Proceedings of the
2nd Workshop on Artificial Intelligence and Formal Verification, Logics, Automata and Synthesis (OVERLAY),
September 25, 2020

Bayesian Neural Predictive Monitoring

Luca Bortolussi[1,4], Francesca Cairoli[1], Nicola Paoletti[2], S. A. Smolka[3], and S. D. Stoller[3]

[1]Department of Mathematics and Geoscience, University of Trieste, Trieste
[2]Department of Computer Science, Royal Holloway University, London
[3]Department of Computer Science, Stony Brook University, USA
[4]Modelling and Simulation Group, Saarland University, Germany

Abstract

Neural State Classification (NSC) is a recently proposed method for runtime predictive monitoring of Hybrid Automata (HA) using deep neural networks (DNNs). NSC predictors have very high accuracy, yet are prone to prediction errors that can negatively impact reliability. To overcome this limitation, we present *Neural Predictive Monitoring* (NPM), a technique that complements NSC predictions with estimates of the predictive uncertainty. These measures yield principled criteria for the rejection of predictions likely to be incorrect, without knowing the true reachability values. We also present an active learning method that significantly reduces the NSC predictor's error rate and the percentage of rejected predictions. NPM is based on the use of Bayesian techniques, Bayesian Neural Networks and Gaussian Processes, to learn respectively the predictor and the rejection rule. Our approach is highly efficient, with computation times on the order of milliseconds, and effective, managing in our experimental evaluation to successfully reject almost all incorrect predictions.

1 Problem Statement

Hybrid systems are a central model for many safety-critical cyber-physical system applications [2]. Their verification typically amounts to solving a hybrid automata (HA) reachability checking problem [11]. Given an HA $\mathcal{M}$, with state space X, time-bounded *reachability checking* is concerned with establishing whether, given an initial state x and a set of target states $D \subseteq X$ – typically a set of unsafe states to avoid – the HA admits a trajectory starting from x that reaches D within a time T. It is denoted as $\mathcal{M} \models \mathsf{Reach}(D, x, T)$. Due to its high computational cost, reachability checking is usually limited to design-time (offline) analysis [7]. Our focus is on the online analysis of hybrid systems and, in particular, on the *predictive monitoring* (PM) problem [9], i.e., the problem of predicting, *at runtime*, whether a violation can be reached from the current state within a given time bound. We aim to derive a function that can solve the PM problem. In solving this problem, we assume a distribution $\mathcal{X}$ of HA states and seek the monitor that predicts HA reachability with minimal error probability w.r.t. $\mathcal{X}$.

Problem 1 (Predictive monitoring for HA reachability)**.** *Given an HA $\mathcal{M}$ with state space X, a distribution $\mathcal{X}$ over X, a time bound T and set of unsafe states $D \subseteq X$, find a function $F^* : X \to \{0, 1\}$ that minimizes the probability that F^* makes a mistake in predicting the truth value of $\mathcal{M} \models \mathsf{Reach}(D, x, T)$:*

$$Pr_{x \sim \mathcal{X}} \left(F^*(x) \neq \mathbf{1}(\mathcal{M} \models \mathsf{Reach}(D, x, T)) \right),$$

where $\mathbf{1}$ *is the indicator function. A state* $x \in X$ *is called* positive *w.r.t. a predictor* $F : X \to \{0, 1\}$ *if* $F(x) = 1$*. Otherwise,* x *is called* negative *(w.r.t.* F*).*

Finding F^*, i.e., finding a function approximation with minimal error probability, is indeed a classical machine learning problem, a *supervised classification* problem in particular, with F^* being the *classifier*, i.e., the function mapping HA state inputs x into one of two classes: positive or negative.

Training set. In supervised learning, one minimizes a measure of the empirical prediction error w.r.t. a *training set*. In our case, the training set Z_t is obtained from a finite sample X_t of $\mathcal{X}$ by labelling the training inputs $x \in X_t$ using some reachability oracle, that is, a hybrid automata reachability checker like [1, 6, 8, 10]. Hence, given a sample X_t of $\mathcal{X}$, the training set is defined by $Z_t = \{(x, \mathbf{1}(\mathsf{Reach}(D, x, T)) \mid x \in X_t\}$.

