Neural Network Action Policy Verification via Predicate Abstraction – Dissertation Abstract

Marcel Vinzent
Supervisor: Jörg Hoffmann
Saarland University
Saarland Informatics Campus
Saarbrücken, Germany
{vinzent, hoffmann}@cs.uni-saarland.de

Abstract

Neural networks (NN) are an increasingly important representation of action policies. With their application for real-time decision-making in safety critical areas, like, e.g., autonomous driving, it arises the need to gain trust in the applied policies. The ultimate goal to gain this trust is through formal verification of the policy-induced behavior. This is a challenging endeavor as it compounds the state space explosion with the difficulty of analyzing even single NN decision episodes.

In our work, we make a contribution to cope with this challenge. We approach safety verification through (over-approximating) abstract reachability analysis. We compute predicate abstractions of the policy-restricted state space; expressing the abstract transition computation as a satisfiability modulo theories (SMT) problem, and devise a range of algorithmic enhancements to avoid costly calls to SMT.

First empirical results show that our approach can outperform competing approaches. Future work will further enhance the technique and extend it to support probabilistic settings.

Introduction

Neural networks (NN) are an increasingly important representation of action policies; in particular for real-time decision making in dynamic environments. The vision is elegant and simple: The NN policy can be learned in advance and at run-time a single (computationally efficient) call to the NN suffices to select an action. But how to verify that such a policy is safe? Leading such a proof is potentially very hard as it compounds the state space explosion with the difficulty of analyzing even single NN decision episodes.

In our work, we contribute to cope with this challenge. We tackle non-deterministic state spaces over bounded-integer state variables. Given an NN action policy π, a **start condition** ϕ_0, and an **unsafety condition** ϕ_U, we verify whether a state $s_U \models \phi_U$ is reachable from a state $s_0 \models \phi_0$ under π.

We approach safety verification via the extension of **predicate abstraction (PA)** (Graf and Saïdi 1997; Ball et al. 2001) to deal with NN action policies. PA is defined through a set $\mathcal{P}$ of **predicates**, where each $p \in \mathcal{P}$ is a linear constraint over the state variables (e.g. $x \leq 5$ or $y \geq z$). Abstract states are characterized by truth value assignments over $\mathcal{P}$, grouping together all concrete states that induce the same truth values. Transitions are over-approximated to preserve all possible behaviors. For our purpose of policy safety verification, we are interested in the predicate abstraction of the **policy-restricted** state space Θ^π, i.e., the subgraph of Θ induced by π. We refer to the predicate abstraction of Θ^π as **policy predicate abstraction (PPA)** $\Theta^\pi_\mathcal{P}$. We build the fragment of $\Theta^\pi_\mathcal{P}$ reachable from ϕ_0. If ϕ_U is not reached then policy π is proven to be safe.

To compute PA, one repeatedly needs to decide whether there is a transition from abstract state A to abstract state A' under some action a: *does there exist a state $s \in A$ s.t. executing a in s results in $s' \in A'$?* This transition problem is routinely addressed using SMT solvers such as $Z3$ (de Moura and Bjørner 2008). Now, to compute the PPA $\Theta^\pi_\mathcal{P}$, one additionally needs to check whether $\pi(s) = a$, i.e., whether the policy selects a in s. Solving this over and over again via calls to SMT quickly becomes infeasible due to the complex structure of neural networks.

In our current work, we hence have devised a range of algorithmic enhancements, leveraging relaxed tests to avoid costly calls to SMT. Most importantly, continuous relaxation of the discrete state variables enables to plug in state-of-the-art SMT solvers tailored to NN (Katz et al. 2017, 2019). We also have devised a method using branch-and-bound around relaxed tests to avoid exact calls to SMT altogether, as well as a method that simplifies SMT calls via information obtained through NN analysis. Empirical results so far show that our approach can outperform competing approaches and that our algorithmic enhancements are required for practicality (Vinzent, Steinmetz, and Hoffmann 2022).

