Neural State Classification for Hybrid Systems

Dung Phan
Department of Computer Science, Stony Brook University, USA

Nicola Paoletti
Department of Computer Science, Royal Holloway, University of London, UK

Timothy Zhang
Department of Computer Science, Stony Brook University, USA

Radu Grosu
Department of Computer Engineering, Technische Universitat Wien, Austria

Scott A. Smolka
Department of Computer Science, Stony Brook University, USA

Scott D. Stoller
Department of Computer Science, Stony Brook University, USA

CCS CONCEPTS

• **Theory of computation → Timed and hybrid models**; *Verification by model checking*; • **Computing methodologies** → *Neural networks.*

ACM Reference Format:
Dung Phan, Nicola Paoletti, Timothy Zhang, Radu Grosu, Scott A. Smolka, and Scott D. Stoller. 2019. Neural State Classification for Hybrid Systems. In *The Fifth International Workshop on Symbolic-Numeric methods for Reasoning about CPS and IoT (SNR '19), April 15, 2019, Montreal, QC, Canada.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3313149.3313372

Paper appeared in: Dung Phan, et al. Neural state classification for hybrid systems. In *International Symposium on Automated Technology for Verification and Analysis*, LNCS 11138, pp. 422–440, 2018.

EXTENDED ABSTRACT

Model checking of hybrid systems is usually expressed in terms of the following reachability problem for hybrid automata (HA) [6]: given an HA $\mathcal{M}$, a set of initial states I, and a set of unsafe states U, determine whether there exists a trajectory of $\mathcal{M}$ starting in an initial state and ending in an unsafe state. The time-bounded version of this problem considers trajectories that are within a given time bound T.

We introduce the *State Classification Problem* (SCP), a generalization of the model checking problem for hybrid systems. Let $\mathbb{B} = \{0,1\}$ be the set of Boolean values. Given an HA $\mathcal{M}$ with state space $S(\mathcal{M})$, time bound T, and set of unsafe states $U \subset S(\mathcal{M})$, the SCP problem is to find a function $F^* : S(\mathcal{M}) \to \mathbb{B}$ such that for all $s \in S(\mathcal{M})$, $F^*(s) = 1$ if $\mathcal{M} \models \text{Reach}(U, s, T)$, i.e., if it is possible for $\mathcal{M}$, starting in s, to reach a state in U within time T; $F^*(s) = 0$ otherwise. A state $s \in S(\mathcal{M})$ is called *positive* if $F^*(s) = 1$. Otherwise, s is *negative*. We call such a function a *state classifier*.

State classification is also useful in at least two other contexts. First, due to random disturbances, a hybrid system may restart in a random state outside the initial region, and we may wish to check the system's safety from that state. Secondly, a classifier can be used for *online model checking* [10], where in the process of monitoring a system's behavior, one would like to determine, in real-time, the fate of the system going forward from the current (non-initial) state.

This paper shows how deep neural networks (DNNs) can be used for state classification, an approach we refer to as *Neural State Classification* (NSC). An NSC classifier is subject to *false positives* (FPs) and, more importantly, *false negatives* (FNs). An FP occurs when a state s is deemed positive when it is actually negative, and, likewise, an FN occurs when s is deemed negative when it is actually positive.

A well-trained NSC classifier offers high accuracy, runs in constant time (approx. 1 ms in our experiments), and takes constant space (e.g., a DNN with l hidden layers and n neurons only requires functions of dimension $l \cdot n$ for its encoding). This makes NSC classifiers very appealing for applications such as online model checking, a type of analysis subject to strict time and space constraints.

Our approach can also classify states of *parametric* HA by encoding each parameter as an additional input to the classifier. This makes NSC more versatile than state-of-the-art hybrid system reachability tools, which provide little or no support for parametric analysis [3, 4].

The NSC method is summarized in Figure 1. We train the state classifier using supervised learning, where the training examples are derived by sampling the state and parameter spaces according to some distribution. Reachability values for the examples are computed by invoking an oracle, i.e., an hybrid system model checker [4] or a simulator when the system is deterministic.

