time. In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases (2013)
7. Carlini, N., et al.: Hidden voice commands. In: 25th {USENIX} Security Symposium ({USENIX} Security 16), pp. 513–530 (2016)
8. Carlini, N., Wagner, D.: Audio adversarial examples: Targeted attacks on speech-to-text. In: 2018 IEEE Security and Privacy Workshops (SPW) (2018)
9. Cho, K., et al.: Learning phrase representations using RNN encoder-decoder for statistical machine translation. In: EMNLP, pp. 1724–1734 (2014)
10. Cousot, P., Cousot, R.: Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: Proceedings of Principles of Programming Languages, ACM (1977)
11. Diro, A., Chilamkurti, N.: Leveraging LSTM networks for attack detection in fog-to-things communications. IEEE Commun. Mag. **56**(9), 124–130 (2018)
12. Dvijotham, K., et al.: Training verified learners with learned verifiers. arXiv:1805.10265 (2018)

13. Fischer, M., Baader, M., Vechev, M.: Certification of semantic perturbations via randomized smoothing. arXiv:2002.12463 (2020)
14. Hannun, A., et al.: Deep speech: Scaling up end-to-end speech recognition. arXiv:1412.5567 (2014)
15. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (1997)
16. Hu, W., Tan, Y.: Black-box attacks against RNN based malware detection algorithms. arXiv:1705.08131 (2017)
17. Jackson, Z.: Free spoken digit dataset (2020). https://github.com/Jakobovski/free-spoken-digit-dataset
18. Jacoby, Y., Barrett, C., Katz, G.: Verifying recurrent neural networks using invariant inference. arXiv:2004.02462 (2020)
19. Khmelnitsky, I., et al.: Property-directed verification of recurrent neural networks. arXiv:2009.10610 (2020)
20. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv:1412.6980 (2014)
21. Ko, C., Lyu, Z., Weng, T., Daniel, L., Wong, N., Lin, D.: Popqorn: Certifying robustness of recurrent neural networks. In: International Conference on Machine Learning (ICML) (2019)
22. Li, J., Qu, S., Li, X., Szurley, J., Kolter, J.Z., Metze, F.: Adversarial music: real world audio adversary against wake-word detection system. In: Advances in Neural Information Processing Systems, pp. 11908–11918 (2019)
23. Lyu, Z., Ko, C.Y., Kong, Z., Wong, N., Lin, D., Daniel, L.: Fastened crown: Tightened neural network robustness certificates. arXiv:1912.00574 (2019)
24. Miné, A.: A new numerical abstract domain based on difference-bound matrices. In: Proceedings of Programs As Data Objects (PADO), pp. 155–172 (2001)
25. Miné, A.: The octagon abstract domain. High. Order Symbolic Comput. **19**(1), 31–100 (2006)
26. Neekhara, P., Hussain, S., Pandey, P., Dubnov, S., McAuley, J., Koushanfar, F.: Universal adversarial perturbations for speech recognition systems. arXiv:1905.03828 (2019)
27. Pachocki, J., et al.: Openai five. https://blog.openai.com/openai-five (2018)
28. Papernot, N., McDaniel, P., Swami, A., Harang, R.: Crafting adversarial input sequences for recurrent neural networks. In: MILCOM 2016–2016 IEEE Military Communications Conference, pp. 49–54. IEEE (2016)
29. Pascual, S., Bonafonte, A., Serra, J.: Segan: Speech enhancement generative adversarial network. arXiv:1703.09452 (2017)
30. Paszke, A., et al.: Automatic differentiation in pytorch (2017)
31. Qin, Y., Carlini, N., Goodfellow, I., Cottrell, G., Raffel, C.: Imperceptible, robust, and targeted adversarial examples for automatic speech recognition. In: International Conference on Machine Learning (ICML) (2019)
32. Raghunathan, A., Steinhardt, J., Liang, P.S.: Semidefinite relaxations for certifying robustness to adversarial examples. In: Advances in Neural Information Processing Systems (NeurIPS), pp. 10877–10887 (2018)
33. Reyes-Ortiz, J.L., Oneto, L., Samà, A., Parra, X., Anguita, D.: Transition-aware human activity recognition using smartphones. Neurocomputing **171**, 754–767 (2016)
34. Sabour, S., Frosst, N., Hinton, G.E.: Dynamic routing between capsules. In: Proceedings of Neural Information Processing Systems (NIPS), pp. 3856–3866 (2017)

35. Sahidullah, M., Saha, G.: Design, analysis and experimental evaluation of block based transformation in MFCC computation for speaker recognition. Speech Commun. **54**(4), 543–565 (2012)
36. Sainath, T.N., Weiss, R.J., Senior, A., Wilson, K.W., Vinyals, O.: Learning the speech front-end with raw waveform CLDNNs. In: Sixteenth Annual Conference of the International Speech Communication Association (2015)
37. Singh, G., Ganvir, R., Püschel, M., Vechev, M.: Beyond the single neuron convex barrier for neural network certification. In: Advances in Neural Information Processing Systems (2019)
38. Singh, G., Gehr, T., Mirman, M., Püschel, M., Vechev, M.: Fast and effective robustness certification. In: Advances in Neural Information Processing Systems (2018)
39. Singh, G., Gehr, T., Püschel, M., Vechev, M.: An abstract domain for certifying neural networks. In: Proceedings of Principles of Programming Languages (POPL) (2019)
40. Singh, G., Püschel, M., Vechev, M.: Fast polyhedra abstract domain. In: Proceedings of Principles of Programming Languages (POPL), pp. 46–59 (2017)
41. Szegedy, C., et al.: Intriguing properties of neural networks. arXiv:1312.6199 (2013)
42. Vaswani, A., et al.: Attention is all you need. In: Proceedings of Neural Information Processing Systems (NIPS) (2017)
43. Vinyals, O., et al.: Alphastar: Mastering the real-time strategy game starcraft ii. DeepMind blog p. 2 (2019)
44. Warden, P.: Speech commands: A dataset for limited-vocabulary speech recognition. arXiv:1804.03209 (2018)
45. Wu, M., Kwiatkowska, M.: Robustness guarantees for deep neural networks on videos. arXiv:1907.00098 (2019)
46. Yang, Z., Li, B., Chen, P.Y., Song, D.: Characterizing audio adversarial examples using temporal dependency (2019)
47. Zhang, H., Weng, T., Chen, P., Hsieh, C., Daniel, L.: Efficient neural network robustness certification with general activation functions. In: Advances in Neural Information Processing Systems, (NeurIPS) (2018)