Demo: Verisig – verifying safety properties of hybrid systems with neural network controllers

Radoslav Ivanov, James Weimer, Oleg Sokolsky, Insup Lee
University of Pennsylvania
Philadelphia, Pennsylvania
{rivanov,weimerj,sokolsky,lee}@seas.upenn.edu

ABSTRACT

This abstract presents Verisig, a scalable tool for verifying safety properties of closed-loop systems with neural network (NN) controllers. Verisig transforms the NN into an equivalent hybrid system and composes this hybrid system with the plant's. This abstract describes this transformation and outlines the tool's building blocks.

ACM Reference Format:
Radoslav Ivanov, James Weimer, Oleg Sokolsky, Insup Lee. 2019. Demo: Verisig – verifying safety properties of hybrid systems with neural network controllers. In *Design Automation for CPS and IoT (DESTION '19), April 15, 2019, Montreal, QC, Canada.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3313151.3313164

1 INTRODUCTION

Recent years have seen a rapid rise in the popularity of deep neural networks (DNNs), driven by DNNs' impressive performance in challenging learning tasks such image processing and language translation. In addition, DNNs are increasingly being used in safety-critical systems such as autonomous vehicles and air traffic collision avoidance systems. Thus, it is important to assure the safety of closed-loop systems with DNN components. However, it is challenging to develop analytic proofs about DNNs due to their complexity.

As a result, several formal verification approaches [3] have been developed in order to reason about the safety of DNNs. These techniques focus on verifying safety properties about the DNN's outputs given conditions on the inputs. Despite their impressive performance, however, these tools cannot be straightforwardly extended to reason about closed-loop systems with DNN controllers, except for special cases such as linear systems.

Different from existing approaches, Verisig [2] is developed to verify safety properties with respect to closed-loop systems. Verisig focuses on DNNs with sigmoid activation functions and exploits the fact that the sigmoid is the solution to a quadratic differential equation. This allows us to transform the DNN into an equivalent hybrid system, which is in turn composed with the plant's hybrid system. Given the resulting hybrid system, we can use existing hybrid system verification tools such as Flow* [1] in order to verify safety properties of the closed-loop system.

To state the problem more precisely, consider a closed-loop system where a plant P, described by a hybrid system H_P, is controlled by a DNN controller h. We assume h is a feedforward neural network with sigmoid/tanh activation functions. Given a property ξ on the initial states X_0 of P, the problem, expressed as property ϕ, is to verify a property ψ of the reachable states $x(t)$ of P:

$$\phi(X_0, x(t)) \equiv \xi(X_0) \Rightarrow \psi(x(t)),\ \forall t \geq 0. \tag{1}$$

We approach the problem by transforming the DNN into an equivalent hybrid system, H_h, and in turn composing H_h with H_P. The key insight used in this transformation is the fact that the sigmoid, $f(x) = 1/(1+e^{-x})$, can be expressed in terms of its derivative, i.e., $f'(x) = f(x)(1 - f(x))$. With this fact in mind, we introduce the proxy function $g(t,x) = f(tx)$, such that $g(1,x) = f(x)$ and $\dot{g}(t,x) = xg(t,x)(1 - g(t,x))$. Thus, each neuron in the DNN can be treated as a state in a dynamical system that evolves according to the "dynamics" of g such that at $t = 1$, g is equal to the sigmoid, i.e., by computing the reachable set for g at $t = 1$, one obtains the reachable set for the sigmoid. Thus, the DNN can be transformed into an equivalent hybrid system by mapping each neuron to a state and each layer to a mode; transitions between modes are triggered when $t = 1$. Finally, H_h is composed with H_P depending on how the controller is used; e.g., in a time-triggered system, there would be a transition from each mode in H_P to H_h with a guard which is enabled when the controller is sampled.

We use Flow* to verify property ϕ about the composed hybrid system $S = H_h \| H_P$. We chose Flow* to perform the hybrid system verification since it scales to systems with a few hundred states. Although the approximation error in Flow* might grow over time, this issue can be alleviated by splitting the initiial condition into subsets and verifying for each one, as described in Section 3.

2 DESIGN

As shown in Figure 1, Verisig is effectively a translation tool from a closed-loop system description to a Flow* model. Since the translation itself is not time-consuming, Verisig is written in Python, which also promotes code readability. Verisig takes as input four items: 1) the DNN architecture; 2) the plant's hybrid system description; 3) composition (glue) transitions between the DNN and the plant and vice versa; 4) the property ϕ. The tool constructs the closed-loop hybrid system and writes it to a model file that is sent as input to Flow*. We describe each of these aspects next.

2.1 DNN Input

Verisig currently accepts two types of DNN descriptions. The first is a Keras[1] model. Keras is an open-source toolbox for training DNNs and has gained popularity due to its extendability and ease of use. If a Keras model is not available, Verisig also accepts a Python dictionary where each layer maps to a two-dimensional array of neuron weights and a one-dimensional array of neuron offsets (each layer h_i is defined as $h_i(x) = a(W_i x + b_i)$, where W_i are the weights, b_i are the offsets, and a is the activation function).