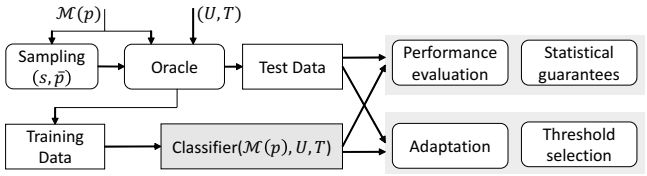


Figure 1: Overview of the NSC approach.

We evaluate a trained state classifier by estimating its accuracy, false-positive rate, and false-negative rate (together with their confidence intervals) on a test dataset of fresh samples. This allows us to quantify how well the classifier extrapolates to unseen states, i.e., the probability that it correctly predicts reachability for any state.

Inspired by statistical model checking [8], we also provide statistical guarantees through sequential hypothesis testing to certify (up to some confidence level) that the classifier meets prescribed accuracy levels on unseen data.

We also consider two tuning methods that can reduce and virtually eliminate false negatives: a new method called *falsification-guided adaptation* that iteratively re-trains the classifier with false negatives

SNR '19, April 15, 2019, Montreal, QC, Canada

ACM ISBN 978-1-4503-6697-7/19/04…$15.00
https://doi.org/10.1145/3313149.3313372

found through adversarial sampling; and *threshold selection*, which adjusts the NN's classification threshold to favor FPs over FNs.

We have applied NSC to six nonlinear hybrid system benchmarks, achieving an accuracy of 99.25% to 99.98%, and a false-negative rate of 0.0033 to 0, which we further reduced to 0.0015 to 0 after tuning the classifier. We believe that this level of accuracy is acceptable in many practical applications, and that these results demonstrate the promise of the NSC approach.

In the rest of this extended abstract, we provide more details about the NSC approach and discuss experimental results.

Generation of Training Data and Test Data

We consider three sampling methods for generation of training data and test data. Note that we do not sample states in U, which are trivially positive.

Uniform sampling, where every state is equi-probable.

Dynamics-aware sampling, which samples a state according to the estimated probability that the state is visited in any time-bounded evolution of the system. This approach estimates a state distribution through isotropic random walks of the HA; i.e., by choosing uniformly at random, at each step of the simulation, the next transition from those available.

Balanced sampling, which seeks to improve accuracy by drawing a balanced number of positive and negative states. This method is especially useful when the set of unsafe states U is a small portion of the overall state space. In this case, uniform sampling would produce imbalanced datasets with an insufficient number of positive samples, leading to a classifier with poor accuracy. For this purpose, we introduce a method for generating arbitrarily large sets of positive samples based on the construction and subsequent simulation of *reverse hybrid automata* (discussed below).

Oracles

Given a state (sample) s of an HA $\mathcal{M}$, an NSC *oracle* is a procedure for labeling s; i.e., for deciding whether $\mathcal{M} \models \mathsf{Reach}(U,s,T)$.

Reachability checker. For nonlinear HA, NSC uses dReal [4], an SMT solver that supports bounded model checking of such HA. dReal provides sound unsatisfiability proofs, but satisfiability is approximated up to a user-defined precision (δ-satisfiability).

Simulator. For deterministic systems, we implemented a simulator based on MATLAB's `ode45` variable-step ODE solver.

Backwards simulator. This is not an oracle per se, but is central to the balanced sampling method. For backwards simulation, we first need to construct the reverse HA $\overleftarrow{\mathcal{M}}$ of $\mathcal{M}$. This construction is based on a novel definition of reverse HA which generalizes the one for rectangular HAs given in [6] and ensures that reverse trajectories obtained by simulating $\overleftarrow{\mathcal{M}}$ are consistent with $\mathcal{M}$. See [9] for further details. To generate a positive sample in the context of balanced sampling, we then just need to invoke the backwards simulator starting from an unsafe state.

