Efficient Neural Network Verification via Order Leading Exploration of Branch-and-Bound Trees

Guanqin Zhang[1,2], Kota Fukuda[3], Zhenya Zhang[3], Dilum Bandara[1,2], Shiping Chen[1,2], Jianjun Zhao[3], Yulei Sui[1]

[1]University of New South Wales, Sydney, Australia
[2]CSIRO's Data61, Australia
[3]Kyushu University, Fukuoka, Japan

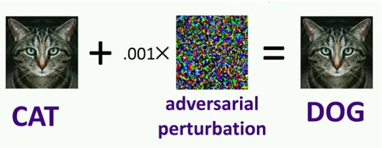
Contribution

We propose Oliva , a novel framework that accelerates **branch-and-bound** neural network verification by incorporating an **order leading exploration** of sub-problems, characterized by:

- A **greedy order** to prioritize sub-problems likely to contain true counterexamples (Oliva^{GR}).
- A **simulated annealing-inspired** approach for **stochastically balancing** exploration and exploitation (Oliva^{SA}).

This framework significantly accelerates the verification process by effectively **navigating** the sub-problem space, and **falsifying** the verification task.

Are Neural Networks Robustness?



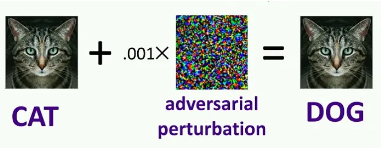
[1]

[2]

[1] Explaining and Harnessing Adversarial Examples

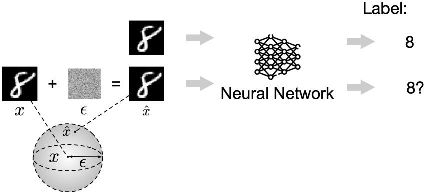
[2] Optical Adversarial Attack.

Are Neural Networks Robustness?



[1]

[2]



Given an original input x, and its perturbed input $\hat{x} \in \Phi : \{\hat{x} \mid \|\hat{x} - x\|_\infty \leq \epsilon\}$, the robustness verification of an NN $f(\cdot)$ is to check $\forall \hat{x} \in \Phi$ such that $f(\hat{x}) \models \Psi$, where Φ is the input property and Ψ is the output property.

Small perturbations on the input can **fool** neural networks to yield **incorrect output**.

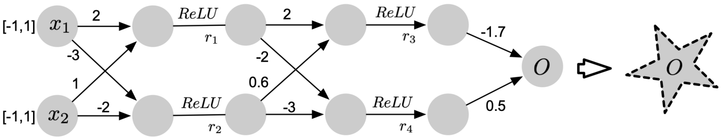
[1] Explaining and Harnessing Adversarial Examples

[2] Optical Adversarial Attack.

Over-Approximation for Neural Network Verification

Specification: $\Phi \wedge \Psi$ **Input:** $\Phi := x_1 \in [-1,1] \wedge x_2 \in [-1,1]$ **Output:** $\Psi = (O > 0)$

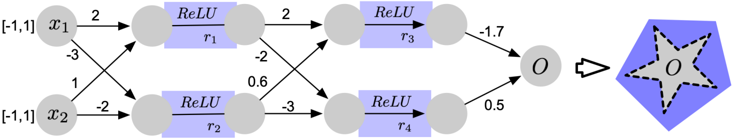
Network: f



Over-Approximation for Neural Network Verification

Specification: $\Phi \wedge \Psi$ Input: $\Phi := x_1 \in [-1,1] \wedge x_2 \in [-1,1]$ Output: $\Psi = (O > 0)$

Network: f



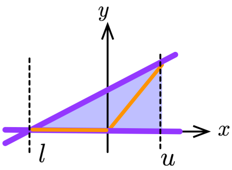


Figure: DeepPoly

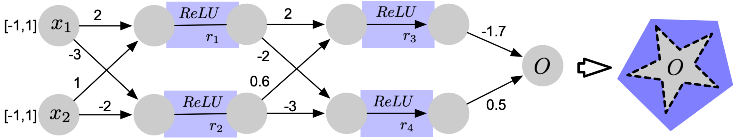
[3] Lowerbound $\hat{p}$: $\min O = \texttt{LPSolver}(\Phi \wedge f \wedge \Psi)$
$\hat{p} = -\mathbf{2.7}$ obtained by conservative over-approximation of active functions (i.e., ReLU) via linear solver and cannot verify the specification.

