ARCH-COMP19 Category Report: Artificial Intelligence / Neural Network Control Systems (AINNCS) for Continuous and Hybrid Systems Plants

Diego Manzanas Lopez[1], Patrick Musau[1], Hoang-Dung Tran[1], Souradeep Dutta[2], Taylor J. Carpenter[3], Radoslav Ivanov[3], Taylor T. Johnson[1]

[1] Vanderbilt University
Nashville, TN
{diego.manzanas.lopez, patrick.musau, dung.h.tran, taylor.johnson}@vanderbilt.edu
[2] University of Colorado at Boulder
Boulder, CO
souradeep.dutta@colorado.edu
[3] University of Pennsylvania
Philadelphia, PA
{carptj, rivanov}@seas.upenn.edu

Abstract

This report presents the results of a friendly competition for formal verification of continuous and hybrid systems with artificial intelligence (AI) components. Specifically, machine learning (ML) components in cyber-physical systems (CPS), such as feedforward neural networks used as feedback controllers in closed-loop systems are considered, which is a class of systems classically known as intelligent control systems, or in more modern and specific terms, neural network control systems (NNCS). For future iterations, we more broadly refer to this category as AI and NNCS (AINNCS). The friendly competition took place as part of the workshop Applied Verification for Continuous and Hybrid Systems (ARCH) in 2019. In the first edition of this AINNCS category at ARCH-COMP, three tools have been applied to solve five different benchmark problems, (in alphabetical order): NNV, Sherlock, and Verisig. This report is a snapshot of the current landscape of tools and the types of benchmarks for which these tools are suited. Due to the diversity of problems and that this is the first iteration of this category, we are not ranking tools in terms of performance, yet the presented results probably provide the most complete assessment of tools for the safety verification of NNCS.

1 Introduction

Artificial Neural Networks have demonstrated an impressive ability to be used as frameworks for solving complex problems in numerous application domains [27]. In fact, the success of these models in contexts such as adaptive control, non-linear system identification [20], image and pattern recognition, function approximation, and machine translation, has stimulated the creation of technologies that are directly impacting our everyday lives [23], and has led researchers to believe that these models possess the power to revolutionize the development of robust and intelligent systems in a diverse set of arenas [22].

Despite these achievements, there have been reservations in utilizing them within high assurance systems due to their susceptibility to unexpected and errant behavior caused by slight perturbations in their inputs [17]. In a famous study by Chrisitian Szegedy et al. [24], the authors demonstrated that by carefully applying a hardly perceptible modification to an input image, one could cause a successfully trained neural network to produce an incorrect classification. These inputs are known as *adversarial examples*, and their discovery has caused concern over the safety, reliability, and security of neural network applications [27]. As a result, there has been a large research impetus towards obtaining an explicit understanding of neural network behavior.

Neural networks are often viewed as ”black boxes,” whose underlying operation is often incomprehensible, and the last several years have witnessed a large number of promising verification methods proposed towards reasoning about the correctness of their behavior. However, it has been demonstrated that neural network verification is an NP-complete problem [16], and while current state-of-the-art verification methods have been able to deal with small networks, they are incapable of dealing with the complexity and scale of networks utilized in practice ([18, 13, 4]). Additionally, while in recent years there has been a large amount of work focused on verifying pre-/post-conditions for neural networks in isolation, reasoning about the behavior of neural network control systems remains a key challenge.

The following report aims to provide a survey of the landscape of the current capabilities of verification tools for *closed-loop systems with neural network controllers*, as these systems have displayed great utility as a means for learning control laws through techniques such as reinforcement learning and data-driven predictive control [10]. Furthermore, this report aims to provide readers with a perspective of the intellectual progression of this rapidly growing field and stimulate the development of efficient and effective methods capable of being utilized within real-life applications.

Disclaimer The presented report of the ARCH friendly competition for *closed-loop systems with neural network controllers*, termed in short AINNCS (Artificial Intelligence / Neural Network Control Systems), aims to provide the landscape of the current capabilities of verification tools for analyzing these systems that are classically known as *intelligent control systems*. We would like to stress that each tool has unique strengths—not all of the specificities can be highlighted within a single report. To reach a consensus in what benchmarks are used, some compromises had to be made so that some tools may benefit more from the presented choice than others. The obtained results have been verified by an independent repeatability evaluation. To establish further trustworthiness of the results, the code with which the results have been obtained is publicly available at gitlab.com/goranf/ARCH-COMP.

Specifically, this report summarizes results obtained in the 2019 friendly competition of the ARCH workshop[1] for verifying hybrid systems with linear and non-linear continuous dynamics

$$\dot{x}(t) = f(x(t), u(t)),$$

where x(t),u(t) correspond to the states and inputs of the plant at time t, respectively. Participating tools are summarized in Sec. 2. Please, see [27] for further details on these and additional tools. The results of our selected benchmark problems are shown in Sec. 3 and are obtained on the tool developers' own machines. Thus, one has to factor in the computational power of the processors used, summarized in Appendix A, as well as the efficiency of the programming language of the tools. The architecture of the closed-loop systems we will evaluate is depicted in Figure 1.

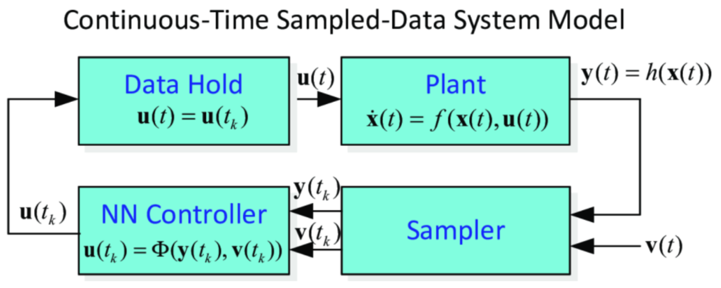
Continuous-Time Sampled-Data System Model
Data Hold
$\mathbf{u}(t) = \mathbf{u}(t_k)$
$\mathbf{u}(t)$
Plant
$\dot{\mathbf{x}}(t) = f(\mathbf{x}(t), \mathbf{u}(t))$
$\mathbf{y}(t) = h(\mathbf{x}(t))$
$\mathbf{u}(t_k)$
NN Controller
$\mathbf{u}(t_k) = \Phi(\mathbf{y}(t_k), \mathbf{v}(t_k))$
$\mathbf{y}(t_k)$
$\mathbf{v}(t_k)$
Sampler
$\mathbf{v}(t)$

Figure 1: Closed-loop architecture of the benchmarks to be verified.

