GRENA: GPU-aided Abstract Refinement for Neural Network Verification

Yuyi Zhong[1†], Shaun Tan Zong Zhi[1], Hanping Xu[1], and Siau-Cheng Khoo[2]

[1] National University of Singapore
{yuyizhong,shauntanzongzhi,xuhanping}@u.nus.edu
[2] National University of Singapore
khoosc@nus.edu.sg

Abstract. Since neural network verification problems can be formulated as optimization problems, linear programming (LP) solvers have been deployed as off-the-shelf tools in such processes. However, existing LP solvers running on CPU scale poorly on large networks. To expedite the process, we propose an LP-solving *theorem* tailored to neural network verification. In practice, we transform the constrained solving problem into an unconstrained problem that can be executed on GPUs, significantly speeding up the solving process. We explicitly include constraints on layers that take more than one predecessor instead of handling multiple predecessors by inefficient concatenation. Our theorem applies to widely used networks, such as fully connected, convolutional, and residual networks. From our evaluation, our GPU-aided solver achieves comparable precision to the state-of-the-art (SOTA) solver GUROBI with significant speed improvements and helps acquire competitive verification precision compared to advanced verification methods.

Keywords: Abstract Refinement · Linear Programming · Neural Network Verification.

1 Introduction

Researchers have investigated the verification of neural networks due to their wide application [25,18,28]. Throughout the evolution of verification techniques, abstract interpretation-based techniques [20,6,22,24,23,27,32,15] continue to play an important role. However, due to the nature of over-approximation, the methods could suffer from severe precision loss for deeper networks. Theoretically, such abstraction can be refined with the help of (mixed integer) linear programming (MILP or LP) [21,31,34] where GUROBI [11] solver is commonly used despite the scalability concern that it executes on the CPU.

Therefore, we propose a tailored theorem to accelerate LP solving for abstract-refinement-based methods. Notably, our theorem could handle three types of constraints: output constraints, intermediate neuron constraints and constraints of layers that take more than one predecessor, which enhances the scientific rigor

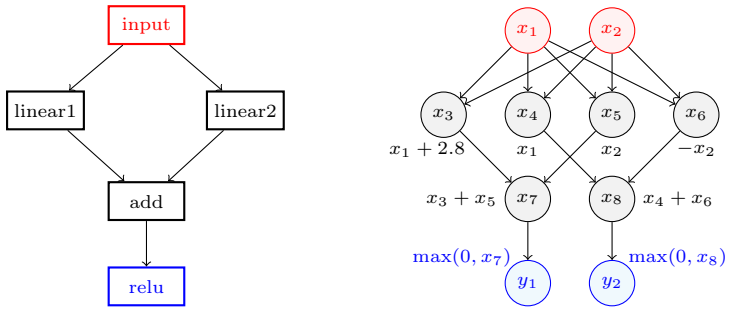
† Corresponding author

of the verification of residual networks. Our paper offers a methodical transformation from the stage of verification specification to the stage of effective implementation as an analyzer named GRENA (GPU-aided abstract REfinement for Neural network verificAtion), and we assess it against the state-of-the-art tools to empirically support its strong solving and verification capacities. Our dockerized system, data, usage documentation and experiment scripts are available at https://github.com/Grena-verifier/Grena-verifier. We summarize our contributions below:

- We propose a novel, formal and rigorous theorem to solve constrained optimization problems that include output constraints, multi-ReLU constraints, and complex constraints of residual network layers. Specifically, to the best of our knowledge, this is the first work that uses Lagrangian dual on spurious-adversarial-label guided refinement process to enhance the scientific rigor of the verification of residual networks.
- We utilize the multi-ReLU abstraction in WraLU [12] to further tighten our constraint set for precision improvement.
- We provide strong and effective implementations and demonstrate the verification efficiency of our system through empirical experiments, and deliver a video showcase [3] of our analyzer.

2 Overview

To provide an intuitive understanding, we use an example in Figure 1 to show how the approach works given the network and the input space $I = [-1, 1] \times [-1, 1]$ of 2 input neurons x_1, x_2. This network has 2 output neurons y_1, y_2, corresponding to two labels L_1, L_2 that an input can be classified as, and we aim to verify that $y_1 - y_2 > 0$ for all inputs in I.



(a) The network with an add layer (b) Neuron connections

Fig. 1: The example network to be verified with $y_1 - y_2 > 0$ with input space $I = [-1, 1] \times [-1, 1]$

[3] https://drive.google.com/file/d/17v1WnabNrzC-ZwJzJ4dLTQmm9JvDYfj5/view?usp=sharing

We first apply the abstract interpretation technique, as deployed by DeepPoly [20], to compute the reachable statuses for each neuron at Figure 1b and represent them by four elements (l_i, u_i, l_i^s, u_i^s). The *concrete* lower and upper bounds l_i, u_i form an interval $[l_i, u_i]$ that over-approximates all the values that neuron x_i could take. The *symbolic* constraints l_i^s, u_i^s are linear expressions of x_i defined over preceding neurons while satisfying $l_i^s \leq x_i \leq u_i^s$. The abstract values are displayed near the corresponding nodes at Figure 2.