[3] An Abstract Domain for Certifying Neural Networks

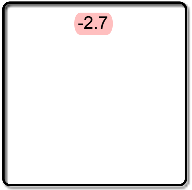
Specification Distance

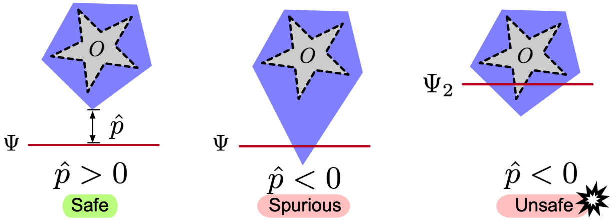
Specification: $\Phi \wedge \Psi$ Input: $\Phi := x_1 \in [-1, 1] \wedge x_2 \in [-1, 1]$ Output: $\Psi = (O > 0)$

Network: f



$\hat{p} =$ **-2.7** is the **specification distance** of the overapproximation.

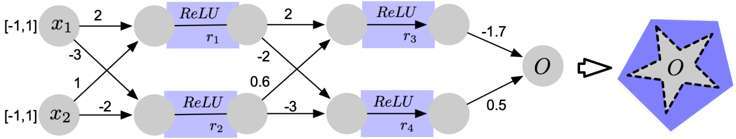


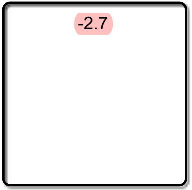


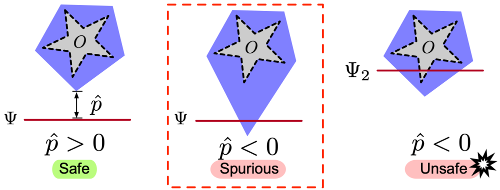
Spurious Counterexample

Specification: $\Phi \wedge \Psi$ Input: $\Phi := x_1 \in [-1,1] \wedge x_2 \in [-1,1]$ Output: $\Psi = (O > 0)$

Network: f





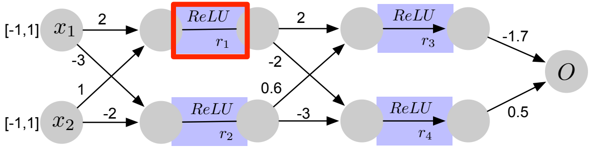


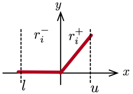
Now, it cannot verify the specification.
Branch-and-bound specification tree splitting is needed!

Specification Tree

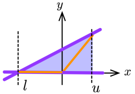
Specification: $\Phi \wedge \Psi$ Input: $\Phi := x_1 \in [-1,1] \wedge x_2 \in [-1,1]$ Output: $\Psi = (O > 0)$

Network: f





Precise spliting ReLU r_1

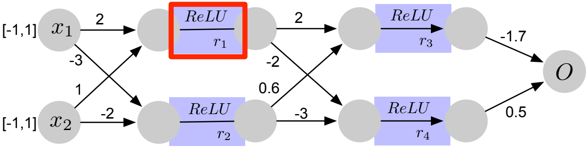


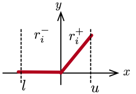
Over-approximations for ReLUs r_2 r_3 r_4

Specification Tree

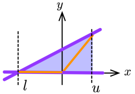
Specification: $\Phi \wedge \Psi$ Input: $\Phi := x_1 \in [-1,1] \wedge x_2 \in [-1,1]$ Output: $\Psi = (O > 0)$

Network: f





Precise spliting ReLU r_1



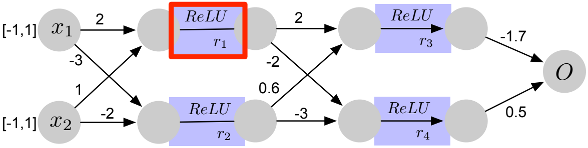
Over-approximations for ReLUs r_2 r_3 r_4

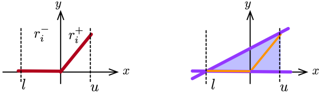
ReLU r_1 is selected to be split. An *atomic proposition* with respect to the selected r_1 neuron is enforced as either $\mathtt{x}_1 \geq 0$ (denoted r_i^+) or $\mathtt{x}_1 < 0$ (denoted r_i^-).