Future research will investigate techniques for automatic abstraction refinement, specifically via counter example guided abstraction refinement (Clarke et al. 2000). Here, we will develop refinement approaches to rule out spuriousness related to the policy. Additionally, we will leverage further NN analysis techniques to enhance the efficacy of our approach, e.g., symbolic propagation (Li et al. 2019) and adversarial attack methods (Goodfellow, Shlens, and Szegedy 2015). We also plan to compute quantitative safety results via value iteration (Givan, Leach, and Dean 1997) in the abstract state space of probabilistic systems.

Related Work

There has been remarkable progress on analyzing individual **NN decision episodes**. In our work so far, we query *Marabou* (Katz et al. 2019), which extends Simplex by a

lazy case splitting approach to handle piecewise-linear activation functions. *Marabou* utilizes a symbolic interval propagation approach (Wang et al. 2018b) which leverages interval arithmetic to propagate bounds on the NN input through the network. Other related work (Ehlers 2017) leverages non-symbolic bound propagation for linear relaxation of NN activation nodes. Follow-up work (Wang et al. 2018a) then combines symbolic bound propagation with linear relaxation. Recent work (Li et al. 2019) extends the symbolic propagation approach to general abstract domains; e.g., also zonotopes rather than simple interval bounds. In the context of our work, such bound propagation techniques can be utilized to over-approximate the possible NN policy behavior. Additionally, adversarial attack methods (e.g. (Goodfellow, Shlens, and Szegedy 2015)), can in principle be adapted to certify transition existence.

The verification of **NN decision sequences** – NN policies executed in an environment – is in its infancy. For software verification, there is initial work on abstract interpretation of programs with calls to NN sub-procedures (Christakis et al. 2021). Gros et al. (2020b) apply statistical model checking to statistically verify NN action policies, but this approach is limited to small numbers of start states as these need to be explicitly enumerated. Tran et al. (2019) use star sets to exactly compute respectively over-approximate reachable sets of a system controlled by a neural network; focusing on linear systems however.

The verification of NN controllers through polynomial approximation has been studied in several works (e.g. (Huang et al. 2019; Ivanov et al. 2021)). These approaches are conceptually very different to our work. Specifically, Ivanov et al. (2021) focus on NN with sigmoid/tanh activation functions[1] which allows to compile the NN behavior into a hybrid system – whose composition with the controlled system is amenable to known verification techniques. On the one hand, this immediately enables to perform verification for a broad range of, possibly complex, systems. On the other hand, such verification is bound to approximation. In contrast, our approach is tailored to piecewise-linear activation functions (e.g. ReLU) and leverages NN-specific techniques which enable for exact analysis.

In a context closer to AI sequential decision making, recent work (Akintunde et al. 2018, 2019) explores the use of MIP encodings for bounded-length verification of NN controlled systems. The approach is technically rather different to ours. While we compute individual (abstract) transitions, in bounded-length verification one checks fixed-size path existence, i.e., transition sequences, via a monolithic MIP encoding; iteratively for paths of increasing sizes. This requires to solve encodings of stepwise increasing cost. Empirical evaluations show that for our purposes the practicability of bounded-length verification is rather limited (Vinzent, Steinmetz, and Hoffmann 2022).

Besides verification, there are also **other techniques to gain trust** in an NN action policy, e.g., safe reinforcement learning (see (García and Fernández 2015) for an overview), especially shielding (e.g. (Alshiekh et al. 2017)); or testing (e.g. (Steinmetz et al. 2022)); or any manner of explainable AI that may help to elucidate the NN's action decisions, in particular visualization (e.g. (Gros et al. 2020a)). In fact, beyond verification, our policy predicate abstraction technique might also turn out to be useful for policy visualization purposes; enabling zooming in the policy-restricted state space based on abstraction predicates.

[1]Arguably, piecewise-linear activation functions can be smoothly approximated (e.g., ReLU via Swish (Ramachandran, Zoph, and Le 2018)) and vice versa (e.g., (Dutta et al. 2018)).

