Practical Analysis of Neural Networks Using Constraints Solving

Tewodros A. Beyene Amit Sahu
fortiss - Research Institute of the Free State of Bavaria
Guerickestraße 25, 80805 München, Germany
{beyene,sahu}@fortiss.org

The use of machine learning in safety-critical systems is currently demanding tools and methods for assuring the safety of these systems. In response to this demand, there have been recent efforts to extend traditional safety assurance approaches to systems with Learning-Enabled Components (LECs). In this paper, we present a deductive approach based on Constrained Horn Clauses (CHCs) for verifying and explaining neural networks and their decisions. In addition, CHCs are also proposed as evidence language for specifying neural networks in their safety certification process. We illustrate our approach on the CARLA simulator using a set of scenario from the autonomous driving domain.

1 Introduction

The employment of new capabilities such as machine learning in safety-critical systems is currently demanding tools, methods, and techniques for assuring the safety and dependability of these systems. In response to this demand, there have been recent efforts to extend standard safety assurance approaches, e.g., for traditional software, to systems with LECs. These efforts target two key challenges. The first challenge is developing efficient analysis, testing, and verification methods that are the ultimate sources of safety evidence, on which safety assurance arguments can be built. For example, state-of-the-art verification methods like Reluplex [10] use modified SMT solvers to handle the non-convex ReLU function by leveraging the piecewise linear nature of ReLUs. The efficiency of this method highly depends on the number of neurons since the ReLUs are applied at each neuron. For instance, a network of 300 ReLU nodes takes around 50 thousand seconds to verify a property! This is computationally unscalable given neural networks in real-world applications can have hundreds of layers and millions of activations per layer. The same is true for other solutions summarized in [1]. The second challenge is finding evidence artefacts that contribute to developing confidence in the safe operation of the LECs [4].

CHCs are logical implications involving undefined predicates. Methods based on these clauses have enjoyed tremendous success in performing various analysis tasks, such as verification of temporal properties and program termination, refinement type checking, and rely-guarantee reasoning, over traditional software [7]. These methods use CHCs to encode verification conditions that contain undefined auxiliary predicates that must be inferred.

In this paper, we propose an approach based on CHCs to tackle the two aforementioned challenges. First, given an NN and a safety property, CHCs are used for specifying verification conditions for the property as well as for specifying explanation about decisions and decision boundaries of the NN. Second, the set of CHCs specifying the NN can be further used as evidence artefacts in safety certification.

Submitted to:
HCVS 2022

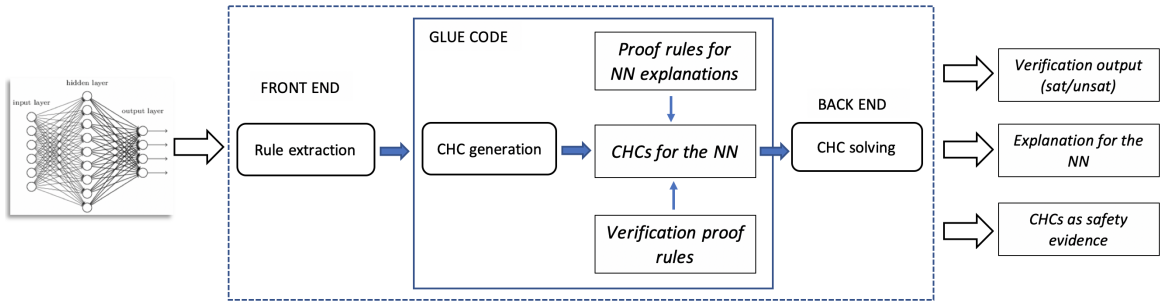


Figure 1: CHC-based approach for NN verification and explanation

2 The constraints-based approach

The task of developing CHC-based verification methods have become a lot simpler with the increasing availability of off-the-shelf constraints generation front-ends and back-end solvers. The increasing availability of such tools has reduced this task to the task of picking the appropriate front- and back-ends together with implementing a glue code to connect them [8].