The goal of the friendly competition is not to rank the results, but rather to present the landscape of existing solutions in a breadth that is not possible with scientific publications in classical venues. Such publications would typically require the presentation of novel techniques, while this report showcases the current state-of-the-art tools. For all results reported by each participant, we have run an independent repeatability evaluation.

The selection of the benchmarks has been conducted within the forum of the ARCH website (cps-vo.org/group/ARCH), which is visible for registered users and registration is open for anyone. All tools presented in this report use some form of reachability analysis. This, however, is not a constraint set by the organizers of the friendly competition. We hope to encourage further tool developers to showcase their results in future editions.

2 Participating Tools

We present a brief overview of all the participating tools in this friendly competition, which are NNV, Verisig, and Sherlock. The tools participating in this AINNCS category *Artificial Intelligence / Neural Network Control Systems in Continuous and Hybrid Systems Plants* are introduced subsequently in alphabetical order.

[1]Workshop on Applied Verification for Continuous and Hybrid Systems (ARCH), cps-vo.org/group/ARCH

NNV NNV (Neural Network Verification Tool) [29, 28, 26, 30, 25] is a Matlab based toolbox that implements reachability-analysis methods for neural network verification with a particular focus on applications of closed-loop neural network control systems in autonomous cyber-physical systems. NNV uses a star-set based algorithm that allows for a layer-by-layer computation of the exact reachable set for feed-forward deep neural networks. The star-set based algorithm is naturally parallelizeable which allowed NNV to be designed to perform efficiently on multi-core platforms. Additionally, in the event that a particular safety property is violated, NNV can be usd to construct and visualize the complete set of counterexample inputs for a neural network. Using NNV in combination with HyST [6, 5] CORA[1, 2, 3] allows for the verification of closed-loop neural network control systems. NNV along with the relevant experiments and related publications are available at `https://github.com/verivital/nnv`.

Sherlock Sherlock, [12] is a tool for range analysis for deep feedforward neural networks. Sherlock uses an interleaving of gradient based local search technique, and mixed integer linear programming based global optimization solver to solve an optimization problem, for range analysis. Sherlock can be used to verify properties about closed loop systems, with deep neural networks as controllers as well [11]. This can be accomplished using ideas like template polyhedra and acceleration of reach set computation explained in [11]. In, [10], the authors extended Sherlock to generate polynomial rules to characterize the local behavior of the network in a formally sound way. This enabled the computation of reach sets, for plants modelled as an ODE, and a neural network as the controller. The tool is available for download at `https://github.com/souradeep-111/sherlock`.

Verisig Verisig [14] is a tool for verifying safety properties of closed-loop hybrid systems with deep neural network (DNN) components. Verisig supports sigmoid and tanh-based networks and exploits the fact that the sigmoid (and the tanh) is the solution to a quadratic differential equation, which allows us to transform the neural network into an equivalent hybrid system. By composing the network's hybrid system with the plant's, Verisig transforms the problem into a hybrid system verification problem which can be solved using state-of-the-art reachability tools. Specifically, Verisig uses Flow* [9] to solve the resulting hybrid system reachability problem. Verisig is available at `https://verisig.org`, and the code can be found at `https://github.com/verisig`.

3 Benchmarks

For the 2019 edition, we selected 5 benchmarks for this new category. These 5 benchmarks include a cart pole (inverted pendulum), an Adaptive Cruise Control (ACC) system, a TORA and several others, which are introduced subsequently in no particular order and can be found and downloaded online[2] or in the AINNCS category repeatability evaluation repository[3].

3.1 Non-linear Cart-Pole

This benchmark is obtained from the OpenAI gym [8], and corresponds to the version of the cart-pole introduced by Barto, Sutton, and Anderson [7]. Although the overall scheme of this system is very similar to the inverted pendulum, which also consists of a pendulum attached to a cart, this variation of the system is unique in that the dynamic equations used to represent

[2] `https://github.com/verivital/ARCH-COMP19-AINNCS`
[3] `https://gitlab.com/goranf/ARCH-COMP/tree/master/2019/AINNCS`

it are more complex. Moreover, in this case, we represent the dynamics of the system as a set of non-linear differential equations and alter the controller structure in addition to the initial states and constraints. The dynamics of the cart-pole system are described as follows

$$\begin{aligned} \ddot{x} &= \frac{u + ml\omega^2 \sin(\theta)}{m_t} - \frac{ml(g\sin(\theta) - \cos(\theta))(\frac{u+ml\omega^2\sin(\theta)}{m_t})}{l(\frac{4}{3} - m\frac{\cos^2(\theta)}{m_t})}\frac{\cos(\theta)}{m_t}, \\ \ddot{\theta} &= \frac{g\sin(\theta) - \cos(\theta)(\frac{u+ml\omega^2\sin(\theta)}{m_t})}{l(\frac{4}{3} - m\frac{\cos^2(\theta)}{m_t})}\frac{\cos(\theta)}{m_t} \end{aligned} \tag{1}$$

where $u \in \{-10, 10\}$ is the input force, which either pushes the cart left or right, $g = 9.8$ is gravity, $m = 0.1$ is the pole's mass, $l = 0.5$ is half the pole's length, $m_t = 1.1$ is the total mass, x is the position of the cart, θ is the angle of the pendulum with respect to the positive y-axis, $v = \dot{x}$ is the linear velocity of the cart, and $\omega = \dot{\theta}$ is the angular velocity of the pendulum. The controller has four inputs $(x, \dot{x}, \theta, \dot{\theta})$, four layers with $[24, 48, 12, 2]$ neurons respectively, and two outputs. The two outputs are then compared, and the input sent to the plant depends on which output index has the greatest value. Thus, as an example if $output_1 > output_2$ then the input force supplied to the plant is 10. However if $output_1 < output_2$ then the input supplied to the plant is -10.