Background

State Space Representation. A state space is a tuple $\langle \mathcal{V}, \mathcal{L}, \mathcal{O} \rangle$ of **state variables** $\mathcal{V}$, **action labels** $\mathcal{L}$, and **operators** $\mathcal{O}$. The domain D_v of each variable $v \in \mathcal{V}$ is a non-empty bounded integer interval. Exp denotes the set of **linear integer expressions** over $\mathcal{V}$ (i.e., of the form $d_1 \cdot v_1 + \cdots + d_r \cdot v_r + c$ with $d_1, \ldots, d_r, c \in \mathbb{Z}$). C denotes the set of **linear integer constraints** over $\mathcal{V}$, (i.e., of the form $e_1 \bowtie e_2$ with $\bowtie \in \{\leq, =, \geq\}$ and $e_1, e_2 \in Exp$), and all Boolean combinations thereof. An **operator** $o \in \mathcal{O}$ is a tuple (g, l, u) with **label** $l \in \mathcal{L}$, **guard** $g \in C$, and (partial) **update** $u \colon \mathcal{V} \to Exp$.

A (partial) **variable assignment** s over $\mathcal{V}$ is a function with domain $dom(s) \subseteq \mathcal{V}$ and $s(v) \in D_v$ for all $v \in dom(s)$. By $s_1[s_2]$ we denote the update of s_1 by s_2, i.e., $dom(s_1[s_2]) = dom(s_1) \cup dom(s_2)$, where $s_1[s_2](v) = s_2(v)$ if $v \in dom(s_2)$ and $s_1[s_2](v) = s_1(v)$ otherwise. By $e(s)$, respectively $\phi \in C$, we denote the evaluation of $e \in Exp$, respectively $\phi \in C$, over s. We write $s \models \phi$, if $\phi(s)$ evaluates to true.

The **state space** of $\langle \mathcal{V}, \mathcal{L}, \mathcal{O} \rangle$ is a labeled transition system (LTS) $\Theta = \langle \mathcal{S}, \mathcal{L}, \mathcal{T} \rangle$. The **states** $\mathcal{S}$ are the complete variable assignments over $\mathcal{V}$. For the **transitions** $\mathcal{T} \subseteq \mathcal{S} \times \mathcal{L} \times \mathcal{S}$ it holds $(s, l, s') \in \mathcal{T}$ iff there exists an operator $o = (g, l, u)$ such that $s \models g$ (the guard is satisfied in the source state s) and $s' = s[u(s)]$ with $u(s) = \{v \mapsto u(v)(s) \mid v \in dom(u)\}$ (the successor state s' is the update of s by u evaluated over s). We also write $s \models o$ for $s \models g$ and $s[\![o]\!]$ for $s[u(s)]$.

Observe that the separation between action labels and operators allows both, state-dependent effects (different operators with the same label l applicable in different states); as well as action outcome non-determinism (different operators with the same label l applicable in the same state).

NN Action Policies. An **action policy** π is a function $\mathcal{S} \to \mathcal{L}$. The **policy-restricted state space** Θ^π is the subgraph $\langle \mathcal{S}, \mathcal{L}, \mathcal{T}^\pi \rangle$ of Θ with $\mathcal{T}^\pi = \{(s, l, s') \in \mathcal{T} \mid \pi(s) = l\}$.

We consider action policies represented by **neural networks** (NN). Specifically, we focus on fully connected feed-forward NN with piecewise-linear activation functions. In the input layer there is an input for each state variable; and in the output layer there is an output for each action label. The policy selects an action label by applying argmax to the output layer.

Policy Safety. A **safety property** is a pair $\rho = (\phi_0, \phi_U)$, where $\phi_0, \phi_U \in C$. Here, ϕ_U identifies the set of unsafe states that should be unreachable from the set of possible start states represented by ϕ_0. That is, π is **unsafe** with respect to ρ iff there exist states $s_0, s_U \in \mathcal{S}$ such that $s_0 \models \phi_0$,

$s_U \models \phi_U$, and s_U is reachable from s_0 in policy-restricted Θ^π. Otherwise π is **safe**.