Prediction errors. No machine learning technique can completely avoid prediction errors. Therefore, we have to deal with predictive monitors F that are prone to prediction errors, which are of two kinds: *false positives* (FPs) and *false negatives* (FNs). These errors are respectively denoted by predicates $fn(x)$ and $fp(x)$, whereas $pe(x) = fn(x) \vee fp(x)$ denotes a generic prediction error.

Let f be the discriminant function associated to the classifier F, i.e., the function that maps the inputs into class likelihoods. A central objective of this work is to derive, given a predictor f, a rejection criterion R_f able to identify states x that are wrongly classified by F, i.e., FNs and FPs, without knowing the true reachability value of x. Further, R_f should be optimal, that is, it should ensure minimal probability of rejection errors w.r.t. the state distribution $\mathcal{X}$.

Our solution relies on enriching each prediction with a measure of predictive uncertainty: given f, we define a function $u_f : X \to U$ mapping an HA state $x \in X$ into some measure $u_f(x)$ of the uncertainty of f about x. The only requirements are that u_f must be point-specific and should not use any knowledge about the true reachability value. The function u_f is then used to build an optimal error detection criterion, as explained below.

Problem 2 (Uncertainty-based error detection)**.** *Given a reachability predictor* f*, a distribution* $\mathcal{X}$ *over HA states* X*, a predictive uncertainty measure* $u_f : X \to U$ *over some uncertainty domain* U*, and a kind of error* $e \in \{pe, fn, fp\}$*, find an optimal error detection rule* $G^*_{f,e} : U \to \{0, 1\}$*, i.e., a function that minimizes the probability that* $G^*_{f,e}$ *makes a mistake in recognizing errors of type* e*:*

$$Pr_{x \sim \mathcal{X}}(e(x) \neq G^*_{f,e}(u_f(x))).$$

As for Problem 1, we can obtain a sub-optimal solution $G_{f,e}$ to Problem 2 by expressing the latter as a supervised learning problem, where the inputs are, once again, sampled according to $\mathcal{X}$ and labelled using a reachability oracle. These observations need to be independent from the above introduced training set Z_t, i.e., the set used to learn the reachability predictor f.

For $e \in \{pe, fp, fn\}$, the final rejection rule $R_{f,e}$ for detecting HA states where the reachability prediction should not be trusted, and thus rejected, is readily obtained by the composition of the uncertainty measure and the error detection rule $R_{f,e} = G_{f,e} \circ u_f : X \to \{0, 1\}$, where $R_{f,e}(x) = 1$ if the prediction on state x is rejected; $R_{f,e}(x) = 0$, otherwise.

Validation set. As done for the training set, a set Z_v is obtained from a sample X_v of $\mathcal{X}$, independent from X_t. Given an uncertainty measure u_f, we build a *validation set* $W^e_v = \{(u_f(x), \mathbf{1}(e(x))) \mid x \in X_v\}$ which is used to learn $G_{f,e}$. The inputs of W^e_v, referred to as U_v, are the uncertainty measures evaluated on X_v: $U_v = \{u_f(x) \mid x \in X_v\}$. Each input $u_f(x) \in U_v$ is then labelled respectively with 1 or 0 depending on whether or not the classifier f makes an error of type e.

A graphical scheme of the full NPM pipeline is presented in 1.

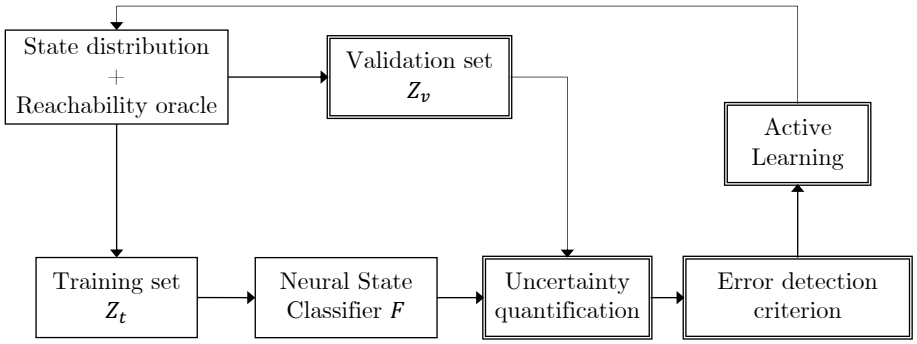

Figure 1: Overview of the NPM framework. Training of the neural state classifier F and retraining via active learning are performed offline. The only components used at runtime are the classifier F and the error-detection criterion.