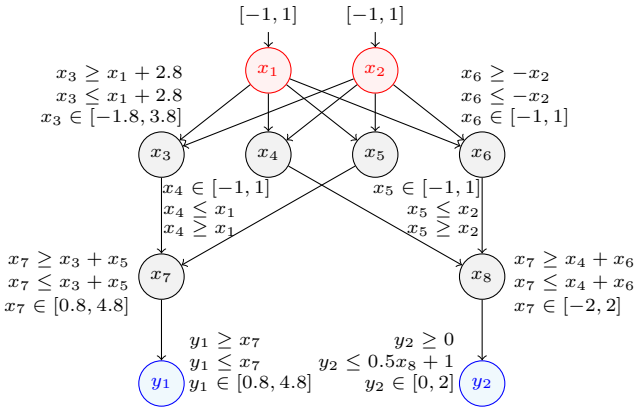


Fig. 2: The network to perform abstract interpretation

Based on the abstraction, the computed value for the lower bound of $y_1 - y_2$ is -0.2 ($y_1 - y_2$ will be treated as an auxiliary neuron in order to compute its lower bound, the details can be found in [20]), failing to assert that $y_1 - y_2 > 0$. However, this failure is due to the over-approximation error, and there is no such instance that leads to $y_1 - y_2 < 0$. To prove $y_1 - y_2 < 0$ to be infeasible, we will construct a constraint set that encodes the existence of spurious counterexamples together with the network constraints (conjunction of all linear inequities including the *concrete* bounds and *symbolic* constraints of all neurons). Based on the constraint set, we send it to our tailored LP solver (details of our solving theorem at subsection 3.1) to resolve the concrete bounds of input neurons (x_1, x_2) and those linear neurons (x_8) that are followed by ReLU and take both negative and positive values. The returned bounds will be tighter, as shown in Figure 3, and diminish the inconclusiveness produced by the previous abstract interpretation.

Based on the updated bounds, we rerun abstract interpretation and update the abstract values of all neurons accordingly, as shown in Figure 4. Based on the new abstraction, the lower bound of $y_1 - y_2$ is 0.7, making $y_1 - y_2 \leq 0$ actually infeasible, which means that y_1 dominates over y_2 and we could conclude that $y_1 - y_2 > 0$.

In summary, our system uses LP solving and abstract interpretation to eliminate adversarial labels that are actually infeasible. Note that in our con-

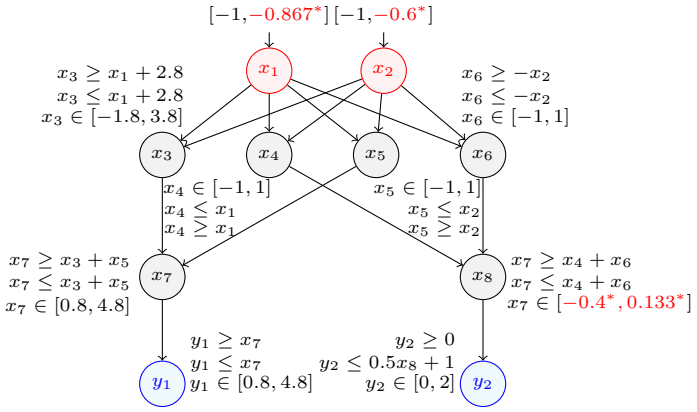


Fig. 3: The result of resolving bounds (in red marked by *)

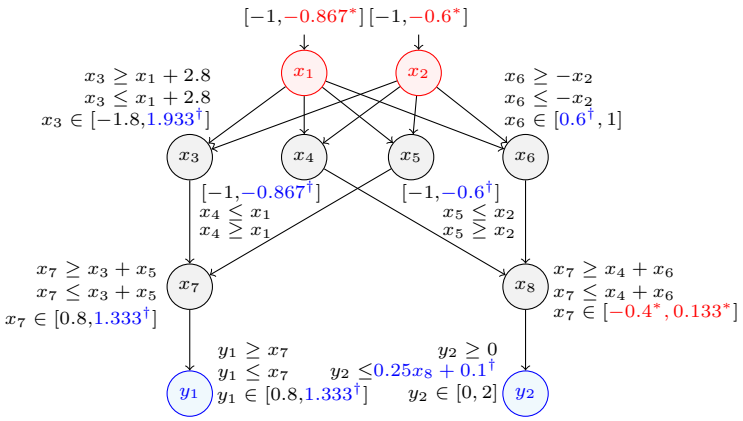


Fig. 4: The updated abstract values (in blue marked by †)

straint set, we explicitly encode an Add layer that takes two predecessors as $[x_7, x_8] = [x_3, x_4] + [x_5, x_6]$, and we will elaborate on how our theorem handles two predecessors at Theorem 1 instead of simply concatenating two predecessors into one in an engineering manner.

3 Methodologies

We provide a simplified case that only contains one adversarial label y_2 in the previous section. But in general, the verification process repeatedly selects multiple adversarial labels and attempts to eliminate them through iterations of refinements as illustrated in Figure 5. In each iteration, we take the encoding of multiple adversarial labels (the disjunction is handled by following the convention in [34]), the current network abstraction, plus the SOTA *WraLU multi-neuron*