Policy Predicate Abstraction

In general, it is not feasible to perform explicit reachability analysis of ϕ_U (from ϕ_0) in Θ^π. Instead, our approach is to perform reachability analysis in the abstract state space obtained through **predicate abstraction** (Graf and Saïdi 1997). Given a set of predicates $\mathcal{P} \subseteq C$, an **abstract state** $s_\mathcal{P}$ is a (complete) truth value assignment over $\mathcal{P}$. The abstraction of a (concrete) state $s \in \mathcal{S}$ is the abstract state $s|_\mathcal{P}$ with $s|_\mathcal{P}(p) = p(s)$ for each $p \in \mathcal{P}$. Conversely, $[s_\mathcal{P}] = \{s' \in \mathcal{S} \mid s'|_\mathcal{P} = s_\mathcal{P}\}$ denotes the concretization of $s_\mathcal{P}$, i.e., the set of all concrete state represented by $s_\mathcal{P}$. Accordingly, we say that $s_\mathcal{P}$ satisfies a constraint $\phi \in C$, written $s_\mathcal{P} \models \phi$, iff there exists $s \in [s_\mathcal{P}]$ such that $s \models \phi$.

The abstract state space is then defined in a transition-preserving manner:

Definition 1 (Predicate Abstraction of Θ^π)**.** The **predicate abstraction** of Θ^π over $\mathcal{P}$ is the LTS $\Theta^\pi_\mathcal{P} = \langle \mathcal{S}_\mathcal{P}, \mathcal{L}, \mathcal{T}^\pi_\mathcal{P} \rangle$, where $\mathcal{S}_\mathcal{P}$ is the set of all predicates states over $\mathcal{P}$, and $\mathcal{T}^\pi_\mathcal{P} = \{(s|_\mathcal{P}, l, s'|_\mathcal{P}) \mid (s, l, s') \in \mathcal{T}^\pi\}$.

Due to the underlying policy π, we refer to $\Theta^\pi_\mathcal{P}$ as **policy predicate abstraction**. Due to its over-approximating nature, safety of π can be proven via safety in $\Theta^\pi_\mathcal{P}$:

Proposition 2 (Safety in $\Theta^\pi_\mathcal{P}$)**.** Let $\rho = (\phi_0, \phi_U)$ be a safety property. If there do **not** exist $s_\mathcal{P}, s'_\mathcal{P} \in \mathcal{S}_\mathcal{P}$ with $s_\mathcal{P} \models \phi_0$, $s'_\mathcal{P} \models \phi_U$ such that $s'_\mathcal{P}$ is reachable from $s_\mathcal{P}$ in $\Theta^\pi_\mathcal{P}$, then π is **safe** with respect to ρ.

The computation of $\Theta^\pi_\mathcal{P}$ necessitates to solve the **transition problem** for every possible abstract state transition: $(s_\mathcal{P}, l, s'_\mathcal{P}) \in \mathcal{T}^\pi_\mathcal{P}$ iff there exists an operator $o \in \mathcal{O}$ with label l and a concrete state $s \in [s_\mathcal{P}]$ such that $s \models o$, $s[\![o]\!] \in s'_\mathcal{P}$, and $\pi(s) = l$. We can check this via individual tests for each l-labeled operator:

Definition 3 (Transition Test of $\Theta^\pi_\mathcal{P}$)**.** Let $s_\mathcal{P}, s'_\mathcal{P} \in \mathcal{S}_\mathcal{P}$, and let $o = (g, l, u)$ be an operator. The **transition test** of $\Theta^\pi_\mathcal{P}$, denoted $\mathbf{TSat}^\pi(s_\mathcal{P}, o, s'_\mathcal{P})$, is fulfilled iff there exists $s \in [s_\mathcal{P}]$ such that $s \models o$, $s[\![o]\!] \in [s'_\mathcal{P}]$ and $\pi(s) = l$.

Transition tests are routinely addressed as satisfiability modulo theories (SMT) (Barrett et al. 1994) problems. Predicate abstraction is applicable in principle so long as any method for solving these is available. Compared to standard predicate abstraction approaches, the dominating source of complexity in computing policy predicate abstraction is that, in addition to the standard transition condition, one needs to check whether the policy π actually selects l in $s \in [s_\mathcal{P}]$.