Specification Tree

Specification: $\Phi \wedge \Psi$ Input: $\Phi := x_1 \in [-1,1] \wedge x_2 \in [-1,1]$ Output: $\Psi = (O > 0)$

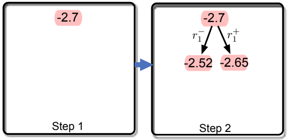
Network: f





Precise spliting ReLU r_1 **Over-approximations for ReLUs** r_2 r_3 r_4

ReLU r_1 is selected to be split. An *atomic proposition* with respect to the selected r_1 neuron is enforced as either $\mathtt{x}_1 \geq 0$ (denoted r_i^+) or $\mathtt{x}_1 < 0$ (denoted r_i^-).

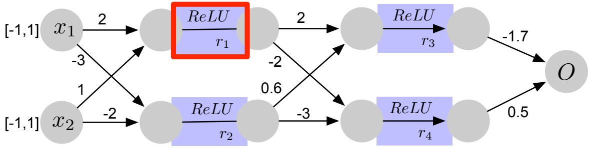


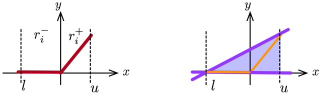
Specification Tree

Specification Tree

Specification: $\Phi \wedge \Psi$ Input: $\Phi := x_1 \in [-1,1] \wedge x_2 \in [-1,1]$ Output: $\Psi = (O > 0)$

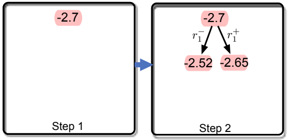
Network: f





Precise spliting ReLU r_1 **Over-approximations for ReLUs** r_2 r_3 r_4

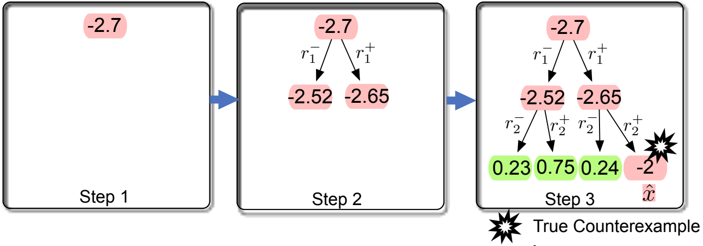
ReLU r_1 is selected to be split. An *atomic proposition* with respect to the selected r_1 neuron is enforced as either $x_1 \geq 0$ (denoted r_i^+) or $x_1 < 0$ (denoted r_i^-).



Specification Tree

A specification tree $\mathcal{T}$ consists **nodes** as tuples $\langle \Gamma, \hat{p} \rangle$ that record the sub-problems explored. Γ records the split subproblems with the enforced ReLU conditions. Subproblem with the split ReLU (r_1^+) is computed by $\hat{p} =$ $\texttt{LPSolver}(\Phi \wedge f \wedge \Psi \wedge r_1^+)$, while the rest of the ReLUs are kept approximated.

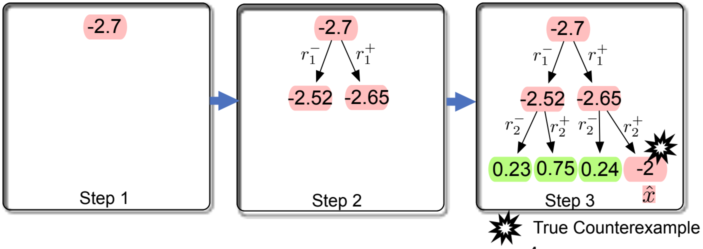
Existing Approach



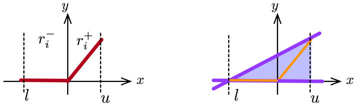
Tree grows when continuing to spit ReLU r_2 [4].