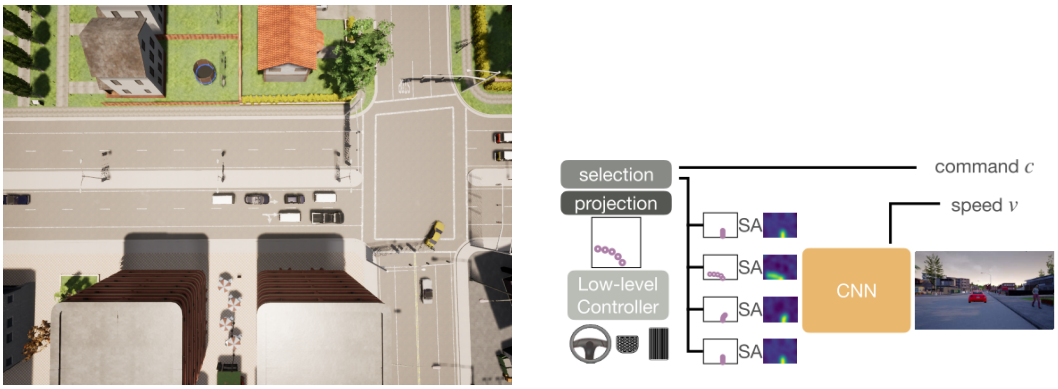
Inspired by this success, our approach consists of three components: *front end*, *glue code*, *back end*. A high-level depiction of the approach is shown in Fig. 1. The front end employs off-the-shelf rule extraction tools to extract rules that concretely specify the behaviour of the input neural network [9]. The glue code takes the extracted rules and translates them into a set of CHCs. Implemented in Java, this component performs a top-down recursive descent through each rule to introduce auxiliary predicates and generate a set CHCs over these predicates. The back end employs an off-the-shelf CHC solver to get a set of logical constraints for the unknown predicates [3]. These results correspond to the final verification outputs or explanations of the neural network.

Verification proof rules - The approach uses proof rules for solving turn-based graph games over infinite-state symbolic transition systems [2]. Such two-player games can model learning-enabled systems as they can explicitly specify interactions between the learning-enabled component (LEC) (first player) and its environment (second player). By encoding the systems to verify as two-player games with symbolic transition systems, these proof rules can be used for verifying both safety and liveness properties.

Proof rules for generating explanations - An explanation $\varphi_c(v)$ for a given class c is defined as a first-order formula in DNF that describes the decision boundary for the class. The formula should: (1) over-approximate all inputs for the given class, i.e., $\forall\, t_i = (x_i(v), c)$ then $x_i(v) \rightarrow \varphi_c(v)$ holds, and (2) not overlap with any other class, i.e., $\forall\, t_j = (x_j(v), c_j)$ such that $c_j \neq c$ then $\varphi_c(v) \wedge x_j(v) \rightarrow \bot$ holds. These conditions can be specified as a set of CHCs, and solving for the unknown $\varphi_c(v)$ amounts to finding an explanation for the decision boundary of class c.

3 Illustration

The approach is illustrated using a sensorimotor agent on the CARLA simulator [6]. The agent is a vision-based autonomous system that sees the world through an RGB image from the front-facing camera and predicts waypoints for the vehicle to steer towards. These waypoints are translated by a low-level controller into driving commands.



(a) CARLA simulation of the agent movement. (b) Sensorimotor Agent

Figure 2: A CARLA simulator scenario of agent movement.

The agent gets the position of the ego vehicle in coordinates x, y, the coordinates of other vehicles (in the vicinity) x_i, y_i where $0 \leq i \leq 4$, current speed s and current movement direction $\hat{x}, \hat{y}$. Other vehicles are the vehicles in front of the ego vehicle within a four-meter radius. Subsequently, i in subscript refers to values of an other (than ego) vehicle. A camera image provides more (pixelated) information but we limit it for rule extraction: position coordinates, speed, and movement direction. The system uses a convolutional network to predict the next waypoints of the ego vehicle. The network consists of a randomly initialized ResNet-18 backbone with three up-convolutional layers to produce an output feature map (476.3MB). These waypoints are further translated into steering, throttle, and brake using the desired speed from the waypoints. However, for the use case, we only consider the predicted desired speed p_s. The initial condition of the scenario, which specifies the starting positions of the ego and other vehicles, is given as $init(v) := \Big(x_e = x_{init^e} \wedge y_e = y_{init^e} \wedge \Big(\bigwedge_{1\leq i\leq 4} x_i = x_{init^i} \wedge y_i = y_{init^i} \wedge s_i = s_{init^i} \wedge \hat{x}_i = \hat{x}_{init^i} \wedge \hat{y}_i = \hat{y}_{init^i}\Big)\Big)$.