2 Methods

In the NSC method of [14] deep neural networks are used to solve Problem 1. In this work, we rely on Bayesian neural networks (BNN) [5] which combine the best of probabilistic models and neural networks by learning a posterior distribution over the weights of the neural network. BNNs benefit from the statistical guarantees of probabilistic models and from the ability of neural networks to be universal function approximators. In a nutshell, a *prior* over the parameters is used as input to a neural network, the output of the neural network is then used to compute the *likelihood* – with a specified distribution – and from these it is possible to compute the *posterior* distribution of the parameters, either by Hamiltonian Monte Carlo (HMC) sampling [4] or by Variational Inference (VI) [5]. The rationale behind this choice is that BNNs are well suited to solve Problem 2 as they are able to capture predictive uncertainty via the *posterior predictive distribution* [5], which accounts for the uncertainty about the parameters. The function u_f maps a point $x \in X$ to the mean μ_x and the standard deviation σ_x of the empirical approximation of the predictive distribution.

Bayesian uncertainty measure. Given observations Z_t and a BNN $f_{\mathbf{w}}$ with $\mathbf{w} \sim p(\mathbf{w}|Z_t)$, we define the Bayesian uncertainty measure $u_f : X \to U \subseteq \mathbb{R}^2$ as the function mapping inputs x into the empirical mean, μ_x, and variance, σ_x^2, of the predictive distribution $p(\hat{y} \mid x, Z_t)$, where $\hat{y}$ is the class predicted by f_w, where w is a sample from the posterior $p(\mathbf{w} \mid Z_t)$. Formally, $\forall x \in X$, $u_f(x) = (\mu_x, \sigma_x^2)$.

An unseen input x must have sufficiently low uncertainty values in order for the prediction to be accepted. We seek to find those uncertainty values that optimally separate the points in U_v – the validation set – in relation to their classes, that is, separate points yielding errors from those that do not. Optimal thresholds can be automatically identified by solving an additional supervised learning problem, in this work we leverage Gaussian Process classification (GPC) [15].

2.1 Active Learning

As the accuracy of a classifier increases with the quality and the quantity of observed data, adding samples to Z_t will generate a more accurate predictor, and similarly, adding samples to W_v will lead to more precise error detection. Ideally, one wants to maximize accuracy while using the least possible amount of additional samples, because obtaining labeled data is expensive (in NPM, labelling each sample entails solving a reachability checking problem), and the size of the datasets affects the complexity and the dimension of the problem. Therefore, to improve the accuracy of our learning models efficiently, we need a strategy to identify the most "informative" additional samples.

For this purpose, we propose an *uncertainty-aware active learning* solution, where the re-training points are derived by first sampling a large pool of unlabeled data, and then considering only those points

where the current predictor f is still uncertain according to the rejection rule R_f.

Active learning algorithm. We query the HA reachability checker to label the points rejected by R_f and we divide them into two groups: one group is added to the training set Z_t, producing the augmented dataset Z_t^a; the other is added to the validation set Z_v, producing Z_v^a. The first step consists in retraining the reachability predictor on Z_t^a. Let f_a denote the new predictor. The second step requires extracting the augmented validation dataset W_v^a from Z_v^a. To do so, we must first train a new uncertainty measure u_{f_a} so as to reflect the new predictor f_a. Then, the new error detection rule G_{f_a} is trained on W_v^a. In conclusion, this process leads to an updated rejection rule $R_{f_a} = G_{f_a} \circ u_{f_a}$, which is expected to have a reduced rate of incorrect rejections.

A frequentist approach is also possible [16]: the NSC is a deterministic Neural Network, the uncertainty is measured using the theory of conformal predictions [3, 13, 17] and the rejection rule is described by a support vector classifier [5].