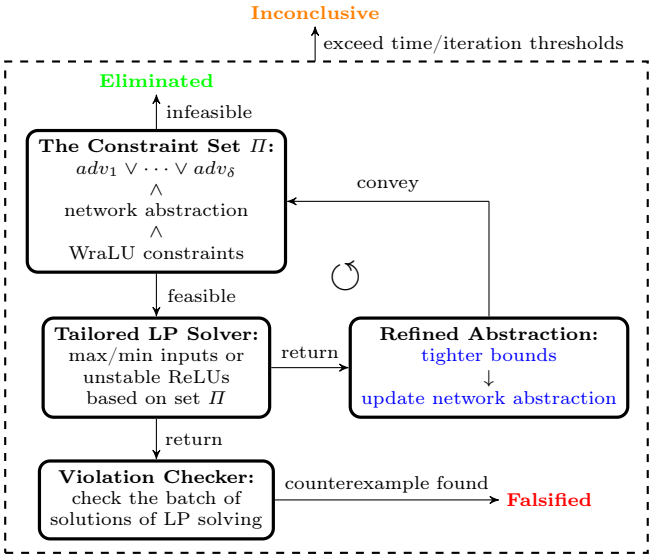


Fig. 5: The iterative process of abstract refinement

constraints as the constraint set. We eliminate δ spurious adversarial labels if the constraint set is infeasible, and eliminating all adversarial labels results in successful verification. If the constraint set is feasible, we send it to our *tailored LP solver on GPU* (details deferred till subsection 3.1) and resolve neuron bounds to obtain a refined abstraction, where the refined abstraction is used in the next iteration. Furthermore, as a feasible constraint set indicates the possibility of a property violation, we collect *the batch of input neuron assignments* during each solving substep and pass them to the model to check if they constitute an adversarial example which falsifies the property. We repeat this process until a conclusive result is obtained; or until the time/iteration threshold has exceeded, indicating inconclusive result.

3.1 GPU-aided Linear Programming Solver

This subsection presents our theorem of transforming a constrained linear programming problem into an unconstrained solving problem *amenable to GPU acceleration*.

Preliminaries. Given a network with $L+1$ layers and each layer corresponds to a layer index, the *input layer* is at index 0 and the output layer is at index L. We denote the set of all ReLU layer indexes as $[R]$, the set of all linear layer indexes with one connected preceding layer as $[L_1]$, the set of all indexes of linear layers that take two preceding layers as $[L_2]$. We assume that $[R] \cup [L_1] \cup [L_2] = [1, \ldots, L]$ and both $1, L \in [L_1]$. The output and input/preceding layer of a ReLU layer are respectively represented by $\hat{x}^{(i)}$ and $\hat{x}_p^{(i)}$, for $i \in [R]$. Given a neuron index j and a layer index i, $\hat{x}^{(i)j}$ represents the j-th neuron at i-th layer and $\hat{x}_p^{(i)j}$ refers to its input neuron. Symbol $x^{(i)}, i \in [L_1] \cup [L_2]$ represents the output of a

linear layer; symbols $\hat{x}^{(0)}, x^{(0)}$ both denote the input layer. Symbol $x_p^{(i)}, i \in [L_1]$ refers to the predecessor of layer $x^{(i)}$ for $i \in [L_1]$; whereas $x_{p_1}^{(i)}, x_{p_2}^{(i)}$ are the two preceding layers of layer $x^{(i)}$ for $i \in [L_2]$. Finally, we designate $S(i)$ as a set that includes the indexes of all connected succeeding layers of layer i and $i_s \in S(i)$; the set $S^2(i) = \cup_{i_s \in S(i)} S(i_s)$, which includes the successors' indexes of succeeding layers of layer i and $i_{s^2} \in S^2(i)$.

Theorem 1. *The constrained optimization problem in neural network verification (as shown in Equation 1) can be transformed into an unconstrained problem in Equation 2 by using Lagrangian dual.*

Proof. The derivation can be found at this appendix [4].

In detail, the constrained problem formulation is given as:

$$
\begin{aligned}
&\min_{x,\hat{x}} c^{(0)}\hat{x}^{(0)} + \sum_{i\in[R]} c^{(i)T}\hat{x}_p^{(i)} \\
&\text{s.t. } l^{(0)} \le \hat{x}^{(0)} \le u^{(0)}; Hx^{(L)} + d \le 0 \\
&x^{(i)} = W^{(i)}x_p^{(i)} + b^{(i)}, \text{for } i \in [L_1] \\
&x^{(i)} = x_{p_1}^{(i)} + x_{p_2}^{(i)}, \text{for } i \in [L_2] \\
&\hat{x}^{(i)j} = \hat{x}_p^{(i)j}, \text{for } i \in [R], j \in I^{+(i)} \\
&\hat{x}^{(i)j} = 0, \text{for } i \in [R], j \in I^{-(i)} \\
&\hat{x}^{(i)j} \ge 0, \hat{x}^{(i)j} \ge \hat{x}_p^{(i)j}, \text{for } i \in [R], j \in I^{\pm(i)} \\
&\hat{x}^{(i)j} \le \frac{u^{(i)j}}{u^{(i)j} - l^{(i)j}}(\hat{x}_p^{(i)j} - l^{(i)j}), \text{for } i \in [R], j \in I^{\pm(i)} \\
&P^{(i)}\hat{x}_p^{(i)} + \hat{P}^{(i)}\hat{x}^{(i)} - p^{(i)} \le 0, \text{for } i \in [R]
\end{aligned}
\tag{1}
$$

In detail, $l^{(0)}, u^{(0)}$ record the lower and upper bounds of input neurons; $Hx^{(L)} + d \le 0$ represents the output constraints that encode the existence of multiple adversarial examples. For ReLU neurons, their functionalities depend on the stability statuses. For example, suppose a linear layer i is followed by a ReLU layer i_s. A ReLU neuron is stably activated if it takes a non-negative input interval, in which case it equals the input neuron, and we collect the indexes of those non-negative input neurons at layer i as $I^{+(i)}$. Stably deactivated ReLU neurons have non-positive inputs, with outputs that are always evaluated to 0, and we denote the indexes of those non-positive input neurons as a set $I^{-(i)}$. Unstable ReLU neurons take both positive and negative input values, their corresponding input neuron indexes are recorded in $I^{\pm(i)}$. In particular, the unstable ReLU neuron is approximated by an orange-colored triangle shape as Figure 6 illustrates, where $l^{(i)j}, u^{(i)j}$ record its input interval and $\frac{u^{(i)j}}{u^{(i)j} - l^{(i)j}}$ is abbreviated as $s^{(i)j}$.