[4] Branch and Bound for Piecewise Linear Neural Network Verification, J. of Machine Learning Research 2020.

Existing Approach



Tree grows when continuing to spit ReLU r_2 [4].

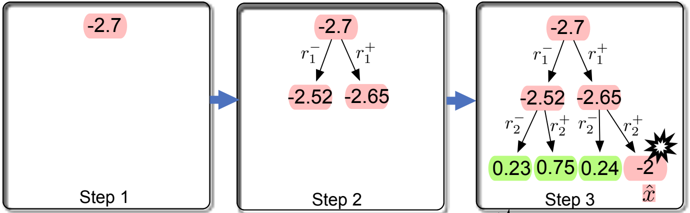


Precise spliting ReLU r_1 r_2 **Over-approximations for ReLUs** r_3 r_4

Keep branching ReLU r_2, while ReLUs r_3 and r_4 are kept over-approximation, we find the true counterexample in the end.

[4] Branch and Bound for Piecewise Linear Neural Network Verification, J. of Machine Learning Research 2020.

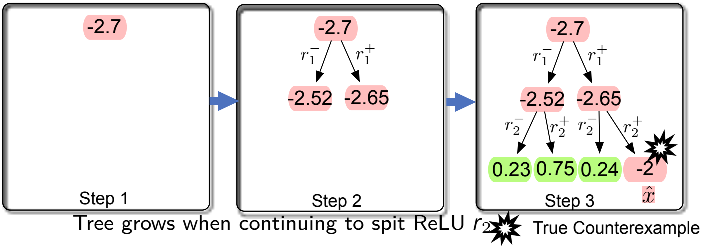
Our Insights



Tree grows when continuing to spit ReLU r_2

True Counterexample

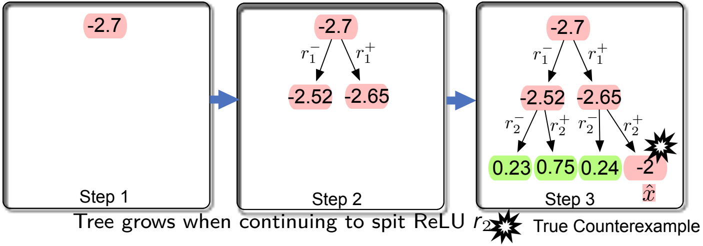
Our Insights



From our insights, **counterexamples** are related to two factors:

- The nodes (specification distance $\hat{p}$) with smaller values are more likely to have a true counterexample.
- The node's depth measures the over-approximation level ($|\Gamma|$): deeper with less approximation

Our Insights

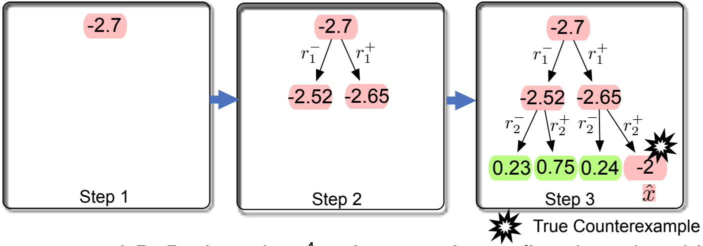


From our insights, **counterexamples** are related to two factors:

- The nodes (specification distance $\hat{p}$) with smaller values are more likely to have a true counterexample.
- The node's depth measures the over-approximation level ($|\Gamma|$): deeper with less approximation

If we find a true **counterexample** in the subproblem, it is also a **counterexample** for the root problem (step 1).