3 Experimental Results

We experimentally evaluate the proposed method on a benchmark of six hybrid system models with varying degrees of complexity. We consider four deterministic case studies: the model of the spiking neuron (SN) action potential [14] and the classic inverted pendulum (IP) on a cart, which are two-dimensional models with non-linear dynamics, the artificial pancreas (AP) [12], which is a six-dimensional non-linear model, and the helicopter model (HC) [14], which is a linear model with 29 state variables. In addition, we analyze two non-deterministic models with non-linear dynamics: a cruise controller (CC) [14], whose input space has four dimensions, and a triple water tank (TWT)[1], which is a three-dimensional model.

Performance measures. We want our method to be capable of working at runtime, which means it must be extremely fast in making predictions and deciding whether to trust them. The time required to train the reachability predictor and the error detection rule does not affect its runtime efficiency, as it is performed in advance (offline) only once. Furthermore, to avoid overfitting, we did not tune the architecture (i.e., number of neurons and hidden layers) to optimize the performance for our data and, for the sake of simplicity, we choose the same architecture for all the case studies. However, each case study requires a fine tuning of the the inference hyper-parameters. VI parameters differ from HMC hyper-parameters. Keeping that in mind, the relevant performance metrics for NPM are the *accuracy of the reachability predictor* F, the *error detection rate*, that measures the proportion of errors made on the test set by F that are actually recognized by R_f, and the overall *rejection rate* of the rejection rule R_f, that measures the overall proportion of test points rejected by R_f. Clearly, we want our method to be reliable and thus, detect the majority of prediction errors (high detection rate) without being overly conservative, i.e., keeping a low rejection rate, in order not to reduce its effectiveness.

Computational costs. Training an NPM requires training the state classifier, generating the datasets W_v, which requires computing the uncertainty values for each point in Z_v, and finally training the error rejection rule. All these steps are performed offline. On the other hand, the time required to evaluate the NPM, on a given test input x_*, is composed of the time needed to make a prediction and the time needed to choose whether to accept it or not, which take on the order of milliseconds. NPM's efficiency is not directly affected by the complexity of the system under analysis but only by the complexity of the underlying learning problems.

Evaluation. We consider three configurations: the *initial* configuration, where the predictor and error detection rules are derived via supervised learning; the *active* configuration, where the initial models are retrained via our uncertainty-aware active learning; and the *passive* configuration, where the initial models are retrained using a uniform sampling strategy to augment the dataset and with the same number

[1] `http://dreal.github.io/benchmarks/networks/water/`

of observations of the active configuration. In this way, we can evaluate the benefits of employing an uncertainty-based criterion to retrain our models.

Our experimental evaluation demonstrates that our reachability predictors attain high accuracies, consistently above 97.4% and the benefits of active learning are visible from an overall reduction of the rejection rate and an overall increase in the state-classifier accuracy compared to the passive approach. Furthermore, VI scales better than HMC with respect to the dimension of the system and we observe that on average VI outperforms HMC: the initial VI approach yield an NSC accuracy of 98.95%, a rejection rate of 5.19% and a recognition rate of 86.18%. The passive results introduce only minor improvements, whereas active learning yields a significant reduction in the rejection rate (from 5.19% to 2.36%), an increase in the NSC accuracy (from 98.95% to 99.32%) and an increase in the overall recognition rate (from 86.18% to 91.85%).

		Acc.	**Det. rate**	**Rej. rate**		**Acc.**	**Det. rate**	**Rej. rate**
Artificial Pancreas (AP)								
Initial	VI	99.29	95.77	5.17	HMC	98.95	98.09	8.32
Active	VI	99.71	96.55	1.75	HMC	99.38	90.32	4.17
Passive	VI	99.47	100.00	3.29	HMC	95.12	100.00	26.79
Inverted Pendulum (IP)								
Initial	VI	99.58	80.95	1.68	HMC	87.45	100.00	24.72
Active	VI	99.58	85.71	1.31	HMC	99.15	87.06	4.17
Passive	VI	99.71	75.81	1.13	HMC	98.49	100.00	13.02
Cruise Controller (CC)								
Initial	VI	99.01	97.98	5.53	HMC	97.22	99.64	8.41
Active	VI	99.84	100.00	1.25	HMC	99.47	94.34	3.14
Passive	VI	99.74	100.00	1.46	HMC	95.75	97.03	8.42
Triple Water Tank (TWT)								
Initial	VI	99.60	77.50	1.11	HMC	97.50	96.80	5.04
Active	VI	99.67	84.85	1.61	HMC	99.20	95.00	3.70
Passive	VI	99.50	75.40	20.32	HMC	91.86	52.58	11.31
Helicopter (HC)								
Initial	VI	98.14	88.71	13.64	HMC	97.74	89.82	14.71
Active	VI	98.90	92.86	1.98	HMC	97.74	73.01	7.06
Passive	VI	98.66	87.54	9.94	HMC	97.77	65.47	6.40
Spiking Neuron (SN)								
Initial	VI	98.18	91.21	7.91	HMC	98.32	74.40	9.89
Active	VI	98.20	91.11	6.26	HMC	98.89	87.38	5.91
Passive	VI	98.20	98.52	14.57	HMC	98.21	74.86	14.85