Specifications For this benchmark, the verification objective is to demonstrate that the pole will eventually reach the upward position and that it will remain there. In other words, the goal is to achieve a value of $\theta = 0$ and stay there. Some other specifications to be met are, for at least 12 seconds, x $\in$ [-2.4,2.4] and $\theta \in$ [-15,15] degrees. The initial conditions for all state variables were chosen uniformly at random between [-0.05, 0.05].

3.2 Adaptive Cruise Controller (ACC)

The Adaptive Cruise Control (ACC) System simulates a system that tracks a set velocity and maintains a safe distance from a lead vehicle by adjusting the longitudinal acceleration of an ego vehicle. The neural network computes optimal control actions while satisfying safe distance, velocity, and acceleration constraints using model predictive control (MPC) [21]. For this case study, the ego car is set to travel at a set speed $V_{set} = 30$ and maintains a safe distance D_{safe} from the lead car. The car's dynamics are described as follows:

$$\begin{aligned} \dot{x}_{\text{lead}}(t) = v_{\text{lead}}(t), \dot{v}_{\text{lead}}(t) = \gamma_{\text{lead}}(t), \dot{\gamma}_{\text{lead}}(t) = -2\gamma_{\text{lead}}(t) + 2a_{\text{lead}} - uv_{\text{lead}}^2(t), \\ \dot{x}_{\text{ego}}(t) = v_{\text{ego}}(t), \dot{v}_{\text{ego}}(t) = \gamma_{\text{ego}}(t), \dot{\gamma}_{\text{ego}}(t) = -2\gamma_{\text{ego}}(t) + 2a_{\text{ego}} - uv_{\text{ego}}^2(t), \end{aligned} \tag{2}$$

where x_i is the position, v_i is the velocity, γ_i is the acceleration of the car, a_i is the acceleration control input applied to the car, and $u = 0.0001$ is the friction control where $i \in \{\text{ego}, \text{lead}\}$. For this benchmark we have developed four neural network controllers with 3, 5, 7, and 10 hidden layers of 20 neurons each. All of them have the same number of inputs (v_{set}, T_{gap}, v_{ego}, D_{rel}, v_{rel}), and one output (a_{ego}).

Specifications The verification objective of this system is that given a scenario where both cars are driving safely, the lead car suddenly slows down with a_{lead} = -2. We want to check whether there is a collision in the following 5 seconds. Formally, this safety specification of the system can be expressed as $D_{\text{rel}} = x_{\text{lead}}$ - $x_{\text{ego}} \geq D_{\text{safe}}$, where $D_{\text{safe}} = D_{\text{default}} + T_{\text{gap}} \times v_{\text{ego}}$, and $T_{\text{gap}} = 1.4$ seconds and $D_{\text{default}} = 10$. The initial conditions are: $x_{\text{lead}}(0) \in$ [90,110], $v_{\text{lead}}(0) \in$ [32,32.2], $\gamma_{\text{lead}}(0) = \gamma_{\text{ego}}(0) = 0$, $v_{\text{ego}}(0) \in$ [30, 30.2], $x_{\text{ego}} \in$ [10,11].

3.3 Sherlock-Benchmark-7

This benchmark was first proposed in [31]. The authors originally designed a discontinuous sliding mode controller for this system in [31]. The model is a 3 dimensional system, given by the following equations:

$$\dot{x_1} = x_3^3 - x_2 + w, \dot{x_2} = x_3, \dot{x_3} = u, \tag{3}$$

where w is a bounded error in the range $[-0.01, 0.01]$. A neural network controller was trained for this system, using a model predictive controller as a demonstrator. The trained network had 2 hidden layers, with 300 neurons in the first layer, and 200 in the second layer. The sampling time for this controller was $0.5s$.

Specification The verification problem here is that of target reachability. For an initial set of, $x_1 \in [0.35, 0.45], x_2 \in [0.45, 0.55], x_3 \in [0.25, 0, 35]$, it is required to prove that the system converges to $x \in [-0.032, 0.032]^3$, within $2s$.

3.4 Sherlock-Benchmark-9 (TORA)

This benchmark is that of a TORA (translational oscillations by a rotational actuator) [10, 15]. The model is that of a cart attached to a wall with a spring, and is free to move an friction-less surface. The cart itself has a weight attached to an arm inside it, which is free to rotate about an axis. This serves as the control input, in order to stabilize the cart at $\mathbf{x} = \mathbf{0}$. The model is 4 dimensional system, given by the following equations :

$$\dot{x_1} = x_2, \dot{x_2} = -x_1 + 0.1sin(x_3), \dot{x_3} = x_4, \dot{x_4} = u, \tag{4}$$

where w is a bounded error in the range $[-0.01, 0.01]$. A neural network controller was trained for this system, using data-driven model predictive controller proposed in [11]. The trained network had 3 hidden layers, with 100 neurons in each layer making a total of 300 neurons. The sampling time for this controller was $1s$.

Specification The verification problem here is that of safety. For an initial set of, $x_1 \in [0.6, 0.7], x_2 \in [-0.7, -0.6], x_3 \in [-0.4, -0.3], x_4 \in [0.5, 0.6]$, it is required to prove that the system stays within the box $\mathbf{x} \in [-1, 1]^3$, for a time window of $20s$.

3.5 Sherlock-Benchmark-10 (Unicycle Car Model)

This benchmark is that of a unicycle model of a car [10]. It models the dynamics of a car involving 4 variables, specifically the x and y coordinates on a 2 dimensional plane, as well as velocity magnitude (speed) and steering angle.

$$\dot{x_1} = x_4cos(x_3), \dot{x_2} = x_4sin(x_3), \dot{x_3} = u_2, \dot{x_4} = u_1 + w, \tag{5}$$

where w is a bounded error in the range $[-1e-4, 1e-4]$. A neural network controller was trained for this system, using a model predictive controller as a "demonstrator" or "teacher". The trained network has 1 hidden layer, with 500 neurons. The sampling time for this controller was $0.2s$.