A-posteriori Statistical Guarantees

Given the infeasibility of training machine-learning models with guaranteed accuracy on unseen data, we provide statistical guarantees *a posteriori*, i.e., after training. Inspired by statistical approaches to model checking [8], we employ hypothesis testing, Wald's sequential probability ratio test (SPRT) in particular, to certify that our classifiers meet prescribed levels of accuracy and FN/FP rates.

We provide guarantees of the form $P_{\mathsf{A}} \geq \theta_{\mathsf{A}}$ (i.e., the true accuracy value is above θ_{A}), $P_{\mathsf{FN}} \leq \theta_{\mathsf{FN}}$ and $P_{\mathsf{FP}} \leq \theta_{\mathsf{FP}}$ (i.e., the true rate of FNs and FPs are below θ_{FN} and θ_{FP}, respectively). Because of their statistical nature, such guarantees are precise up to arbitrary bounds on the probability of Type-I error and Type-II errors, respectively denoted by $\alpha,\beta \in (0,1)$.

Reducing False Negatives Via Falsification

For NSC, it is important to reduce the rate of FNs, the most serious errors from a safety- critical perspective. One approach is based on tuning the classification threshold of the NN. We introduce another, more systematic, approach that works by retraining the classifier with unseen FN samples found in the test stage, thereby making the classifier more conservative. For this purpose, we devised a whitebox *falsification-guided adaptation algorithm* that, at each iteration, searches for FNs using *adversarial sampling*; i.e., by solving an optimization problem that maximizes the disagreement between NN-predicted and true reachability values. The optimization problem exploits the knowledge of an NN classifier function (whitebox approach). FNs found in this way are used to retrain the classifier. The algorithm iterates until the falsifier cannot find any more FNs or until a maximum number of iterations is reached. In our experiments, we used genetic algorithms to solve this optimization problem, but other nonlinear optimization methods are equally supported.

In [9], we proved that under some assumptions about the performance of the classifier and the falsifier, our algorithm converges to an empty set of FNs. Although it may be difficult in practice to guarantee that these assumptions are satisfied, our algorithm performs reasonably well in practice, as shown in the next section.

Experimental Evaluation

We evaluated our NSC approach on six hybrid-system benchmarks: spiking neuron [2], inverted pendulum, a quadcopter system [5], a cruise controller [2], a powertrain model [7], and a helicopter model [1]. These case studies represent a broad spectrum of hybrid systems and varying degrees of complexity (deterministic, nondeterministic, nonlinear dynamics, 2–29 variables, 1–6 modes, 1–11 transitions). We used MATLAB's `train` function to learn the NSC neural networks.

We considered the following types of classifiers: sigmoid DNNs (**DNN-S**) with 3 hidden layers of 10 neurons each; shallow sigmoid NNs (**SNN**), with one hidden layer of 20 neurons; ReLU DNNs (**DNN-R**), with 3 hidden layers of 10 neurons each and rectified linear unit (ReLU) activation function for the hidden layers; support vector machines with radial kernel (**SVM**); binary decision trees (**BDT**); and a simple nearest neighbor classifier (**NBOR**).

We learned the classifiers from relatively small datasets, using training sets of 20K samples and test sets of 10K samples, except where noted otherwise. The NN architecture (numbers of layers and neurons) was chosen empirically. To avoid overfitting, we did not tune the architecture to optimize the performance for our data.