Table 1: Comparison of initial and active approaches on an illustrative case study (the artificial pancreas). Measures of performance are: **Acc.** denotes the NSC accuracy, **Det. rate** denotes the error detection rate and **Rej. rate** denotes the rejection rate.

4 Conclusion

Our approach overcomes the computational footprint of reachability checking (infeasible at runtime), while improving on traditional runtime verification by being able to detect future violations in a preemptive way. Among future directions, we plan to extend our approach to support more complex and real-world systems that include noise and partial observability.

References

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

[2] R. Alur. Formal verification of hybrid systems. In *Proceedings of the Ninth ACM International Conference on Embedded Software (EMSOFT)*, pages 273–278, Oct 2011.

[3] V. Balasubramanian, S.-S. Ho, and V. Vovk. *Conformal prediction for reliable machine learning: theory, adaptations and applications*. Newnes, 2014.

[4] M. Betancourt. A conceptual introduction to hamiltonian monte carlo. *arXiv preprint arXiv:1701.02434*, 2017.

[5] C. M. Bishop. *Pattern recognition and machine learning*. Springer, 2006.

[6] S. Bogomolov, M. Forets, G. Frehse, K. Potomkin, and C. Schilling. Juliareach: a toolbox for set-based reachability. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, pages 39–44, 2019.

[7] T. Brihaye, L. Doyen, G. Geeraerts, J. Ouaknine, J.-F. Raskin, and J. Worrell. On reachability for hybrid automata over bounded time. In L. Aceto, M. Henzinger, and J. Sgall, editors, *Automata, Languages and Programming*, pages 416–427, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

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

[9] X. Chen and S. Sankaranarayanan. Model predictive real-time monitoring of linear systems. In *Real-Time Systems Symposium (RTSS), 2017 IEEE*, pages 297–306. IEEE, 2017.

[10] S. Gao, S. Kong, and E. M. Clarke. dreal: An smt solver for nonlinear theories over the reals. In *International conference on automated deduction*, pages 208–214. Springer, 2013.

[11] T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata? *Journal of computer and system sciences*, 57(1):94–124, 1998.

[12] N. Paoletti, K. S. Liu, S. A. Smolka, and S. Lin. Data-driven robust control for type 1 diabetes under meal and exercise uncertainties. In *International Conference on Computational Methods in Systems Biology*, pages 214–232. Springer, 2017.

[13] H. Papadopoulos. Inductive conformal prediction: Theory and application to neural networks. In *Tools in artificial intelligence*. InTech, 2008.

[14] D. Phan, N. Paoletti, T. Zhang, R. Grosu, S. A. Smolka, and S. D. Stoller. Neural state classification for hybrid systems. In *Automated Technology for Verification and Analysis*, volume 11138 of *Lecture Notes in Computer Science*, pages 422–440, 2018.

[15] C. E. Rasmussen and C. K. Williams. *Gaussian processes for machine learning*, volume 1. MIT press Cambridge, 2006.

[16] S. D. Stoller. Neural predictive monitoring. In *Runtime Verification: 19th International Conference, RV 2019, Porto, Portugal, October 8–11, 2019, Proceedings*, volume 11757, page 129. Springer Nature, 2019.

[17] V. Vovk, A. Gammerman, and G. Shafer. *Algorithmic learning in a random world*. Springer Science & Business Media, 2005.