Specification The verification problem here is that of reachability. For an initial set of , $x_1 \in [9.5, 9.55], x_2 \in [-4.5, -4.45], x_3 \in [2.1, 2.11], x_4 \in [1.5, 1.51]$, it is required to prove that the system reaches the set $x_1 \in [-0.6, 0.6], x_2 \in [-0.2, 0.2], x_3 \in [-0.06, 0.06], x_4 \in [-0.3, 0.3]$ within a time window of $10s$.

4 Verification Results

For each of the three participating tools, we aim to evaluate the verification results on all proposed benchmarks and the output reachable sets are shown.

4.1 NNV

We introduce the results of all the benchmarks proposed utilizing *NNV*.

4.1.1 Cartpole

There are currently no results for this benchmark due to the hybrid (bang-bang) nature of the neural network controller.

4.1.2 ACC

In the case of the ACC benchmark, we present the results using the neural network controller with 101 neurons (5-by-20) with ReLU activation functions, and we use a time horizon of 5 seconds, as shown in Figure 2.

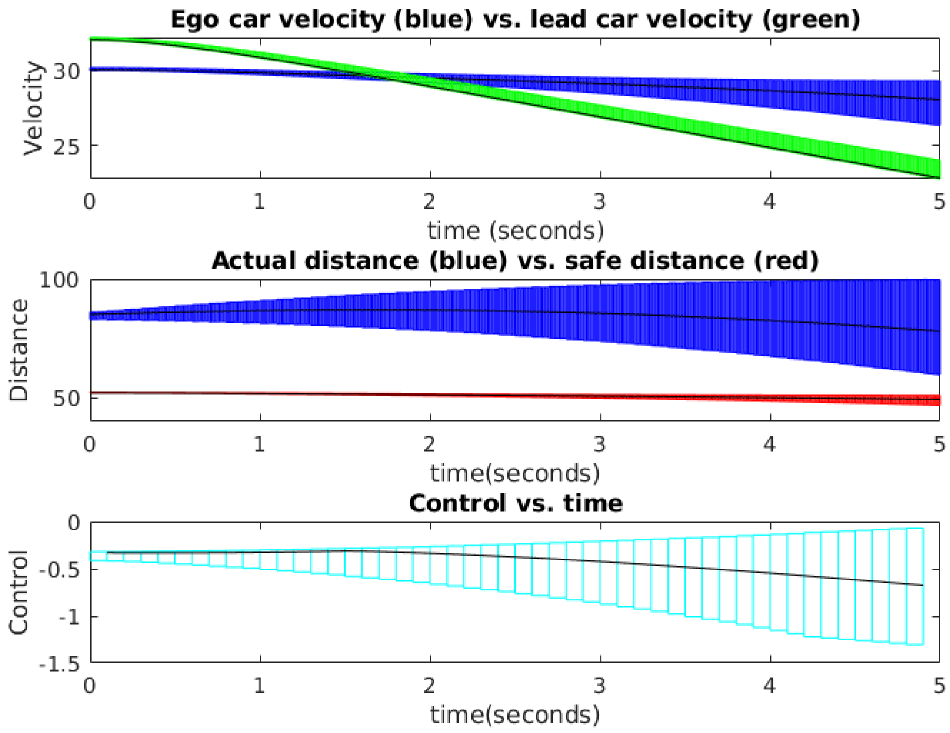

Figure 2: ***NNV.*** Reachability analysis results of the ACC benchmark using a controller with 5 hidden layers (ReLU) of 20 neurons each.

4.1.3 Sherlock-Benchmark-7

There are currently no results for this benchmark. When evaluating NNV on this benchmark, CORA runs out of memory when calculating the reachable set of the plant, regardless of the number of control steps we evaluate it on. This occurs due to a seemingly infinite recursion when attempting to split initial states to yield more precise calculations, so it is the result of overapproximation error growth relative to the error bound parameters used.

4.1.4 Sherlock-Benchmark-9

This benchmark has been evaluated for 2 different number of control steps 10 and 20.

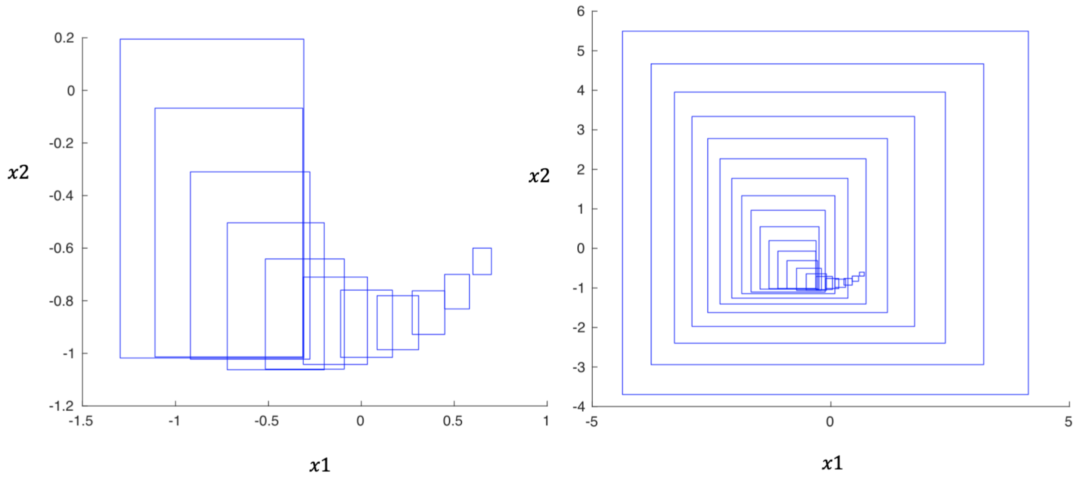

Figure 3: ***NNV.*** Reachability analysis results of benchmark 7. The graph on the left shows the reachable set for 10 control steps. The figure in the right corresponds to 20 steps.

4.1.5 Sherlock-Benchmark-10

There are currently no results for this benchmark. This arose as this benchmark has multiple control inputs, and a tool implementation of the version of NNV evaluated did not allow for more than one control input, although this has been generalized following the competition.