Algorithmic Enhancements

An exact SMT solution of $\mathrm{TSat}^\pi(s_\mathcal{P}, o, s'_\mathcal{P})$ is computationally very expensive – specifically due to the large number of disjunctions encoding every (piecewise-linear) activation function in the NN representation of π. In our current work, we address this via a range of algorithmic enhancements, leveraging relaxed tests that over-approximate $\mathrm{TSat}^\pi(s_\mathcal{P}, o, s'_\mathcal{P})$. If such a relaxed test is violated, then $\mathrm{TSat}^\pi(s_\mathcal{P}, o, s'_\mathcal{P})$ is violated as well.

Necessary Conditions. One relaxation technique is through tests on necessary conditions of $\mathrm{TSat}^\pi(s_\mathcal{P}, o, s'_\mathcal{P})$. Necessary condition that we check are: label selection ($\exists s \in [s_\mathcal{P}] \colon \pi(s) = l$; short $l \in \pi(s_\mathcal{P})$), operator applicability ($s_\mathcal{P} \models o$), respectively their combination ($\exists s \in [s_\mathcal{P}] \colon \pi(s) = l \wedge s \models o$), and the non-policy-restricted transition condition ($\exists s \in [s_\mathcal{P}] \colon s \models o \wedge s[\![o]\!] \in [s'_\mathcal{P}]$).

These conditions essentially check different parts of $\mathrm{TSat}^\pi(s_\mathcal{P}, o, s'_\mathcal{P})$ in isolation. Tests on these conditions have the decisive advantage that, if one such test is violated, one can skip all corresponding transition tests. For instance, if we find $l \notin \pi(s_\mathcal{P})$, then $\mathrm{TSat}^\pi(s_\mathcal{P}, o, s'_\mathcal{P})$ is violated and can thus be skipped for all l-labeled operators o and all abstract successor states $s'_\mathcal{P}$. Additionally, tests ignoring the NN selection condition are usually much cheaper to answer.

Continuous Relaxation. Another relaxation technique is through continuous relaxation. Each test ($\mathrm{TSat}^\pi(s_\mathcal{P}, o, s'_\mathcal{P})$ as well as tests on necessary conditions) can be relaxed by interpreting the integer state variables as continuous variables (with real-valued interval domains). The advantage of this relaxation is the applicability of existing SMT solvers specialized to NN analysis. Specifically, in our work so far, we query *Marabou* (Katz et al. 2019), an SMT solver tailored to NN with piecewise-linear activation functions.

If a relaxed test is violated, the corresponding exact test is violated too (and can be skipped). However, continuously-relaxed tests can be utilized even further: If the solution found to a relaxed test happens to be integer, the corresponding exact test is derived to be fulfilled as well. While this will in general not be the case, we can iterate relaxed tests in a **branch & bound (B&B)** search for such a solution. In each iteration, if there exists a state variable v assigned to a non-integer value α in the relaxed solution found by the solver, we pick one such v and create two search branches, restricting v to be less equal $\lfloor \alpha \rfloor$ respectively greater equal $\lceil \alpha \rceil$. A branch is terminated once the relaxed tests is found to be violated, or when an integer solution is found. If no integer solution is found during the search, the exact tests is violated. The advantage of this approach is the applicability of existing SMT solvers dedicated to NN analysis to answer not only relaxed but also exact tests.

Fixing Activation Cases. Additionally to the enhancement trough relaxed tests, there are further techniques from NN analysis that can be utilized. One such option is **fixing activation cases** in the NN. That is, given value bounds on the neurons in the network one can potentially prune some or even fix one of the (piecewise-linear) activation cases of the respective neuron. For instance, for the prominent ReLU activation function $\mathrm{ReLU}(x) := \max(x, 0)$ the idea works as follows: If the activation-function input x is known to be less equal 0, then one can fix $\mathrm{ReLU}(x) = 0$; if x is known to be greater equal 0, one can fix $\mathrm{ReLU}(x) = x$.