Constraints $P^{(i)}\hat{x}_p^{(i)} + \hat{P}^{(i)}\hat{x}^{(i)} - p^{(i)} \le 0$ capture the dependencies of multiple ReLU neurons in the same layer, which is obtained from the WraLU[12] method

[4] https://github.com/Grena-verifier/misc-files/blob/master/theorem_proof.pdf

to improve solving precision. The coefficients $c^{(0)}$ and $c^{(i)}, i \in [R]$ are used to control the objective function. As we aim to resolve the input neurons as well as the input lower and upper bounds of unstable ReLU neurons to refine the abstraction, we only set one element among $c^{(0)}, c^{(i)}, i \in [R]$ as 1 (for lower bound computation) or -1 (for upper bound) for the respective neuron, the rest of the elements are set as 0.

Eventually, we transform the constrained solving problem into an unconstrained one using Lagrangian variables as shown below, where we annotate $[x]_+ = \max(x, 0), [x]_- = -\min(x, 0)$:

$$
\begin{aligned}
&\max_{\gamma,v,\pi,\alpha} l^{(0)}[c^{(0)T} - v^{(1)T}w^{(1)}]_+ - u^{(0)}[v^{(1)T}w^{(1)} - c^{(0)T}]_+ + \gamma^T d \\
&+ \sum_{i\in[R]} \sum_{j\in I^{\pm(i)}} [\hat{v}^{(i)j}]_+ \cdot s^{(i)j} \cdot l^{(i)j} - \sum_{i\in[R]} \pi^{(i)T} p^{(i)} - \sum_{i\in[L_1]} v^{(i)T} b^{(i)} \\
&\text{s.t. } v^{(L)} = -H^T\gamma;\ \gamma, \pi \geq 0;\ \alpha \in [0,1] \\
&\text{for } i \in [L_1] \cup [L_2], i_s \in [R] \cap S(i), i_s \notin [L_2]: \\
&v^{(i)j} = -c^{(i_s)j}, j \in I^{-(i)} \\
&v^{(i)j} = \sum_{i_{s^2}\in S(i_s)\cap[L_1]} v^{(i_{s^2})T} W^{(i_{s^2})}_{:,j} - c^{(i_s)j}, j \in I^{+(i)} \\
&\text{for } j \in I^{\pm(i)}: v^{(i)j} = s^{(i_s)j}[\hat{v}^{(i_s)j}]_+ - c^{(i_s)j} - \pi^{(i_s)T} P^{(i_s)}_{:,j} - \alpha^{(i_s)j}[\hat{v}^{(i_s)j}]_- \\
&\hat{v}^{(i)j} = \sum_{i_{s^2}\in S(i_s)\cap[L_1]} v^{(i_{s^2})T} W^{(i_{s^2})}_{:,j} - \pi^{(i_s)T} \hat{P}^{(i_s)}_{:,j} \\
&\text{for } i \in [L_1] \text{ and } i_s \in [L_2] \cap S(i) \text{ and } i_s \notin [R]: \\
&v^{(i)} = v^{(i_s)}
\end{aligned}
\tag{2}
$$

Any valid setting of $\gamma, \pi \geq 0; \alpha \in [0,1]$ leads to a safe lower bound of the original problem. Based on the values of γ, π, α, we compute the values of $v^{(i)}$ and $\hat{v}^{(i)}$ in reverse order from $v^{(L)}$ to $v^{(0)}$. Using all assignments of variables, we could compute the objective value. In practice, the solving process starts with a valid initialization of γ, π, α, then we optimize these variables using gradient information.

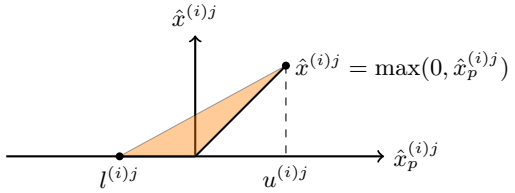


Fig. 6: The approximation of a ReLU neuron

Algorithm 1 Bounds tightening procedure

Input:
- M: neural network model
- $\mathcal{L}_L$: list of old lower bounds for all ReLU and input layers
- $\mathcal{L}_U$: list of old upper bounds for all ReLU and input layers
- Π: output constraints
- Θ: WraLU constraints

Output: improved lower and upper bounds

$\mathcal{S} \leftarrow$ CREATE_SOLVER_MODEL$(M, \mathcal{L}_L, \mathcal{L}_U, \Pi, \Theta)$
$list\_new\_L, list\_new\_U \leftarrow [], []$ ▷ initialization
for i **in** RANGE(len$(\mathcal{L}_L)$) **do** ▷ solve for each layer
  $\mathcal{S}$.SET_LAYER(i) ▷ reset to solve for this layer
  $\mathcal{S}$.INITALIZE_LAGRANGIAN_VARS()
  $max\_obj \leftarrow$ TRAIN_UNTIL_CONVERGENCE$(\mathcal{S})$
  $N_L, N_U \leftarrow$ GET_NEW_BOUNDS$(\mathcal{L}_L[i], \mathcal{L}_U[i], max_obj)$
    ▷ improve old bounds based on solved values
  $list\_new\_L$.APPEND(N_L) ▷ record updated bounds
  $list\_new\_U$.APPEND(N_U)
return $list\_new\_L, list\_new\_U$

Algorithm 1 shows the process of solving tighter bounds for each layer by training Lagrangian variables. While Lagrangian multipliers are commonly used in prior works [28,30,5,9], to the best of our knowledge, our method is the first to apply them to spurious-adversarial-label-guided refinement. Furthermore, we incorporate multi-neuron constraints, output constraints and L_2 layer constraints that explicitly consider two preceding layers, which enhances the theoretical rigor of residual network verification.