4.2 Sherlock

This section presents the analysis results on the benchmarks with Sherlock. All experiments for Sherlock were run on an Intel Core i5 2.5GHz machine, with 8GB RAM, running Ubuntu 16.04 LTS.

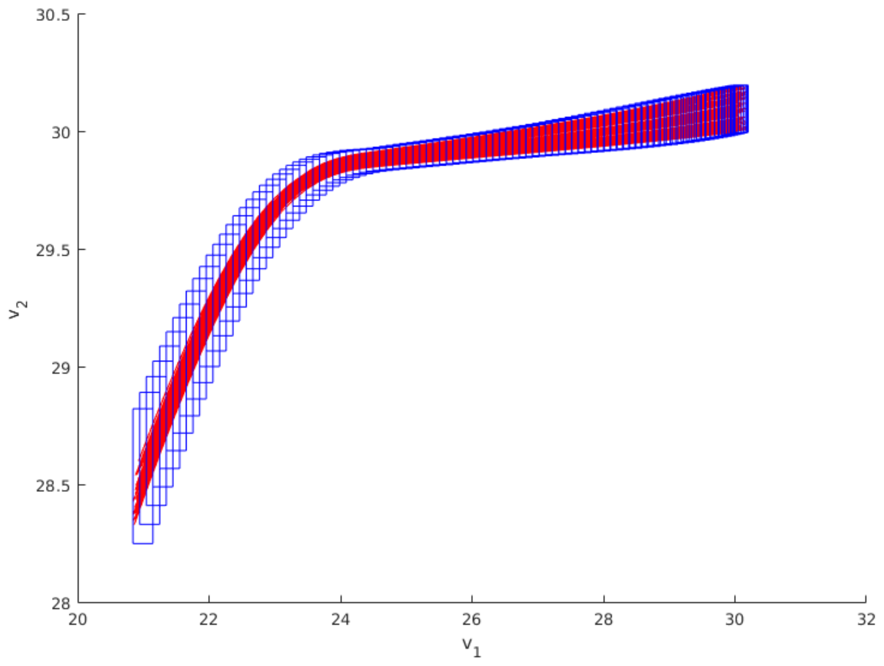

Figure 4: ***Sherlock.*** Reach sets of the ACC benchmark as obtained from Sherlock, along with 100 randomly chosen trajectories

4.2.1 ACC

The reach sets for this example has been shown in Fig 4. The initial sets shown in the figure correspond to $x_{lead}(0) \in [90, 91], x_{ego}(0) \in [10, 11]$, $v_{\text{lead}}(0) \in$ [32,32.2], $\gamma_{\text{lead}}(0) = \gamma_{\text{ego}}(0) = 0$, $v_{\text{ego}}(0) \in$ [30, 30.2]. The Sherlock could successfully verify the safety of this controller over a bounded time horizon of 5s, for the full range. The initial set for x_{lead} in the original benchmark could be broken up into, smaller sub-intervals, and then verified individually. The computation of reach sets took less to 4s to complete. A degree 1 polynomial was enough to approximate the neural network, with a maximum error of 0.12. The percentage of time spent was 66.65% , and 29.23% in Sherlock and Flow* respectively.

4.2.2 Cartpole

There are currently no results for this benchmark.

4.2.3 Sherlock-Benchmark-7

The reach sets for this example are shown in Fig 5. The Sherlock could successfully verify the safety of this controller over a bounded time horizon of 2s. A degree 2 polynomial was enough to approximate the neural network, with a maximum error of 0.09. The percentage of time spent was 95.7% , and 2.6% in Sherlock and Flow* respectively. The total execution time of the reachability computation was $41s$.

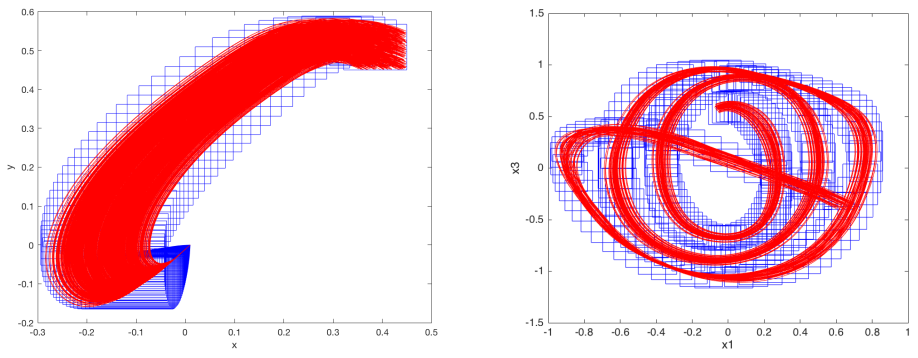

Figure 5: ***Sherlock*** Reach sets of Benchmark 7 and 9 (left to right) from Sherlock along with 100 trajectories with randomly chosen initial points

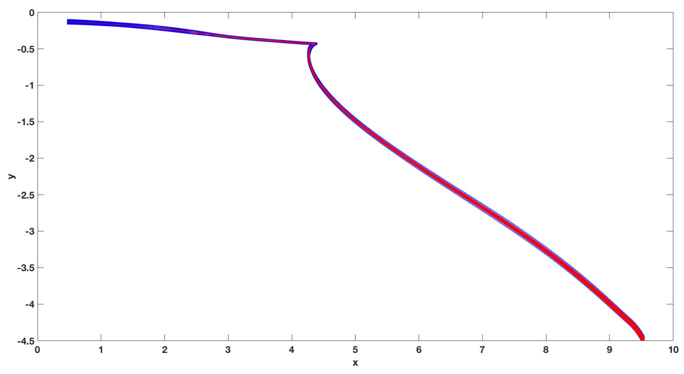

Figure 6: ***Sherlock*** Reach sets of Benchmark 10 from Sherlock along with 100 trajectories with randomly chosen initial points

4.2.4 Sherlock-Benchmark-9

The reach sets for this example are shown in Fig 5. Sherlock could successfully verify the safety of this controller over a bounded time horizon of 20s. A degree 2 polynomial was enough to approximate the neural network, with a maximum error of 0.06. The percentage of time spent was 88.1% , and 8.8% in Sherlock and Flow* respectively. The total execution time of the reachability computation was $19.2s$.