There is a broad range of NN analysis techniques that can be utilized to compute such bounds (e.g. (Wang et al. 2018b; Li et al. 2019)) given constraints on the input of an NN (in our case $s_\mathcal{P}$). Towards transition tests – relaxed or exact – activation case fixing as well as the derived bounds themselves can be used to simplify the resulting SMT encodings

and to prune the SMT search space. This idea has been deployed in other work before (e.g. (Mohammadi et al. 2020; Katz et al. 2019)). Specifically, *Marabou* (Katz et al. 2019), which we query for relaxed tests, fixes activation cases based on bounds implied by individual constraints, respectively derived through symbolic interval propagation on the network topology (Wang et al. 2018b). In our work, we then also extract these bounds towards activation case fixing in exact tests.

Experimental Results

In our experiments so far (Vinzent, Steinmetz, and Hoffmann 2022), we evaluated our approach on a collection of benchmarks that involve action outcome non-determinism. As competing approaches, we implemented an explicit-search approach enumerating all states the policy can reach, as well as a bounded-length verification approach following the ideas of Akintunde et al. (2018; 2019). The results show that our algorithmic enhancements – in particular relaxed tests answered by leveraging dedicated NN analysis techniques – are required for practicality, and that our approach can outperform its competitors.

Future Work

We distinguish three research lines that we plan to cover in our future work: technical enhancements, adaption to other settings, and automatic abstraction refinement. We also note that there are many more approaches from formal methods that (when adapted to the the NN policy setting) can in principle serve as competing verification approaches (e.g. (Tonetta 2009; Cimatti et al. 2016)).

Technical Enhancements. There remains a broad range of NN analysis techniques that can be utilized to improve performance. For instance, techniques to derive tight neuron value bounds (e.g. (Li et al. 2019)) can be leveraged to enhance activation case fixing. In principle, our approach can profit from any progress in the analysis of single NN decision episodes.

Furthermore, we also plan to leverage **adversarial attack** methods for under-approximation purposes (e.g. (Goodfellow, Shlens, and Szegedy 2015)). In essence, if an attack finds a solution to a transition test, the call to SMT can be skipped. If not, there still may exist a solution and we call SMT. That said, robustness guarantees on adversarial attacks (e.g. (Hein and Andriushchenko 2017)) might even be used to prove that a solution does not exist.

On the technical level there is also a vast potential for **parallelization** – running several transition tests at once (different (solver) techniques, different transitions); as well as **incremental solving** – e.g., preserving the solver-internal state between selection and transition tests.

Adaption to Other Settings. In our current setting, we consider non-probabilistic state spaces with non-deterministic action outcomes. A natural extension to make our approach amenable to **probabilistic** systems is by performing value iteration (e.g. (Givan, Leach, and Dean 1997)) in the abstract state space; obtaining quantitative safety results. The abstraction computation itself remains unchanged.

Currently, we allow π to select inapplicable actions, i.e., there may exist $s \in \mathcal{S}$ such that $\pi(s)$ does not label any outgoing transition and the policy execution stalls. Here, as potential future work, we plan to extend our approach to enable stalling detection. Alternatively, a prominent option is also to super-impose applicability on π, restricting its selection to the applicable actions. Again, we plan to adapt our approach to such settings as part of future work. In principle, both adaptions are straight-forward; resulting in significantly more expensive SMT problems tough.

Abstraction Refinement. In our work so far, we assume the predicate set $\mathcal{P}$ to be provided as input. Yet, the converse of Proposition 2 does not hold, i.e., unsafety in $\Theta^{\pi}_{\mathcal{P}}$ does not imply unsafety in Θ^{π}. Depending on $\mathcal{P}$, the abstraction may contain *spurious* paths without correspondence in the concrete state space. Hence, automatic abstraction generation (respectively refinement) remains key future work towards a complete verification procedure.

Here, **counter example guided abstraction refinement** (CEGAR) (e.g. (Clarke et al. 2000)) is a common procedure to refine $\mathcal{P}$, iteratively removing spurious (unsafe) paths until either the abstraction is proven safe, or a *non-spurious* unsafe path is found – proving unsafety of the concrete system. The adaption of CEGAR to NN policy verification is non-trivial since refinement predicates must reflect NN selection behavior; a path $s^1_{\mathcal{P}}, l^1, \ldots, s^i_{\mathcal{P}}, l^i, \ldots, s^n_{\mathcal{P}}$ may be spurious due to $l^i \in \pi(s^i_{\mathcal{P}})$ while $\pi(s^i) \neq l^i$ in the path concretizations. Our future research will address this challenge. One idea is to approximate (state-dependent) "selection guards" via constraints that split $s^i_{\mathcal{P}}$ based on NN behavior, e.g., with respect to s_i in a path concretization. The selection guards can then be fed into standard refinement procedures based on weakest precondition computation.