4 Experiments

We compare the performance of our prototypical verifier GRENA with SOTA verifiers including the incomplete tool WraLU [12] and the complete tool α, β-CROWN [3] - the winner of VNNCOMP (International Verification of Neural Networks Competition). In addition, we compare our tailored LP solver with SOTA GUROBI with respect to returned bound tightness and execution time.

4.1 Experiment Setup

The dataset includes MNIST (denoted as 'M') [2] and CIFAR10 (shortened as 'C') [10]. We test fully-connected (denoted as 'FC'), convolutional ('Conv') and residual ('Res') networks with various sizes, that are obtained from the ERAN system [4] and VNNCOMP [1]. The number of intermediate layers (#Layers), the number of intermediate neurons (#Neurons), and the trained defense are

enumerated in Table 1 (a trained defense is a defense method against adversarial examples to improve robustness of networks).

Table 1: Detailed information of the experimental networks

Network	**Type**	ϵ	**#Layers**	**#Neurons**	**Defense**
M_6x256	FC	0.033	6	1,010	None
M_ConvSmall	Conv	0.11	3	3,604	None
M_ConvMed	Conv	0.1	3	5,704	None
M_ConvBig	Conv	0.313	6	48,064	DiffAI[14]
C_ConvMed	Conv	0.006	3	7,144	PGD[13]
C_ConvBig	Conv	0.0078	6	62,464	DiffAI
C_Resnet4b	Res	0.0042	10	14,436	None
C_ResnetA	Res	0.0033	8	11,364	None
C_ResnetB	Res	0.012	8	11,364	None

4.2 Comparison with SoTA Verifiers

To test the verification performance of GRENA, we select 30 images from the datasets for each network to verify robustness and compare the results and time costs. To verify robustness, we choose a perturbation parameter ϵ for each tested network as indicated in Table 1 and apply the perturbation to each image. We check if all the "perturbed" images within ϵ will be classified the same as the original image by the networks as the perturbation is imperceptible to human eyes. If so, we conclude the robustness to be verified. Otherwise, if a counterexample with a different label is detected, we falsify the robustness property. If the analysis is inconclusive, we return *unknown* (abbreviated as '#Unk') to the user.

The verification results of each tool and average execution time per image are shown in Table 2. We can observe that we *outperforms both WraLU and* α, β*-CROWN* with respect to precision as we return more conclusive results (either verified or falsified). In particular, we return **50.7%** more conclusive images than WraLU while WraLU fails to handle two residual networks. Even compared with the complete tool α, β-CROWN, our tool produces **13** more conclusive images in total and achieve **better or the same** verification/falsification precision on most networks. The empirical results demonstrate *the strong verification efficiency of our system.*

4.3 Comparision with GUROBI

We now compare the bound-solving abilities of our tailored solver to those of GUROBI in the context of neural network encoding. We select one image for

Table 2: The verification results of WraLU, α, β-CROWN and our system GRENA with average execution time per image

Network	**Methods**	Verification results			
		#Unk	#Verify	#Falsify	Time(s)
M_6x256	WraLU	27	3	0	26.6
	α, β-CROWN	8	**12**	**10**	87.9
	GRENA	15	7	8	195.5
M_ConvSmall	WraLU	17	13	0	7.2
	α, β-CROWN	0	**26**	**4**	5.8
	GRENA	0	**26**	**4**	35.7
M_ConvMed	WraLU	15	15	0	18.0
	α, β-CROWN	2	22	**6**	28.6
	GRENA	0	**24**	**6**	42.9
M_ConvBig	WraLU	10	**20**	0	35.6
	α, β-CROWN	3	**20**	7	31.8
	GRENA	2	**20**	**8**	77.0
C_ConvMed	WraLU	19	11	0	60.8
	α, β-CROWN	8	17	5	84.8
	GRENA	0	**23**	**7**	119.9
C_ConvBig	WraLU	0	**30**	0	32.7
	α, β-CROWN	2	28	0	25.1
	GRENA	0	**30**	0	58.0
C_Resnet4b	WraLU	7	23	0	49.7
	α, β-CROWN	0	**26**	**4**	6.3
	GRENA	0	**26**	**4**	87.8
C_ResnetA	WraLU	-	-	-	-
	α, β-CROWN	0	**27**	**3**	10.2
	GRENA	0	**27**	**3**	77.6
C_ResnetB	WraLU	-	-	-	-
	α, β-CROWN	8	16	**6**	100.2
	GRENA	1	**23**	**6**	191.5

each network and collect all the constraints where we use the constraint set to solve all unstable neuron bounds and input bounds by our solver and GUROBI, later we compare the tightness of the solved bounds as visualized in Figure 7.