4.2.5 Sherlock-Benchmark-10

The reach sets for this example are shown in Fig 6. Sherlock could successfully verify the safety of this controller over a bounded time horizon of 10s. A degree 2 polynomial was enough to approximate the neural network, with a maximum error of 0.03. The percentage of time spent was 1.8% , and 97.3% in Sherlock and Flow* respectively. The total execution time of the reachability computation was $329.3s$.

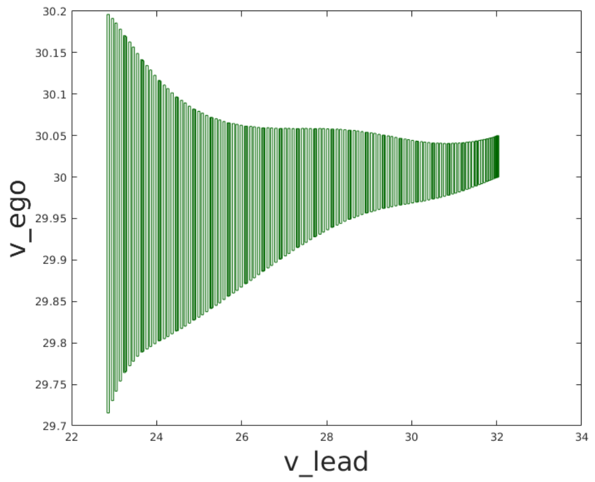

Figure 7: ***Verisig*** Reachable sets for v_{ego} and v_{lead} for the initial condition considered by Verisig.

4.3 Verisig

This section describes the Verisig results on the ACC and Cartpole benchmarks. Note that Verisig does not support ReLUs, so all networks considered in this section have *tanh* activation functions that were retrained with source data.

4.3.1 ACC

In the ACC example, we verified safety for the initial condition $x_{lead}(0) \in [90, 91], x_{ego}(0) \in [10, 10.5]$, $v_{\text{lead}}(0) \in$ [32,32.05], $\gamma_{\text{lead}}(0) = \gamma_{\text{ego}}(0) = 0$, $v_{\text{ego}}(0) \in$ [30, 30.05]. The rest of initial condition could be verified by gridding it up into pieces of similar size and verifying for each one. The reachable sets for the two velocities are shown in Figure 7. Note that Verisig used a tanh-based neural network (with three hidden layers and 20 neurons per layer), so the results are slightly different from the other tools'.

4.3.2 Cartpole

Since the controller in the Cartpole example is effectively a bang-bang controller, computing the reachable sets for this problem requires enumerating an exponentially increasing (with the length of the scenario) number of paths. Thus, we could not verify the system for the entire duration of the scenario (4s). At the same, we are able to verify safety for 0.5s. Specifically, we consider the initial set

$$x(0) \in [0, 0.01] \times v(0) \in [0, 0.01] \times \theta(0) \in [-0.05, 0.05] \times \omega(0) \in [-0.05, 0.05],$$

i.e., this is the entire initial range for θ and ω and a subset of the range for x and v. The rest of the initial set could be verified in a similar fashion.

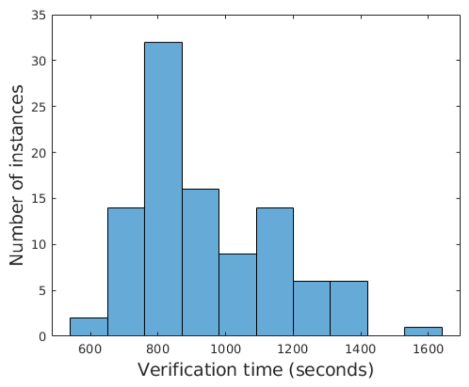

Figure 8: ***Verisig*** Histogram of the verification runtimes over 100 instances on the Cartpole example.

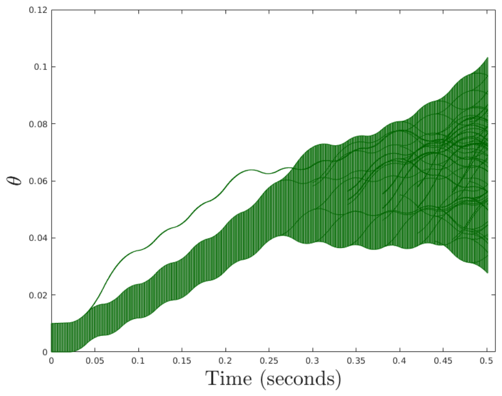

Figure 9: ***Verisig*** Reachable sets for θ for the initial condition $x(0), v(0), \theta(0), \omega(0) \in [0, 0.01]$.

To verify the above initial set, we split it into 100 subsets (we divide the ω and θ intervals into 10 subintervals each) and verify for each one. The average runtime over the 100 instances is about 947s. A histogram with the runtimes over the 100 instances is shown in Figure 8. Finally, Figure 9 shows the reachable sets for one of those instances – note that the branching introduces multiple different (possibly spurious) paths with small uncertainty each.

Note that we used a tanh-based DNN with two hidden layers with 48 and 24 neurons per layer, respectively.

4.3.3 Sherlock-Benchmark-7

There are currently no results for this benchmark.

4.3.4 Sherlock-Benchmark-9

There are currently no results for this benchmark.

4.3.5 Sherlock-Benchmark-10

There are currently no results for this benchmark.

5 Category Status and Challenges

In the first iteration of the AINNCS category at ARCH-COMP, the three tools NNV, Sherlock, and Verisig successfully analyzed different aspects of the five benchmark problems. In spite of some success analyzing the benchmarks, the primary outcome of this first iteration of the AINNCS category are challenges that arose in the competition that we discuss next.

Hybrid dynamics: The cartpole benchmark was included in part because its bang-bang controller induces a form of hybrid dynamics, either a maximum or minimum control signal is applied to the plant. This example was challenging for all tools, and as such, a primary result of the competition is that there do not currently exist any feasible methods for analyzing hybrid systems composed with neural network controllers. For tools that were able to model this, specifically Verisig, issues arose including a blow-up of possible paths in the bounded model checking. As such, approaches effective for handling hybrid dynamics need to be integrated into the approaches for analyzing these systems.