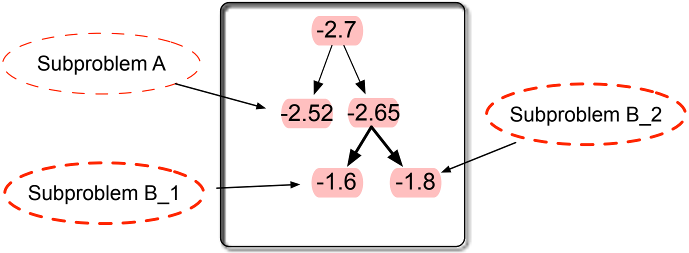
Limitations of existing approaches and our contribution



- The conventional BaB algorithm [4] **exhaustively** verifies the subproblems (each leaf node of BaB tree), which is **inefficient** in discovering counterexamples.
- Our approach explores the **order** of the subproblems to quickly identify true counterexamples (i.e., **eagerly falsify** the verification instance based on the counterexample potential of the subproblem spaces).

[4] Branch and bound for piecewise linear neural network verification. J. of Machine Learning Research 2020.

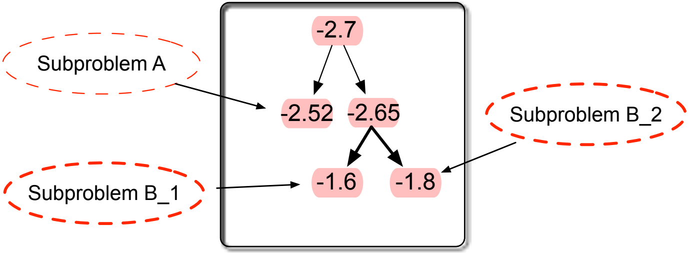
Contribution I: Inferring the Potentiality of Counterexample



Contribution:
Order-leading verification of subproblems based on counterexample potentiality

$$[\![\Gamma]\!] = \begin{cases} -\infty & \textit{if } \hat{p} > 0 \\ +\infty & \textit{true CE} \\ \lambda \frac{|\Gamma|}{K} + (1-\lambda)\frac{\hat{p}}{\hat{p}_{min}} & \textit{otherwise} \end{cases}$$

Contribution I: Inferring the Potentiality of Counterexample

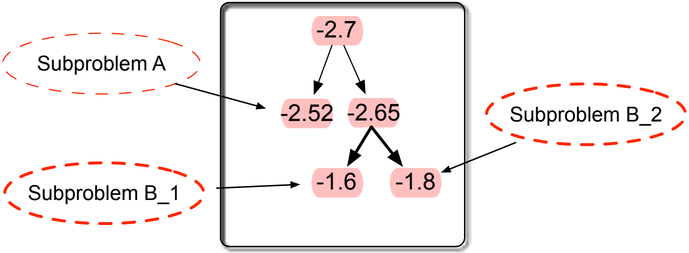


Contribution:
Order-leading verification of subproblems based on counterexample potentiality

$$[\![\Gamma]\!] = \begin{cases} -\infty & \textit{if } \hat{p} > 0 \\ +\infty & \textit{true CE} \\ \lambda \frac{|\Gamma|}{K} + (1-\lambda)\frac{\hat{p}}{\hat{p}_{min}} & \textit{otherwise} \end{cases}$$

$[\![\Gamma_A]\!] = 0.7$
$[\![\Gamma_{B\_1}]\!] = 0.9$
$[\![\Gamma_{B\_2}]\!] = 1.1$
so Γ_{B_2} is selected

Contribution I: Inferring the Potentiality of Counterexample



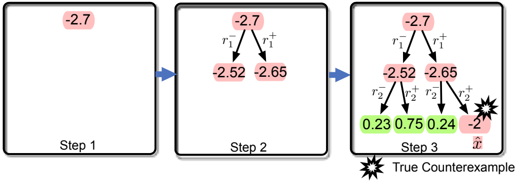
Contribution:
Order-leading verification of subproblems based on counterexample potentiality

$$[\![\Gamma]\!] = \begin{cases} -\infty & \textit{if } \hat{p} > 0 \\ +\infty & \textit{true CE} \\ \lambda \frac{|\Gamma|}{K} + (1-\lambda)\frac{\hat{p}}{\hat{p}_{min}} & \textit{otherwise} \end{cases}$$

$[\![\Gamma_A]\!] = 0.7$
$[\![\Gamma_{B\_1}]\!] = 0.9$
$[\![\Gamma_{B\_2}]\!] = 1.1$
so Γ_{B_2} is selected

- The nodes ($[\![\Gamma_i]\!]$) with larger **rewards** (of counterexample potentiality) are more likely to have a true counterexample.