Figure 7 depicts log-scale histograms of bound improvements for both GUROBI and our tailored solver, where "improvement" is defined as the original neuron bound minus the new neuron interval returned by the two solvers. The bar heights represent the number of neurons with improvements at the magnitude indicated on the x-axis. Figure 7a, 7d, 7e and 7f show significant overlap between the orange and blue bars, meaning our tailored solver achieved **comparable** improvements to GUROBI. It is noteworthy that the average solving time

for GUROBI was 35503.32 seconds, while our GPU-accelerated solver took only 47.38 seconds, achieving an impressive **749**× speedup. More results and details can be found in our Github repo.

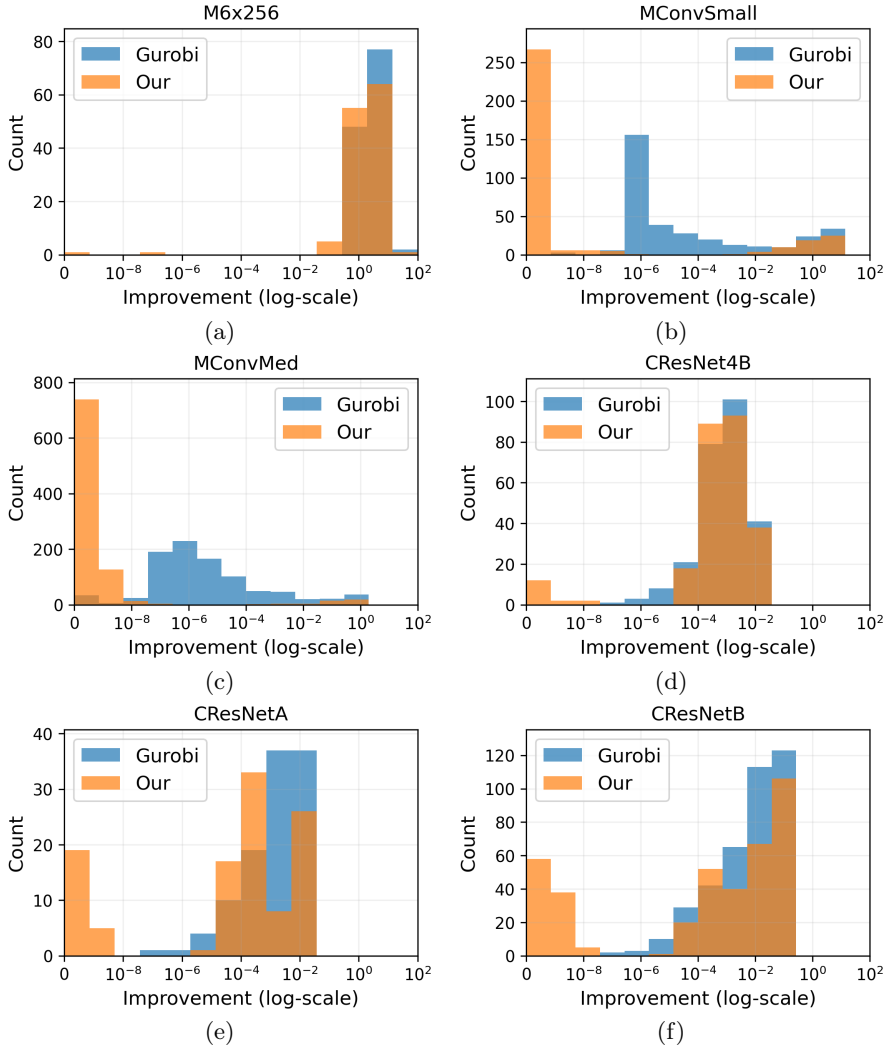


Fig. 7: Bound improvement comparison between our solver and GUROBI

In conclusion, our tailored LP solver *obtained comparable bound improvements* compared to GUROBI while *significantly reducing* the solving time.

5 Related Works

Generally speaking, verification of deep neural networks is an NP-hard problem [8]. Therefore, there are a series of incomplete verification methods that sacrifice completeness. Representative works include those abstract interpretation based [6,19,20,14] or bound propagation based [16,7,29,33,26], etc. To mitigate the precision loss of incomplete methods, researchers have been relying on LP or MILP to encode the network more tightly. For example, DeepSRGR [31] or ARENA [34] or PRIMA [17] systems would invoke the GUROBI solver to resolve LP and obtain tighter neuron intervals. However, the usage of an off-the-shelf solver on the CPU fails to leverage the nature of neural network encoding.

Inspired by works aiming to migrate the verification of neural networks to GPUs with the help of Lagrangian dual problems [28,5,9], we propose our tailored LP solver on GPU that benefits our LP formulation. Note that previous works [28,5,9] only encode one-predecessor cases where the multiple predecessors would be concatenated into one. Although this could be handled by other engineering approaches, it lacks rigorous theoretical derivation. On the contrary, we explicitly encode multi-predecessor cases in our formulation. Furthermore, [28,5] only considers intermediate neuron constraints and [9] only includes output constraints in their constraint set, while our formulation captures both intermediate and output constraints. Lastly, to our knowledge, our method is the first to effectively deploy the Lagrangian dual problem to spurious-adversarial-label-guided refinement process.

6 Conclusion

In this paper, we propose a theorem to solve LP problem on GPU in the context of neural network verification. To the best of our knowledge, our work is the first to use Lagrangian dual on spurious-adversarial-label guided refinement process and encode complex network constraints that take more than one predecessor, which enhances the scientific rigor of the verification of residual networks. We implemented our solving theorem in a GPU-based tailored solver; our empirical study strongly indicates that our tailored solver could return comparable solved values compared to GUROBI while obtaining significant speed gains. Furthermore, it enables our verifier GRENA to return more conclusive results than SOTA verifiers within a reasonable amount of time, demonstrating the strong efficacy of our system.