Performance evaluation. Table 1 shows accuracy and FN rate for all classifiers and case studies, using uniform and balanced sampling. We obtain very high classification accuracy for neuron, pendulum,

		Neuron		Pendulum		Quadcopter		Cruise		Powertrain		Helicopter		
		Acc	FN	Acc	FN	Acc	FN	Acc	FN	Acc	FN	Acc	FN	
DNN-S		**99.81**	**0.1**	**99.98**	**0**	99.83	0.1	99.95	0.01	**96.68**	1.28	**98.49**	**0.84**	**Uniform**
DNN-R		99.52	0.29	99.93	0.04	**99.89**	**0.06**	**99.98**	**0**	96.21	**1.08**	98	0.96	
SNN		99.17	0.43	99.81	**0**	99.85	0.08	99.84	0.15	96.02	1.37	97.69	1.25	
SVM		98.73	0.75	99.84	**0**	97.33	0.69	99.88	0.1	92.26	3.48	95.58	2.42	
BDT		99.3	0.37	99.6	0.17	99.52	0.2	99.84	0.08	95.59	2.19	80.07	9.8	
NBOR		97.03	1.22	99.69	0.14	99.53	0.25	99.49	0.33	71.44	14.51	67.39	16.98	
		Acc	FN	Acc	FN	Acc	FN	Acc	FN	Acc	FN	Acc	FN	
DNN-S		**99.83**	**0.12**	**99.89**	**0**	**99.82**	0.04	99.94	**0**	**97.2**	**0.86**	**98.24**	**0.79**	**Balanced**
DNN-R		99.48	0.24	99.63	0.01	99.67	0.09	**99.95**	**0**	96.07	1.24	97.91	1.2	
SNN		98.89	0.69	99.2	**0**	99.49	**0.01**	99.6	**0**	95.21	1.79	97.58	1.16	
SVM		98.63	0.78	99.37	**0**	96.93	0.2	99.61	**0**	91.84	3.3	95.36	1.85	
BDT		99.07	0.45	99.46	0.05	99.36	0.22	99.9	0.03	95.86	2.4	79.03	10.26	
NBOR		96.95	1.62	99.51	0.04	99.11	0.56	99.47	0.11	71.33	13.99	65.18	17.48	

Table 1: Empirical accuracy (Acc) and FN rate of the state classifiers for each case study, classifier type, and sampling method. Values are in percentages. For each measure and sampling method, the best result is highlighted in bold.

quadcopter and cruise. For these case studies, DNN-based classifiers registered the best performance, with accuracy values ranging between 99.48 % and 99.98 %, and FN rates between 0.24% and 0%.

In contrast, the accuracy for the helicopter and powertrain models is poor if we use only 20K training samples. These models are indeed particularly challenging, owing to their high dimensionality (helicopter) and highly nonlinear dynamics (powertrain). Using 1M samples instead of 20K for the helicopter case study, the accuracy jumps from 98.49% to 99.92%, and the FN rate decreases from 0.84% (20K) to 0.04%. For powertrain, the accuracy increases from 96.68% to 99.25%, and the FN rate decreases from 1.28% to 0.33%.

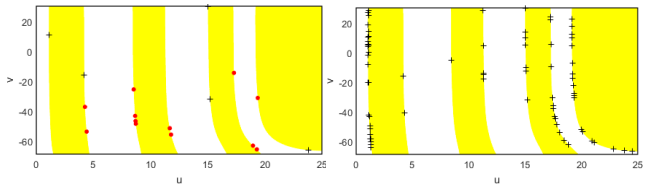
In general, we found that the NN-based classifiers have superior accuracy compared to support vector machines and binary decision trees. As expected, the nearest-neighbor method demonstrated poor prediction capabilities.

We omit results for dynamics-aware sampling, because using dynamics-aware sampling alone yields unbalanced datasets unsuitable for training. Thus, to evaluate this sampling method, we generated training data with a combination of dynamics-aware sampling and either uniform or balanced sampling. Test data consists exclusively of dynamics-aware samples. With these settings, we obtained results similar to Table 1.

Parametric analysis. We show that NSC is an effective approach for parametric systems, being able to classify states in models with parameter values not seen during training. For this purpose, we modified the neuron model by introducing different numbers of parameters. Using DNN-S and 110K training samples, we achieve very high accuracy ($\geq 99.7\%$) for up to two parameters. For three to five parameters, the accuracy decreases but remains relatively high (around 98%), suggesting that larger training sets are required for these cases. Indeed, the input space grows exponentially in the number of parameters, while we kept the size of the training set constant.