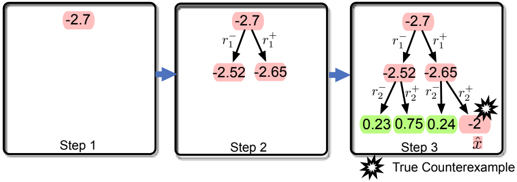
Order Leading Verification Approach (Oliva^{GR})



The conventional BaB algorithm manages "first-come, first-served" processing order, using

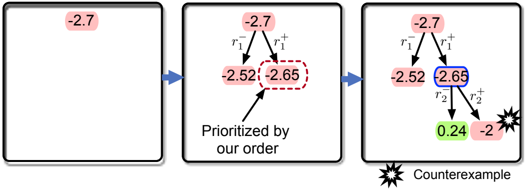
- **7** times of bounding, and **2** of branching (only two ReLUs) to find the couterexample.

Order Leading Verification Approach (Oliva^{GR})



The conventional BaB algorithm manages "first-come, first-served" processing order, using

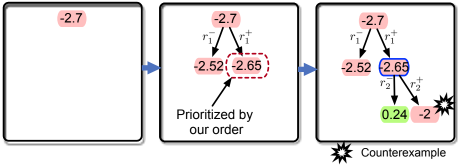
- **7** times of bounding, and **2** of branching (only two ReLUs) to find the couterexample.

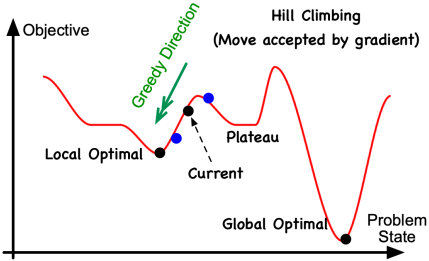


Oliva^{GR} considers the order of the subproblems (subtrees). Hence, it can speed up the verification process and use

- **5** times of bounding, and **2** times of bounding to find the couterexample.

Contribution II: Simulated Annealing (Oliva^{SA})

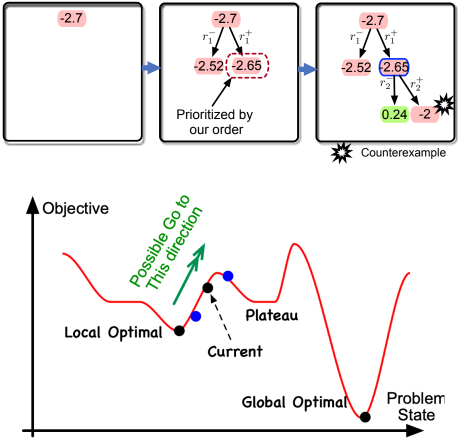




Greedily guided by **counterexample potentiality order** may:

- Trapped in **local optima** (similar to "Hill Climbing").
- unable to achieve **further** improvement.

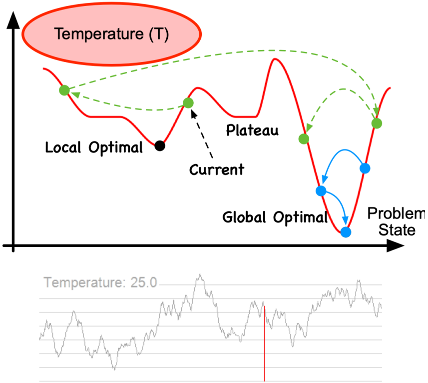
Contribution II: Simulated Annealing (Oliva^{SA})



Greedily guided by **counterexample potentiality order** may:

- Trapped in **local optima** (similar to "Hill Climbing").
- unable to achieve **further** improvement.