Activation function types: For the set of benchmarks proposed, all the neural network controllers contain only ReLU and linear activation functions[4]. In the future, we will add other nonlinear activation functions such as tanh and sigmoid for a more thorough comparison, as Verisig can only deal with these type of smooth functions, and NNV and Sherlock can only effectively analyze plants with ReLU and linear functions (at the time this report was written), although NNV has preliminary support for these nonlinear activations.

Neural network architectures and parameterization: When we compare the neural network architectures presented in this work with some of the networks that can be analyzed in absence of the plant, these are fairly simple, in the sense none of the networks have more than a thousand neurons, neither they exceed 5 hidden layers in their architecture. Also, the maximum number of inputs and outputs of the controllers are 5 (ACC) and 2 (Sherlock-Benchmark-10 and Cart-pole), respectively. One of the main challenges in this aspect is to analyze larger networks and more complex plants in the same NNCS, as their complexity grows very quickly when the plants are highly nonlinear with multiple inputs and outputs, as well as when the number of neurons and/or layers in the controllers increase.

Additionally, the verification results presented assumed the neural networks are fixed. However, as illustrated in part with Verisig's retraining of networks to utilize other activation functions beyond ReLUs, many different architectures and parameterization of the neural network controllers could be considered. For instance, for a fixed plant, a controller could be trained with different numbers of layers, neurons, etc. as well as activation function types, which would allow for evaluating scalability against a fixed plant, focusing more on the neural network aspects. In any event, there are state-space explosion and scalability issues to address in both the neural network controllers and plant analysis.

Time horizons: All the competition tools performed bounded (finite) time horizon reachability analysis, also known as bounded model checking. This was accomplished up to some pre-specified number of control periods k with some pre-specified sampling period for the continuous dynamics' evolution. Alternative approaches for performing unbounded (infinite) time horizon verification exist, such as those building on barrier certificates, a form of continuous analog of the classical inductive invariance proof rule. The existing methods could incorporate invariance checks on the computed reachable states to attempt to determine if the reachability analysis reaches a fixed-point (if the reachability analysis terminates, which for the class of systems considered, is not guaranteed as the reachability analysis with nonlinear plants is undecidable). However, no current methods evaluated in the competition utilize this approach, and this is a promising avenue for future work to provide guarantees beyond finite time horizons.

[4]Except for the ACC case evaluated by Verisig.

Model formats: We present the closed-loop simulations as Simulink files, where the controller is a feedforward neural network Simulink Block, and the plant dynamics are defined as a Stateflow model. We also provide them separately, having the dynamics of the plants as SpaceEx models and MATLAB functions, and the neural networks in ONNX format[5] and .mat format[6]. One of the hopes of this competition and the NNCS benchmark proposal [19] is to unify all the formats currently used in the neural network community. This is where ONNX takes an important role, since it already supports multiple frameworks like TensorFlow, Caffe2, MATLAB, and more.

6 Conclusion and Outlook

This report presents the results on the first ARCH friendly competition for closed-loop systems with neural network controllers. In the first edition of this category three tools were applied to attempt to solve five different benchmark problems, namely NNV, Sherlock, and Verisig. The problems elucidated in this paper are challenging and diverse the presented results probably provide the most complete assessment of tools for the safety verification in AINNCS. We note that each tool has unique strengths and that not all of the specificities can be highlighted within a single report. However the report provides a good overview of the intellectual progression of this rapidly growing field and it is our hope to stimulate the development of efficient and effective methods capable of being utilized within real-world applications.

We would also like to encourage other tool developers to consider participating next year. All authors agree that although the participation consumes time, we have gained unique insights that will allow us to improve in the next iteration. Particularly those items listed in the status and challenges section. Information about the competition in 2020 will be announced on the ARCH website. The reports of other categories can be found in the proceedings and on the ARCH website: cps-vo.org/group/ARCH.

7 Acknowledgments

The material presented in this report is based upon work supported by the National Science Foundation (NSF) under grant number SHF 1736323, the Air Force Office of Scientific Research (AFOSR) through contract number FA9550-18-1-0122, and the Defense Advanced Research Projects Agency (DARPA) Assured Autonomy program through contract numbers FA8750-18-C-0089 and FA8750-18-C-0090. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of AFOSR, DARPA, or NSF.

A Specification of Used Machines

A.1 M_{nnv}

- Processor: Intel Core i7-7820HQ CPU @ 2.90GHz x 4

[5]Open Neural Network Exchange: `https://github.com/onnx/onnx`

[6]Direct input format used by *NNV* without transformation.

- Memory: 32 GB
- Average CPU Mark on www.cpubenchmark.net: 9409 (full), 2070 (single thread)

A.2 $\mathbf{M}_{Sherlock}$

- Processor: Intel Core i5 @ 2.5GHz
- Memory: 8 GB

A.3 $\mathbf{M}_{Verisig}$

- Processor: Intel Core i7-7820HQ CPU @ 2.90GHz x 4
- Memory: 8 GB
- Average CPU Mark on www.cpubenchmark.net: 9409 (full), 2070 (single thread)

References

[1] M. Althoff. An introduction to CORA 2015. In *Proc. of the Workshop on Applied Verification for Continuous and Hybrid Systems*, pages 120–151, 2015.

[2] M. Althoff and D. Grebenyuk. Implementation of interval arithmetic in CORA 2016. In *Proc. of the 3rd International Workshop on Applied Verification for Continuous and Hybrid Systems*, pages 91–105, 2016.

[3] M. Althoff, D. Grebenyuk, and N. Kochdumper. Implementation of Taylor models in CORA 2018. In *Proc. of the 5th International Workshop on Applied Verification for Continuous and Hybrid Systems*, 2018.

[4] Rajeev Alur. Formal Verification of Hybrid Systems. In *Proceedings of the Ninth ACM International Conference on Embedded Software*, EMSOFT '11, pages 273–278, New York, NY, USA, 2011. ACM.

