The Marabou Framework for Verification and Analysis of Deep Neural Networks

September 9, 2022

目录

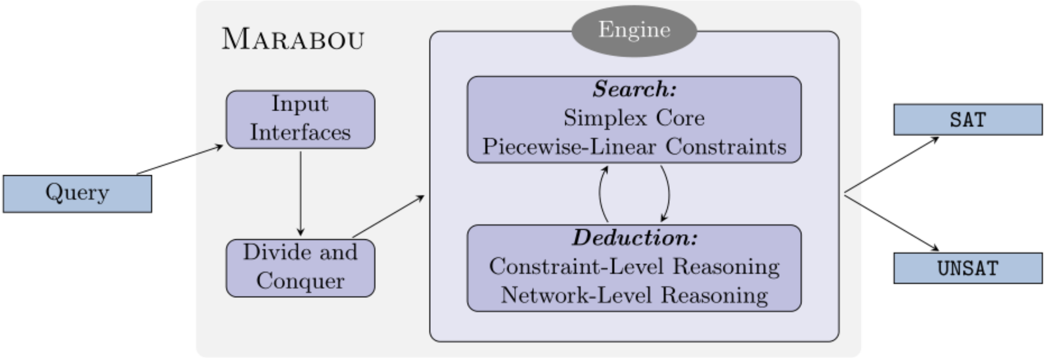
目录

Contribution

- Marabou is an SMT-based tool that can answer queries about a network's properties by transforming these queries into constraint satisfaction problems.
- The Marabou framework is a significant improvement over its predecessor, Reluplex. Specifically, it includes the following enhancements and modifications:
  - support for CNN
  - support for divide-and-conquer solving mode
  - a simplex-based linear programming core that replaces the external solver (GLPK)
  - Multiple interfaces for feeding queries into the solver
  - Support for network-level reasoning and deduction.

目录

Design of Marabou



Simplex Core

- Solve the linear constraints
- Reluplex: solve linear constraints by GLPK. $\rightarrow$ Marabou: implement a new custom solver
  - repeated translation of queries into GLPK and extraction of results from GLPK was a limiting factor on performance
  - black box simplex solver did not afford the flexibility we needed in the context of DNN verification

Piecewise-Linear Constraints

- configuration:
  - abstract class: **PiecewiseLinearConstraint class**
  - class: **Max** and **Relu**
  - objects: constraints
- methods of PiecewiseLinearConstraint class:
  - *satisfied()*
  - *getPossibleFixes()*
  - *getCaseSplits()*:piecewise-linear constraint $\varphi \rightarrow c_1 \vee \ldots \vee c_n$
  - *getEntailedTightenings()*: query the constraint for tighter variable bounds

Constraint and Network-Level Reasoning

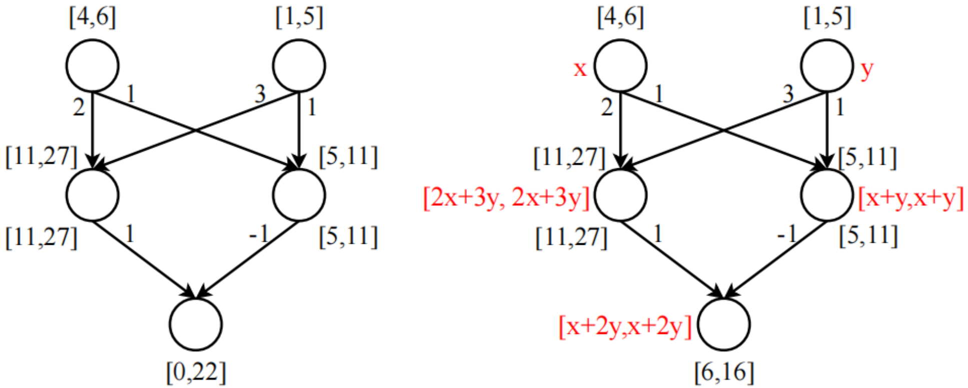
Constraint-level bound tightening: using *getEntailedTightenings()*

$$\text{deriveLowerBound}\ \frac{x_i \in \mathcal{B},\quad l(x_i) < \sum_{x_j \in \text{pos}(x_i)} T_{i,j} \cdot l(x_j) + \sum_{x_j \in \text{neg}(x_i)} T_{i,j} \cdot u(x_j)}{l(x_i) := \sum_{x_j \in \text{pos}(x_i)} T_{i,j} \cdot l(x_j) + \sum_{x_j \in \text{neg}(x_i)} T_{i,j} \cdot u(x_j)}$$

$$\text{deriveUpperBound}\ \frac{x_i \in \mathcal{B},\quad u(x_i) > \sum_{x_j \in \text{pos}(x_i)} T_{i,j} \cdot u(x_j) + \sum_{x_j \in \text{neg}(x_i)} T_{i,j} \cdot l(x_j)}{u(x_i) := \sum_{x_j \in \text{pos}(x_i)} T_{i,j} \cdot u(x_j) + \sum_{x_j \in \text{neg}(x_i)} T_{i,j} \cdot l(x_j)}$$

Constraint and Network-Level Reasoning

DNN-level reasoning: Symbolic interval propagation



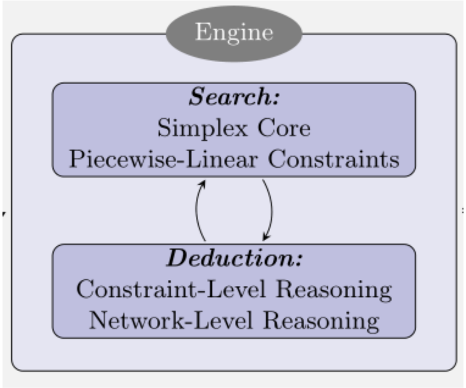
(a) Naive interval propagation (b) Symbolic interval propagation

Constraint and Network-Level Reasoning

DNN-level reasoning:symbolic bound tightening procedure

- Initializing the DNN-level reasoners with the most up-to-date information discovered during the search, such as variable bounds and the state of piecewise-linear constraints
- Feeding any new information that is discovered back into the search procedure.

Engine



- If there is a violated linear constraint, perform a simplex step.
- If there is a violated piecewise-linear constraint, attempt to fix it.
- If a piecewise-linear constraint had to be fixed more than a certain number of times, perform a case split on that constraint.
- If the problem has become unsatisfiable, undo a previous case split (or return UNSAT if no such case split exists).
- Return SAT (all constraints are satisfied).
- Deduction: heuristics

The Divide-and-Conquer Mode and Concurrency

Given a query ϕ, the solver maintains a queue Q of $\langle$query, timeout$\rangle$ pairs. Q is initialized with one element $\langle \phi, T \rangle$, where T, the initial timeout, is a configurable parameter. To solve ϕ, the solver loops through the following steps:

1. Pop a pair $\langle \phi', t' \rangle$ from Q and attempt to solve ϕ' with a timeout of t'.
2. If the problem is `UNSAT` and Q is empty, return `UNSAT`.
3. If the problem is `UNSAT` and Q is not empty, return to step 1.
4. If the problem is `SAT`, return `SAT`.
5. If a timeout occurred, split ϕ' into k sub-queries $\phi'_1, \ldots, \phi'_k$ by partitioning its input region. For each sub-query ϕ'_i, push $\langle \phi'_i, m \cdot t' \rangle$ into Q.

Thank you