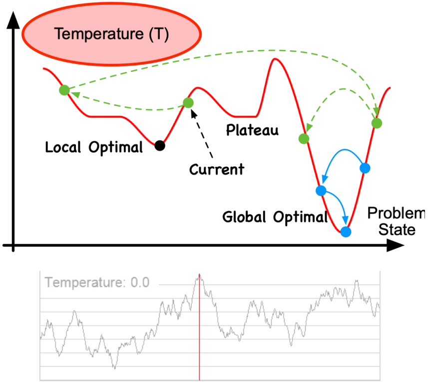
Contribution II: Simulated Annealing (Oliva^{SA})



Oliva^{SA} with simulated annealing can:

- Possible escape **local optima** and occasionally accept worse solution.
- **Temperature** controlled to achieve **further** improvement by balancing **exploration** and **exploitation**.
- Slowly converge to **exploitation**.

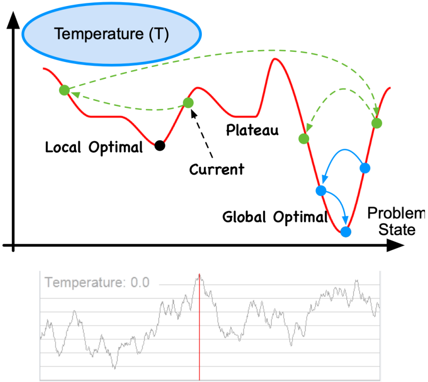
Contribution II: Simulated Annealing (Oliva^{SA})



Oliva^{SA} with simulated annealing can:

- Possible escape **local optima** and occasionally accept worse solution.
- **Temperature** controlled to achieve **further** improvement by balancing **exploration** and **exploitation**.
- Slowly converge to **exploitation**.

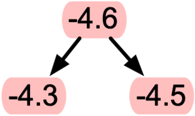
Contribution II: Simulated Annealing (Oliva^{SA})



Oliva^{SA} with simulated annealing can:

- Possible escape **local optima** and occasionally accept worse solution.
- **Temperature** controlled to achieve **further** improvement by balancing **exploration** and **exploitation**.
- Slowly converge to **exploitation**.

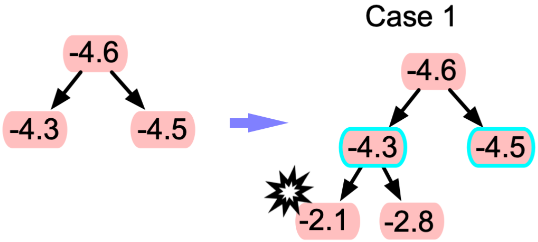
Oliva^{SA} Child Selection Policy



$$\Delta p \leftarrow \exp\left(\frac{\min \mathsf{R}(\Gamma \cdot a) - \max \mathsf{R}(\Gamma \cdot a)}{T}\right) \quad \text{s.t. } a \in \{r_k^+, r_k^-\}$$

$$\Gamma^* \leftarrow \Gamma \cdot a^* \quad \text{s.t. } a^* \leftarrow \begin{cases} \text{randomly choose } r_k^+ \text{ or } r_k^- & \text{if } \mathtt{rand}(0,1) < \Delta p \\ \underset{a \in \{r_k^+, r_k^-\}}{\arg\max}\ \mathsf{R}(\Gamma \cdot a) & \text{otherwise} \end{cases}$$

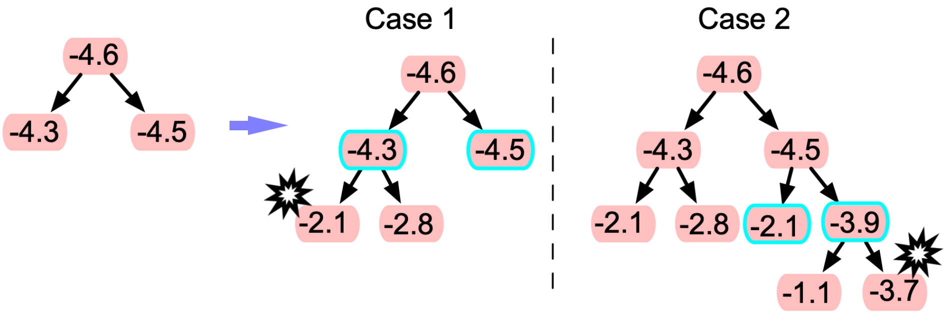
Oliva^{SA} Child Selection Policy



$$\Delta p \leftarrow \exp\left(\frac{\min \mathsf{R}(\Gamma \cdot a) - \max \mathsf{R}(\Gamma \cdot a)}{T\uparrow}\right) \quad \text{s.t.} \quad a \in \{r_k^+, r_k^-\}$$