The transition relation of the ego vehicle represents a non-deterministic choice of the desired speed value p_s, which represents the state of the vehicle after 1 time unit. The next state is denoted by the primed versions of the individual variables, e.g., x'_e, or the whole tuple of variables v'. This relation can be defined as $step(v, v') := (x'_e = x_e + p_s * \hat{x}_e \wedge y'_e = y_e + p_s * \hat{y}_e)$. As the next waypoint, and thus desired speed, are computed by the LEC of the ego vehicle, the transition relation is extracted from the neural network of the LEC. The transition relations for the other vehicles ov also represent non-deterministic choices for their next x, y values. In general, information regarding these choices is made available via the CARLA simulator. For instance, let us assume that the next horizontal position x'_i ranges between x_i and $x_i \pm r^{(i)}_{max}$ (same for y_i position) for the other vehicles, where $r^{(i)}_{max} \leq 5$ is a parameter derived from the simulated agents. The transition relation for other vehicles is given as $step_{ov}((x_i, y_i, r^{(i)}_{max}), (x'_i, y'_i, , r^{(i)}_{max})) := (x_i - r^{(i)}_{max} \leq x'_i \leq x_i + r^{(i)}_{max} \wedge y_i - r^{(i)}_{max} \leq y'_i \leq y_i + r^{(i)}_{max})$.

One important safety requirement in the use case is collision avoidance. The condition that the ego vehicle remains outside the path of the collision is calculated by the angle at which both vehicles are heading towards each other ($ang^\circ = angle((x_i, y_i), (x_e, y_e))$) and the distance ($dist = norm(x_i - x_e, y_i - y_e)$) they can cover between them, i.e., $safe(v) := (ang^\circ \geq 60^\circ \wedge \neg(ang^\circ \leq 15^\circ \wedge dist \leq p_s))$. By encoding the ego vehicle as one player and the other vehicles as the second player, the verification proof rules from Section 2 can be applied to prove the above safety property.

In our experiment setup, a privileged agent is placed in the simulator's autonomous driving leader-

	Description	Number of extracted rules	Average size of rules	Result	Time (sec)
1.	Straight driving	2065	14.5	`unsat`	123
2.	Driving with one turn	1382	13.8	`sat`	574
3.	Full navigation with multiple turns	12414	20.5	`unsat`	730
4.	Full navigation with traffic	5661	17.4	`sat`	2575

Table 1: Verification result for the example scenario

board and run under a given traffic conditions across multiple routes so that its behavior is evaluated. A total of four traffic conditions are considered for the evaluation: *driving straight*, *driving with one turn*, *full navigation with multiple turns*, and *full navigation routes but with traffic*. The experimental results are given in Table 1. For each scenario, the table contains a short description, the number of extracted rules, and the number of conjuncts in the body of each rule. In addition, the result and duration of the constraints solving are also given.

In conclusion, we have presented the first step towards a CHC-based approach for practical analysis of neural networks, where CHC can be used for verification and explanation but also as evidence aretfacts for safety certification of neural networks.

References

[1] Stanley Bak, Changliu Liu & Taylor T. Johnson (2021): *The Second International Verification of Neural Networks Competition (VNN-COMP 2021): Summary and Results*. *CoRR* abs/2109.00498.

[2] Tewodros A. Beyene, Swarat Chaudhuri, Corneliu Popeea & Andrey Rybalchenko (2014): *A constraint-based approach to solving games on infinite graphs*. *POPL*.

[3] Tewodros A. Beyene, Corneliu Popeea & Andrey Rybalchenko (2013): *Solving Existentially Quantified Horn Clauses*. In: *CAV*.

[4] Tewodros A. Beyene & Amit Sahu (2020): *Rule-Based Safety Evidence for Neural Networks*. In: *SAFECOMP Workshops*.

[5] Dian Chen, Brady Zhou, Vladlen Koltun & Philipp Krähenbühl (2019): *Learning by Cheating*. *ArXiv* abs/1912.12294.

[6] Alexey Dosovitskiy, Germán Ros, Felipe Codevilla, Antonio M. López & Vladlen Koltun (2017): *CARLA: An Open Urban Driving Simulator*. *ArXiv* abs/1711.03938.

[7] Sergey Grebenshchikov, Nuno P. Lopes, Corneliu Popeea & Andrey Rybalchenko (2012): *Synthesizing software verifiers from proof rules*. *PLDI*.

[8] A. Gurfinkel, Temesghen Kahsai, Anvesh Komuravelli & Jorge A. Navas (2015): *The SeaHorn Verification Framework*. In: *CAV*.

[9] Tameru Hailesilassie (2016): *Rule Extraction Algorithm for Deep Neural Networks: A Review*. *ArXiv* abs/1610.05267.

[10] Guy Katz, Clark Barrett, David L. Dill, Kyle Julian & Mykel J. Kochenderfer (2017): *Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks*. In: *Computer Aided Verification*.