In the context of CEGAR, we also plan to investigate the potential of incremental abstraction computation, i.e., reusing (transition) information of coarser abstractions when computing the refined abstract state space. Most importantly such information can be used to prune the potential transition space as well as to witness existing transitions. Another technique of interest is lazy abstraction, i.e., refining the abstraction only locally respectively checking (abstract) policy-restriction only once an abstract unsafe path has been found. Moreover, the search for counter examples may also be sped up using heuristics, e.g., again based on information from coarser (non-policy-restricted) abstractions.

Conclusion

In our work, we provide policy predicate abstraction as a new method to address NN policy verification. Our experiments so far have shown that it can outperform competing approaches and that our algorithmic enhancements are required for practicality. An important future task is to address the automatic selection of the abstraction predicates.

Overall, we believe that NN policy verification is important, and we hope that our work provides one basic building block for this huge endeavor.

References

Akintunde, M.; Lomuscio, A.; Maganti, L.; and Pirovano, E. 2018. Reachability Analysis for Neural Agent-Environment Systems. In *KR*.

Akintunde, M. E.; Kevorchian, A.; Lomuscio, A.; and Pirovano, E. 2019. Verification of RNN-Based Neural Agent-Environment Systems. In *AAAI*.

Alshiekh, M.; Bloem, R.; Ehlers, R.; Könighofer, B.; Niekum, S.; and Topcu, U. 2017. Safe Reinforcement Learning via Shielding. *CoRR*, abs/1708.08611.

Ball, T.; Majumdar, R.; Millstein, T. D.; and Rajamani, S. K. 2001. Automatic Predicate Abstraction of C Programs. In *Prog. Lang. Design and Implementation (PLDI)*.

Barrett, C. W.; Sebastiani, R.; Seshia, S. A.; and Tinelli, C. 1994. Satisfiability modulo theories. *In Handbook of Satisfiability*, 825–885.

Christakis, M.; Eniser, H. F.; Hermanns, H.; Hoffmann, J.; Kothari, Y.; Li, J.; Navas, J.; and Wüstholz, V. 2021. Automated Safety Verification of Programs Invoking Neural Networks. In *CAV*.

Cimatti, A.; Griggio, A.; Mover, S.; and Tonetta, S. 2016. Infinite-state invariant checking with IC3 and predicate abstraction. *Formal Methods Syst. Des.*, 49(3): 190–218.

Clarke, E. M.; Grumberg, O.; Jha, S.; Lu, Y.; and Veith, H. 2000. Counterexample-Guided Abstraction Refinement. In *CAV*.

de Moura, L.; and Bjørner, N. 2008. Z3: An Efficient SMT Solver. In *TACAS*.

Dutta, S.; Jha, S.; Sankaranarayanan, S.; and Tiwari, A. 2018. Output Range Analysis for Deep Feedforward Neural Networks. In Dutle, A.; Muñoz, C. A.; and Narkawicz, A., eds., *NFM*.

Ehlers, R. 2017. Formal Verification of Piece-Wise Linear Feed-Forward Neural Networks. In D'Souza, D.; and Kumar, K. N., eds., *Automated Technology for Verification and Analysis*.

García, J.; and Fernández, F. 2015. A comprehensive survey on safe reinforcement learning. *JMLR*, 16: 1437–1480.

Givan, R.; Leach, S. M.; and Dean, T. L. 1997. Bounded Parameter Markov Decision Processes. In Steel, S.; and Alami, R., eds., *ECP*.

Goodfellow, I. J.; Shlens, J.; and Szegedy, C. 2015. Explaining and Harnessing Adversarial Examples. In *Learning Representations (ICLR)*.