[5] S. Bak, S. Bogomolov, T. A. Henzinger, T. T. Johnson, and P. Prakash. Scalable static hybridization methods for analysis of nonlinear systems. In *Proc. of the 19th ACM International Conference on Hybrid Systems: Computation and Control*, pages 155–164, 2016.

[6] Stanley Bak, Sergiy Bogomolov, and Taylor T. Johnson. Hyst: A source transformation and translation tool for hybrid automaton models. In *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control*, HSCC '15, pages 128–133, New York, NY, USA, 2015. ACM.

[7] A. G. Barto, R. S. Sutton, and C. W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13(5):834–846, Sep. 1983.

[8] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *CoRR*, abs/1606.01540, 2016.

[9] Xin Chen, Erika Ábrahám, and Sriram Sankaranarayanan. Flow*: An analyzer for non-linear hybrid systems. In *International Conference on Computer Aided Verification*, pages 258–263. Springer, 2013.

[10] Souradeep Dutta, Xin Chen, and Sriram Sankaranarayanan. Reachability analysis for neural feedback systems using regressive polynomial rule inference. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control, HSCC 2019, Montreal, QC, Canada, April 16-18, 2019.*, pages 157–168, 2019.

[11] Souradeep Dutta, Susmit Jha, Sriram Sankaranarayanan, and Ashish Tiwari. Learning and verification of feedback control systems using feedforward neural networks. *IFAC-PapersOnLine*, 51(16):151 – 156, 2018. 6th IFAC Conference on Analysis and Design of Hybrid Systems ADHS 2018.

[12] Souradeep Dutta, Susmit Jha, Sriram Sankaranarayanan, and Ashish Tiwari. Output range analysis for deep feedforward neural networks. In Aaron Dutle, César Muñoz, and Anthony Narkawicz, editors, *NASA Formal Methods*, pages 121–138, Cham, 2018. Springer International Publishing.

[13] Krishnamurthy Dvijotham, Robert Stanforth, Sven Gowal, Timothy A. Mann, and Pushmeet Kohli. A Dual Approach to Scalable Verification of Deep Networks. *CoRR*, abs/1803.06567, 2018.

[14] R. Ivanov, J. Weimer, R. Alur, G. J. Pappas, and I. Lee. Verisig: verifying safety properties of hybrid systems with neural network controllers. *Proceedings of the 22nd International Conference on Hybrid Systems: Computation and Control*, 2019.

[15] M. Jankovic, D. Fontaine, and P. V. Kokotovic. Tora example: cascade- and passivity-based control designs. *IEEE Transactions on Control Systems Technology*, 4(3):292–297, May 1996.

[16] Guy Katz, Clark Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In Rupak Majumdar and Viktor Kunčak, editors, *Computer Aided Verification*, pages 97–117, Cham, 2017. Springer International Publishing.

[17] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. Adversarial examples in the physical world. *CoRR*, abs/1607.02533, 2016.

[18] Weibo Liu, Zidong Wang, Xiaohui Liu, Nianyin Zeng, Yurong Liu, and Fuad E. Alsaadi. A Survey of Deep Neural Network Architectures and their Applications. *Neurocomputing*, 234:11 – 26, 2017.

[19] Diego Manzanas Lopez, Patrick Musau, Hoang-Dung Tran, Joel A. Rosenfeld, and Taylor T. Johnson. Verification of closed-loop systems with neural network controllers. *CoRR*, abs/1810.01989, 2019.

[20] K. S. Narendra and K. Parthasarathy. Identification and control of dynamical systems using neural networks. *IEEE Transactions on Neural Networks*, 1(1):4–27, March 1990.

[21] S. Joe Qin and Thomas A. Badgwell. An overview of nonlinear model predictive control applications. In Frank Allgöwer and Alex Zheng, editors, *Nonlinear Model Predictive Control*, pages 369–392, Basel, 2000. Birkhäuser Basel.

[22] Stuart Russell, Daniel Dewey, and Max Tegmark. Research Priorities for Robust and Beneficial Artificial Intelligence. *AI Magazine*, 36(4):105, 2015.

[23] Peter Stone, Rodney Brooks, Erik Brynjolfsson, Ryan Calo, Oren Etzioni, Greg Hager, Julia Hirschberg, Shivaram Kalyanakrishnan, Ece Kamar, Kevin Leyton-Brown, David C. Parkes, William Press, AnnaLee Saxenian, Julie Shah, Milind Tambe, and Astro Teller. "artificial intelligence and life in 2030." one hundred year study on artificial intelligence: Report of the 2015-2016 study panel, 2016.

[24] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *CoRR*, abs/1312.6199, 2013.

[25] Hoang-Dung Tran, Patrick Musau, Diego Manzanas Lopez, Xiao Dong Yang, Luan Nguyen, Weiming Xiang, and Taylor T Johnson, editors. *Proceedings of the 7th International Conference On Formal Methods In Software Engineering, FormaliSE 2019, collocated with ICSE 2019, Montral, Canada, May 27, 2019.* ACM, 2019.

[26] Weiming Xiang and Taylor T. Johnson. Reachability analysis and safety verification for neural network control systems. *CoRR*, abs/1805.09944, 2018.

[27] Weiming Xiang, Patrick Musau, Ayana A. Wild, Diego Manzanas Lopez, Nathaniel Hamilton, Xiaodong Yang, Joel A. Rosenfeld, and Taylor T. Johnson. Verification for machine learning, autonomy, and neural networks survey. *CoRR*, abs/1810.01989, 2018.

[28] Weiming Xiang, Hoang-Dung Tran, and Taylor T. Johnson. Output reachable set estimation and verification for multi-layer neural networks. *CoRR*, abs/1708.03322, 2017.

[29] Weiming Xiang, Hoang-Dung Tran, and Taylor T. Johnson. Reachable set computation and safety verification for neural networks with relu activations. *CoRR*, abs/1712.08163, 2017.

[30] Weiming Xiang, Hoang-Dung Tran, and Taylor T. Johnson. Specification-guided safety verification for feedforward neural networks. *CoRR*, abs/1812.06161, 2018.

[31] Dong-Hae Yeom and Young Hoon Joo. Control lyapunov function design by cancelling input singularity. *International Journal of Fuzzy Logic and Intelligent Systems*, 12, 06 2012.