[1]https://github.com/keras-team

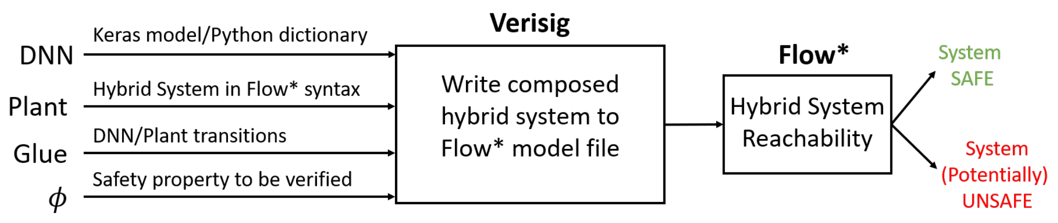
Verisig
DNN
Keras model/Python dictionary
Plant
Hybrid System in Flow* syntax
Glue
DNN/Plant transitions
ϕ
Safety property to be verified
Write composed hybrid system to Flow* model file
Flow*
Hybrid System Reachability
System SAFE
System (Potentially) UNSAFE

Figure 1: Design of Verisig.

2.2 Plant Hybrid System

The plant's hybrid system description is currently provided as a Python dictionary, where each mode index is mapped to a mode description. A mode description is also a dictionary, where the keys are the state dynamics, invariants and transitions for that mode. Since Verisig is strongly tied to Flow*, Verisig supports only Flow* syntax at the moment. We are developing a parser for a standard hybrid system description language such as SpaceEx XML.

2.3 Glue

To compose the plant hybrid system with the DNN model, we introduce two new transitions for each plant mode. The first one leads from the plant mode to the initial mode of the DNN model. The second transition leads from the final mode of the DNN model back to the plant mode. To capture the nature of these transitions (e.g., the DNN is a time- or event-triggered controller), the user needs to specify transition guards, which determine when the DNN is invoked, and resets that perform DNN-specific operations, e.g., normalization of DNN inputs. We are developing a simple language that would allow us to concisely specify this information and automatically generate glue transitions. Alternatively, the user can specify glue transitions explicitly using the Flow* syntax.

2.4 Safety property

The property ϕ is expressed in Flow* syntax. It specifies initial conditions for all plant states as well as unsafe modes and states.

2.5 Flow* model

A Flow* model consists of four parts: 1) mode descriptions; 2) initial conditions; 3) unsafe modes and states; 4) Flow* settings. Verisig writes the first three based on the provided DNN architecture and plant description (already in Flow* syntax). The Flow* settings include parameters such as precision, Taylor Model order, integration step size, etc. Choosing the right settings is essential for the tool to work well – Verisig uses default settings that work well in our case studies but these might need to change for different case studies.

3 CASE STUDY: MOUNTAIN CAR

We now illustrate how to use Verisig in a reinforcement learning case study, namely Mountain Car (MC). MC is a benchmark reinforcement learning problem, in which an underpowered car must drive up a steep hill by driving back to an opposite hill and gathering enough momentum so as to be able to climb the hill. The goal of reinforcement learning is to train a DNN to learn this control policy by applying different inputs and observing a reward after each step. Once a DNN is trained, we use Verisig to verify that the car will go up the hill starting from any initial condition.

In order to use Verisig in a case study, one needs to perform three tasks: 1) train a DNN; 2) encode the plant dynamics into Flow* syntax; 3) split the initial condition into small enough subsets such that each one can be verified by Flow* with small approximation error. We now explain the challenges involved in each step.

3.1 DNN Training

We use an Actor/Critic approach [4] to train a two-layer sigmoid-based DNN with 16 neurons per layer and a tanh-based output layer with one neuron. Note that we constrain the sigmoid weights to have norm less than 1; this constraint improves training and, more importantly, allows Verisig to keep the approximation error small when integrating the sigmoid dynamics. Since Flow* uses interval analysis to bound the error during each integration step, bigger weights result in more uncertainty and larger approximation error.

3.2 Encoding Plant Dynamics into Flow*

Flow* is very sensitive to the plant dynamics description. For example, the MC plant dynamics contain a cosine function in discrete time whereas Flow* only accepts cosine in continuous dynamics. One way around this issue is to dedicate a separate plant mode to compute the cosine in continuous dynamics. This trick comes at a cost since Flow* would need to integrate the cosine dynamics as well, thereby potentially introducing more approximation error.

3.3 Splitting the Initial Condition into Small Subsets

This is the most manually-intensive part of Verisig. Since it is difficult to estimate a priori what initial condition size would result in a small enough approximation error so as to verify the property ϕ, the user has to pick a size and try. Note that large initial sets may not only result in large approximation error but might also add unnecessary branches in the hybrid system (e.g., due to the car leaving the environment boundaries). At the same time, if the initial set is too small, the number of calls to Verisig might become prohibitively large. This challenge can be alleviated by parallelizing these calls once an initial set size is chosen.

REFERENCES

[1] X. Chen, E. Ábrahám, and S. Sankaranarayanan. 2013. Flow*: An analyzer for non-linear hybrid systems. In *International Conference on Computer Aided Verification*. Springer, 258–263.
[2] R. Ivanov, J. Weimer, R. Alur, G. J. Pappas, and I. Lee. 2019. Verisig: verifying safety properties of hybrid systems with neural network controllers. *Proceedings of the 22nd International Conference on Hybrid Systems: Computation and Control* (2019).
[3] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer. 2017. Reluplex: An efficient SMT solver for verifying deep neural networks. In *International Conference on Computer Aided Verification*. Springer, 97–117.
[4] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. 2015. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971* (2015).