Acknowledgments. This research is supported by a Singapore Ministry of Education Academic Research Fund Tier 1 T1-251RES2103.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. 3rd International Verification of Neural Networks Competition (VNN-COMP'22) (2022), https://sites.google.com/view/vnn2022.
2. Deng, L.: The MNIST database of handwritten digit images for machine learning research [best of the web]. IEEE Signal Process. Mag. **29**(6), 141–142 (2012). https://doi.org/10.1109/MSP.2012.2211477, https://doi.org/10.1109/MSP.2012.2211477
3. etc, H.Z.: Alpha-beta-crown: A fast and scalable neural network verifier using the bound propagation framework (2025), https://github.com/Verified-Intelligence/alpha-beta-CROWN. Retrieved on Jan 4th, 2025
4. ETH: ETH Robustness Analyzer for Neural Networks (ERAN) (2025), https://github.com/eth-sri/eran. Retrieved on Jan 2nd, 2025
5. Ferrari, C., Müller, M.N., Jovanovic, N., Vechev, M.T.: Complete verification via multi-neuron relaxation guided branch-and-bound. In: The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022. OpenReview.net (2022), https://openreview.net/forum?id=l_amHf1oaK
6. Gehr, T., Mirman, M., Drachsler-Cohen, D., Tsankov, P., Chaudhuri, S., Vechev, M.T.: AI2: safety and robustness certification of neural networks with abstract interpretation. In: 2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA. pp. 3–18. IEEE Computer Society (2018). https://doi.org/10.1109/SP.2018.00058, https://doi.org/10.1109/SP.2018.00058
7. Gowal, S., Dvijotham, K., Stanforth, R., Bunel, R., Qin, C., Uesato, J., Arandjelovic, R., Mann, T.A., Kohli, P.: Scalable verified training for provably robust image classification. In: 2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019. pp. 4841–4850. IEEE (2019). https://doi.org/10.1109/ICCV.2019.00494, https://doi.org/10.1109/ICCV.2019.00494
8. Katz, G., Barrett, C.W., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: An efficient SMT solver for verifying deep neural networks. CoRR **abs/1702.01135** (2017), http://arxiv.org/abs/1702.01135
9. Kotha, S., Brix, C., Kolter, J.Z., Dvijotham, K., Zhang, H.: Provably bounding neural network preimages. In: Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., Levine, S. (eds.) Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023 (2023), http://papers.nips.cc/paper_files/paper/2023/hash/fe061ec0ae03c5cf5b5323a2b9121bfd-Abstract-Conference.html
10. Krizhevsky, A., Nair, V., Hinton, G.: Cifar-10/100 (canadian institute for advanced research) http://www.cs.toronto.edu/~kriz/cifar.html
11. LLC., G.O.: GUROBI OPTIMIZER (2025), https://www.gurobi.com/. Retrieved on Jan 1st, 2025
12. Ma, Z., Li, J., Bai, G.: Relu hull approximation. Proc. ACM Program. Lang. **8**(POPL), 2260–2287 (2024). https://doi.org/10.1145/3632917, https://doi.org/10.1145/3632917
13. Madry, A., Makelov, A., Schmidt, L., Tsipras, D., Vladu, A.: Towards deep learning models resistant to adversarial attacks. In: 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings. OpenReview.net (2018), https://openreview.net/forum?id=rJzIBfZAb

14. Mirman, M., Gehr, T., Vechev, M.T.: Differentiable abstract interpretation for provably robust neural networks. In: Dy, J.G., Krause, A. (eds.) Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018. Proceedings of Machine Learning Research, vol. 80, pp. 3575–3583. PMLR (2018), http://proceedings.mlr.press/v80/mirman18b.html
15. Müller, C., Serre, F., Singh, G., Püschel, M., Vechev, M.T.: Scaling polyhedral neural network verification on gpus. In: MLSys. mlsys.org (2021)
16. Müller, C., Singh, G., Püschel, M., Vechev, M.T.: Neural network robustness verification on gpus. CoRR **abs/2007.10868** (2020), https://arxiv.org/abs/2007.10868
17. Müller, M.N., Makarchuk, G., Singh, G., Püschel, M., Vechev, M.T.: PRIMA: general and precise neural network certification via scalable convex hull approximations. Proc. ACM Program. Lang. **6**(POPL), 1–33 (2022). https://doi.org/10.1145/3498704, https://doi.org/10.1145/3498704
18. Paulsen, B., Wang, J., Wang, C.: Reludiff: differential verification of deep neural networks. In: Rothermel, G., Bae, D. (eds.) ICSE '20: 42nd International Conference on Software Engineering, Seoul, South Korea, 2020. pp. 714–726. ACM (2020). https://doi.org/10.1145/3377811.3380337, https://doi.org/10.1145/3377811.3380337
19. Singh, G., Gehr, T., Mirman, M., Püschel, M., Vechev, M.T.: Fast and effective robustness certification. In: Bengio, S., Wallach, H.M., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (eds.) Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada. pp. 10825–10836 (2018), https://proceedings.neurips.cc/paper/2018/hash/f2f446980d8e971ef3da97af089481c3-Abstract.html
20. Singh, G., Gehr, T., Püschel, M., Vechev, M.T.: An abstract domain for certifying neural networks. Proc. ACM Program. Lang. **3**(POPL), 41:1–41:30 (2019). https://doi.org/10.1145/3290354, https://doi.org/10.1145/3290354
21. Singh, G., Gehr, T., Püschel, M., Vechev, M.T.: Boosting robustness certification of neural networks. In: 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019. OpenReview.net (2019), https://openreview.net/forum?id=HJgeEh09KQ
22. Tjandraatmadja, C., Anderson, R., Huchette, J., Ma, W., Patel, K.K., Vielma, J.P.: The convex relaxation barrier, revisited: Tightened single-neuron relaxations for neural network verification. Advances in Neural Information Processing Systems **33**, 21675–21686 (2020)
23. Ugare, S., Banerjee, D., Misailovic, S., Singh, G.: Incremental verification of neural networks. Proc. ACM Program. Lang. **7**(PLDI), 1920–1945 (2023). https://doi.org/10.1145/3591299, https://doi.org/10.1145/3591299
24. Ugare, S., Singh, G., Misailovic, S.: Proof transfer for fast certification of multiple approximate neural networks. Proc. ACM Program. Lang. **6**(OOPSLA1), 1–29 (2022). https://doi.org/10.1145/3527319, https://doi.org/10.1145/3527319
25. Urban, C., Christakis, M., Wüstholz, V., Zhang, F.: Perfectly parallel fairness certification of neural networks. Proc. ACM Program. Lang. **4**(OOPSLA), 185:1–185:30 (2020). https://doi.org/10.1145/3428253, https://doi.org/10.1145/3428253
26. Wang, S., Pei, K., Whitehouse, J., Yang, J., Jana, S.: Efficient formal safety analysis of neural networks. In: Bengio, S., Wallach, H.M., Larochelle, H.,