Statistical guarantees. We use SPRT (with $\alpha = \beta = 0.01$) to provide statistical guarantees for our case studies. We assess two properties certifying that the *true* (not the empirical) accuracy and FNs meet given performance levels: $P_{\mathsf{A}} \geq 99.7\%$, and $P_{\mathsf{FN}} \leq 0.2\%$.

We found that the only classifier that guarantees these performance levels for all case studies is the sigmoid DNN. We also determined that a small number of samples suffices to obtain statistical guarantees with the given strength: only 6.25% of the tests needed more than 10K samples to decide the property.



(a) Before adaptation **(b) After adaptation**

Figure 2: Effects of adaptation on the DNN-S for the neuron case study. The white region is the predicted negative region. The yellow region is the predicted positive region. Red dots are FN samples. Crosses are FP samples.

Falsification-guided adaptation. We evaluate the benefits of adaptation by incrementally adapting the trained NNs with FN samples. To measure the impact of adaptation, we tested the DNNs on 10K samples after each iteration of adaptation. For the neuron, quadcopter, and helicopter case studies, our algorithm effectively reduces the FN rate to 0% after only 5-10 iterations, at the cost of a slight increase in the FP rate. For powertrain, the falsifier needs more iterations (150) to reduce the FN rate, which decreases from 0.33% to 0.15%.

Figure 2 visualizes the effects of adaptation on the **DNN-S** classifier for the neuron case study. Fig. 2 (a) shows the predictions of the DNN after training with 20K samples. Fig. 2 (b) shows the predictions of the DNN after adaptation. We see that adaptation expands the predicted positive region so as to enclose all previous FN samples; i.e., they are correctly re-classified as positive. The enlarged positive region also means the adapted DNN is more conservative, producing more FPs as shown in Fig. 2 (b).

ACKNOWLEDGMENTS

AFOSR Grant FA9550-14-1-0261, NSF Grants CPS-1446832, IIS-1447549, CNS-1445770, CNS-1421893, and CCF-1414078, FWF-NFN RiSE Award, and ONR Grant N00014-15-1-2208.

REFERENCES

[1] Stanley Bak and Parasara Sridhar Duggirala. 2017. Rigorous simulation-based analysis of linear hybrid systems. In *TACAS*. Springer, 555–572.
[2] Xin Chen et al. 2015. A benchmark suite for hybrid systems reachability analysis. In *NFM*. Springer, 408–414.
[3] Goran Frehse et al. 2011. SpaceEx: Scalable verification of hybrid systems. In *CAV*. Springer, 379–395.

[4] Sicun Gao, Soonho Kong, and Edmund M Clarke. 2013. dReal: An SMT solver for nonlinear theories over the reals. In *International Conference on Automated Deduction*. Springer, 208–214.
[5] Andrew Gibiansky. 2012. Quadcopter dynamics and simulation. http://andrew.gibiansky.com/blog/physics/quadcopter-dynamics/
[6] Thomas A. Henzinger et al. 1995. What's Decidable About Hybrid Automata?. In *STOC*. 373–382.
[7] Xiaoqing Jin et al. 2014. Powertrain control verification benchmark. In *HSCC*. 253–262.
[8] Axel Legay, Benoît Delahaye, and Saddek Bensalem. 2010. Statistical Model Checking: An Overview. In *RV*. Springer, 122–135.
[9] Dung Phan, Nicola Paoletti, Timothy Zhang, Radu Grosu, Scott A Smolka, and Scott D Stoller. 2018. Neural state classification for hybrid systems. In *International Symposium on Automated Technology for Verification and Analysis*. Springer, 422–440.
[10] Koushik Sen, Grigore Rosu, and Gul Agha. 2004. Online efficient predictive safety analysis of multithreaded programs. In *TACAS*, Vol. 2988. Springer, 123–138.