Graf, S.; and Saïdi, H. 1997. Construction of Abstract State Graphs with PVS. In *CAV*.

Gros, T. P.; Groß, D.; Gumhold, S.; Hoffmann, J.; Klauck, M.; and Steinmetz, M. 2020a. TraceVis: Towards Visualization for Deep Statistical Model Checking. In *Proceedings of the 9th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation (ISoLA'20)*.

Gros, T. P.; Hermanns, H.; Hoffmann, J.; Klauck, M.; and Steinmetz, M. 2020b. Deep Statistical Model Checking. In *Formal Techniques for Distributed Objects, Components, and Systems (FORTE)*.

Hein, M.; and Andriushchenko, M. 2017. Formal Guarantees on the Robustness of a Classifier against Adversarial Manipulation. In *NIPS*.

Huang, S.; Fan, J.; Li, W.; Chen, X.; and Zhu, Q. 2019. ReachNN: Reachability analysis of neural-network controlled systems. *ACM Trans. Emb. Comp. Sys.*, 18: 1–22.

Ivanov, R.; Carpenter, T. J.; Weimer, J.; Alur, R.; Pappas, G. J.; and Lee, I. 2021. Verifying the Safety of Autonomous Systems with Neural Network Controllers. *ACM Trans. Emb. Comp. Sys.*, 20: 7:1–7:26.

Katz, G.; Barrett, C. W.; Dill, D. L.; Julian, K.; and Kochenderfer, M. J. 2017. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. In *CAV*.

Katz, G.; Huang, D. A.; Ibeling, D.; Julian, K.; Lazarus, C.; Lim, R.; Shah, P.; Thakoor, S.; Wu, H.; Zeljic, A.; Dill, D. L.; Kochenderfer, M.; and Barrett, C. 2019. The Marabou Framework for Verification and Analysis of Deep Neural Networks. In *CAV*.

Li, J.; Liu, J.; Yang, P.; Chen, L.; Huang, X.; and Zhang, L. 2019. Analyzing deep neural networks with symbolic propagation: Towards higher precision and faster verification. In *Static Analysis (SAS)*.

Mohammadi, K.; Karimi, A.; Barthe, G.; and Valera, I. 2020. Scaling Guarantees for Nearest Counterfactual Explanations. *CoRR*, abs/2010.04965.

Ramachandran, P.; Zoph, B.; and Le, Q. V. 2018. Searching for Activation Functions. In *ICLR Workshop Track Proceedings*.

Steinmetz, M.; Fiser, D.; Eniser, H.; Ferber, P.; Gros, T.; Heim, P.; Höller, D.; Schuler, X.; Wüstholz, V.; Christakis, M.; and Hoffmann, J. 2022. Debugging a Policy: Automatic Action-Policy Testing in AI Planning. In *Proceedings of the 32nd International Conference on Automated Planning and Scheduling (ICAPS)*.

Tonetta, S. 2009. Abstract Model Checking without Computing the Abstraction. In Cavalcanti, A.; and Dams, D., eds., *Formal Methods*.

Tran, H.; Cai, F.; Lopez, D. M.; Musau, P.; Johnson, T. T.; and Koutsoukos, X. D. 2019. Safety Verification of Cyber-Physical Systems with Reinforcement Learning Control. *ACM Trans. Embed. Comput. Syst.*, 18(5s): 105:1–105:22.

Vinzent, M.; Steinmetz, M.; and Hoffmann, J. 2022. Neural Network Action Policy Verification via Predicate Abstraction. In *Proceedings of the 32nd International Conference on Automated Planning and Scheduling (ICAPS)*.

Wang, S.; Pei, K.; Whitehouse, J.; Yang, J.; and Jana, S. 2018a. Efficient Formal Safety Analysis of Neural Networks. In Bengio, S.; Wallach, H. M.; Larochelle, H.; Grauman, K.; Cesa-Bianchi, N.; and Garnett, R., eds., *NeurIPS*.

Wang, S.; Pei, K.; Whitehouse, J.; Yang, J.; and Jana, S. 2018b. Formal Security Analysis of Neural Networks using Symbolic Intervals. In *USENIX Security Symposium*.