Grauman, K., Cesa-Bianchi, N., Garnett, R. (eds.) Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada. pp. 6369–6379 (2018), https://proceedings.neurips.cc/paper/2018/hash/2ecd2bd94734e5dd392d8678bc64cdab-Abstract.html

27. Wang, S., Pei, K., Whitehouse, J., Yang, J., Jana, S.: Formal security analysis of neural networks using symbolic intervals. In: Enck, W., Felt, A.P. (eds.) 27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018. USENIX Association (2018), https://www.usenix.org/conference/usenixsecurity18/presentation/wang-shiqi
28. Wang, S., Zhang, H., Xu, K., Lin, X., Jana, S., Hsieh, C., Kolter, J.Z.: Beta-crown: Efficient bound propagation with per-neuron split constraints for complete and incomplete neural network verification. CoRR **abs/2103.06624** (2021), https://arxiv.org/abs/2103.06624
29. Weng, T., Zhang, H., Chen, H., Song, Z., Hsieh, C., Daniel, L., Boning, D.S., Dhillon, I.S.: Towards fast computation of certified robustness for relu networks. In: Dy, J.G., Krause, A. (eds.) Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018. Proceedings of Machine Learning Research, vol. 80, pp. 5273–5282. PMLR (2018), http://proceedings.mlr.press/v80/weng18a.html
30. Xu, K., Shi, Z., Zhang, H., Wang, Y., Chang, K., Huang, M., Kailkhura, B., Lin, X., Hsieh, C.: Automatic perturbation analysis for scalable certified robustness and beyond. In: Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., Lin, H. (eds.) Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual (2020), https://proceedings.neurips.cc/paper/2020/hash/0cbc5671ae26f67871cb914d81ef8fc1-Abstract.html
31. Yang, P., Li, R., Li, J., Huang, C., Wang, J., Sun, J., Xue, B., Zhang, L.: Improving neural network verification through spurious region guided refinement. In: Groote, J.F., Larsen, K.G. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 27th International Conference, TACAS 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27 - April 1, 2021, Proceedings, Part I. Lecture Notes in Computer Science, vol. 12651, pp. 389–408. Springer (2021). https://doi.org/10.1007/978-3-030-72016-2_21, https://doi.org/10.1007/978-3-030-72016-2_21
32. Zelazny, T., Wu, H., Barrett, C.W., Katz, G.: On optimizing back-substitution methods for neural network verification. In: Griggio, A., Rungta, N. (eds.) 22nd Formal Methods in Computer-Aided Design, FMCAD 2022, Trento, Italy, October 17-21, 2022. pp. 17–26. IEEE (2022). https://doi.org/10.34727/2022/ISBN.978-3-85448-053-2_7, https://doi.org/10.34727/2022/isbn.978-3-85448-053-2_7
33. Zhang, H., Weng, T., Chen, P., Hsieh, C., Daniel, L.: Efficient neural network robustness certification with general activation functions. In: Bengio, S., Wallach, H.M., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (eds.) Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada. pp. 4944–4953 (2018), https://proceedings.neurips.cc/paper/2018/hash/d04863f100d59b3eb688a11f95b0ae60-Abstract.html

34. Zhong, Y., Ta, Q., Khoo, S.: ARENA: enhancing abstract refinement for neural network verification. In: Dragoi, C., Emmi, M., Wang, J. (eds.) Verification, Model Checking, and Abstract Interpretation - 24th International Conference, VMCAI 2023, Boston, MA, USA, January 16-17, 2023, Proceedings. Lecture Notes in Computer Science, vol. 13881, pp. 366–388. Springer (2023). https://doi.org/10.1007/978-3-031-24950-1_17, https://doi.org/10.1007/978-3-031-24950-1_17