$$\Gamma^* \leftarrow \Gamma \cdot a^* \quad \text{s.t.} \quad a^* \leftarrow \begin{cases} \boxed{\text{randomly choose } r_k^+ \text{ or } r_k^- \quad \text{if } \mathtt{rand}(0,1) < \Delta p \downarrow} \\ \arg\max\limits_{a \in \{r_k^+, r_k^-\}} \mathsf{R}(\Gamma \cdot a) \quad \text{otherwise} \end{cases}$$

Oliva^{SA} Child Selection Policy

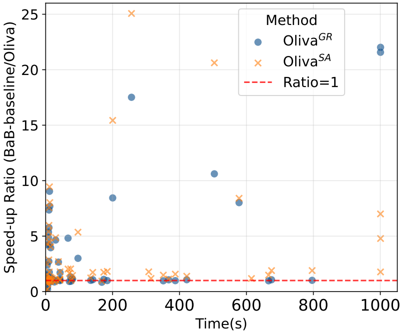


$$\Delta p \leftarrow \exp\left(\frac{\min \mathsf{R}(\Gamma \cdot a) - \max \mathsf{R}(\Gamma \cdot a)}{T\downarrow}\right) \quad \text{s.t. } a \in \{r_k^+, r_k^-\}$$

$$\Gamma^* \leftarrow \Gamma \cdot a^* \quad \text{s.t. } a^* \leftarrow \begin{cases} \text{randomly choose } r_k^+ \text{ or } r_k^- & \text{if } \texttt{rand}(0,1) < \Delta p \uparrow \\ \underset{a \in \{r_k^+, r_k^-\}}{\arg\max}\ \mathsf{R}(\Gamma \cdot a) & \text{otherwise} \end{cases}$$

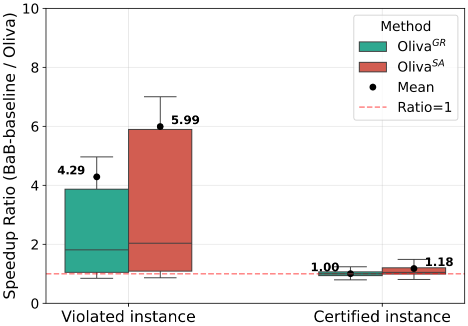
Order Leading Verification Approach (Oliva)

Experiment Results with model MNIST-L2 by 241 problem instances



Each point is a verification problem:

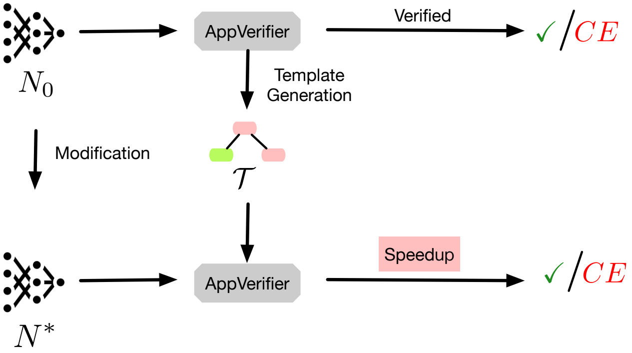
- x-axis: time costs by BaB-baseline
- y-axix: our speedup over BaB-baseline



Distribution of our speedup for violated instances and for certified instances.

Future Work

Order-leading **Incremental** Neural Network Verification - OOPSLA'25



Q&A

Code:
https://github.com/DeepLearningVerification/Oliva

Paper

Reference

to be added