Demo Abstract : Sherlock - A Tool For Verification Of Neural Network Feedback Systems

Souradeep Dutta, Xin Chen, Susmit Jha, Sriram Sankaranarayanan, Ashish Tiwari

ABSTRACT

We present an approach for the synthesis and verification of neural network controllers for closed loop dynamical systems, modelled as an ordinary differential equation. Feedforward neural networks are ubiquitous when it comes to approximating functions, especially in the machine learning literature. The proposed verification technique tries to construct an over-approximation of the system trajectories using a combination of tools, such as, Sherlock and Flow*. In addition to computing reach sets, we incorporate counter examples or bad traces into the synthesis phase of the controller as well. We go back and forth between verification and counter example generation until the system outputs a fully verified controller, or the training fails to terminate in a neural network which is compliant with the desired specifications. We demonstrate the effectiveness of our approach over a suite of benchmarks ranging from 2 to 17 variables.

CCS CONCEPTS

• **Computing methodologies** → **Neural networks**; • **Computer systems organization** → **Embedded and cyber-physical systems**; • **Mathematics of computing** → *Interval arithmetic*; *Differential equations*;

KEYWORDS

reachability analysis, polynomial regression , neural network, hybrid system, flowpipe construction

ACM Reference Format:
Souradeep Dutta, Xin Chen, Susmit Jha, Sriram Sankaranarayanan, Ashish Tiwari. 2019. Demo Abstract : Sherlock - A Tool For Verification Of Neural Network Feedback Systems . In *22nd ACM International Conference on Hybrid Systems: Computation and Control (HSCC '19), April 16–18, 2019, Montreal, QC, Canada.* ACM, New York, NY, USA, Article 4, 2 pages. https://doi.org/10.1145/3302504.3313351

1 INTRODUCTION

Neural networks have found their way into a large of gamut of applications, ranging from autonomous cars and unmanned aerial vehicles to medical applications. Typically, a neural network is used to perform tasks related to control, guidance, and classification. In most of these applications, the neural networks are synthesized or trained using methods like reinforcement learning, learning from demonstrations, or learning from a large precomputed table of control commands. Interestingly enough, most of these systems are not correct by construction. We aim to address this by developing verification techniques that can be potentially included in the design phase of the system. There has been an upsurge of recent interests in proving properties about neural networks. Some of the recent developments in this domain are, [2, 4, 5]. But, a lot of the recent literature has been around proving assertions about the neural networks in isolation, things like input-output bounds, and robustness of image classification networks. In this paper, we propose and demonstrate techniques which can verify properties of the dynamical system given by an ODE in a closed loop with a neural network. The approach is around building reach sets of the system using Flow* [1] for the ODE part, and Sherlock for the neural network part. Note that the simple amalgamation of the intervals computed by reachability tools for the ODE and neural network doesn't work for any of the benchmarks we present in this paper.

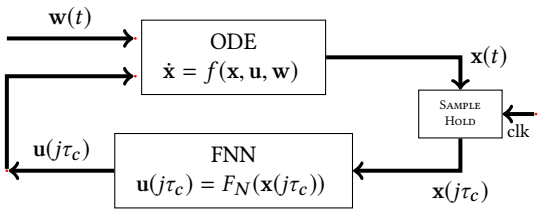

Figure 1: Block diagram of a neural feedback control system.

2 PROBLEM STATEMENT AND APPROACH

2.1 Problem Statement

Definition 2.1 (Neural Network). A k layer, n input, neural network with N neurons per hidden layer is described by matrices: $(W_0, \mathbf{b}_0), \ldots, (W_{k-1}, \mathbf{b}_{k-1}), (W_k, \mathbf{b}_k)$, wherein (a) $W_0, \mathbf{b}_0$ are $N \times n$ and $N \times 1$ matrices denoting the weights connecting the inputs to the first hidden layer, (b) $W_i, \mathbf{b}_i$ for $i \in [1, k-1]$ connect layer i to layer $i+1$ and (c) $W_k, \mathbf{b}_k$ connect the last layer k to the output.

At each neuron the output value gets computed from the input values using the *activation function* σ. In general, this function is usually some non-linear monotonic function, but here we focus on neural networks with "Relu" activation function $\sigma(z) : \max(z, 0)$. Nevertheless, most of the techniques that we present do not intrinsically depend on this restriction. Semantically, a neural network $\mathcal{N}$ with just a single output computes a continuous function $F_N : \mathbb{R}^n \to \mathbb{R}$, which is a composition given by $F_N := F_k \circ \cdots \circ F_0$, where $F_i(\mathbf{z})$ is the function computed by individual layers.

Definition 2.2. (Neural Feedback System) A Neural Feedback System $\mathcal{S}$ is tuple $\langle X, U, W, f(x, u, w), \mathcal{N}, \tau_c \rangle$, where $\dot{x} = f(x, u, w)$,

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
HSCC '19, April 16–18, 2019, Montreal, QC, Canada
© 2019 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-6282-5/19/04.
https://doi.org/10.1145/3302504.3313351

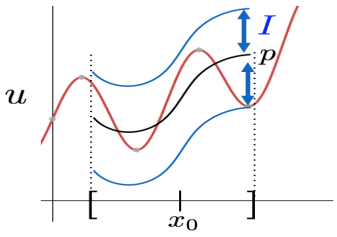

Figure 2: Polynomial Rule plus Interval: Red curve - neural network output around x_0. Black curve - polynomial obtained by regression. Blue curve - the upper and lower polynomial bounds after adding the interval error I.

defines the dynamical system, and, x, u, and w defines the state vector, control input, and disturbance respectively. We assume, that the state space is given by $X \in \mathfrak{R}^n$, the control input range is given by $U \in \mathfrak{R}^m$, and $W \in \mathfrak{R}^l$ is the set of disturbances. The controller is assumed to act with a time period of τ_c.

Note, the controller acts in discrete time, and consequently, the control signal is a piecewise constant signal. That is, u is constant for $t \in [q\tau_c, (q+1)\tau_c)$, for a positive integer q. The trajectory $x(t)$ of such a system, in a bounded time horizon $[0, T]$, with some initial state x_0, disturbance signal $w : [0, T] \rightarrow W$, and control signal $u : [0, T] \rightarrow U$, is such that for every time interval $t \in [q\tau_c, (q+1)\tau_c)$, $\frac{dx}{dt}(t) = f(x(t), u(t), w(t))$. Note that $u(t) = F_N(x(q\tau_c))$, where F_N is output of the neural network. Our goal is to compute reach sets $\mathcal{R}$ of a *Neural Feedback System*, into a time T into the future, starting from set of initial states $\mathcal{R}_\prime$.

2.2 Approach

The key step in our approach creates a sound abstraction of the behavior of the neural network function $F_N(x)$. Instead of treating the NFS as a hybrid automaton, where each "mode" is determined by a linear region of the neural network, we think of it as a continuous feedback system. The main reason is that the number of such modes can in principle be exponential in the number of neurons in the network. Instead, at each time step we abstract the feedback of the neural network by a polynomial of a given degree plus some error bound .

Let's assume that we are given an initial set X_0 and the aim is to compute the reach sets for N control steps into the future, which would translate to a time horizon of $T = N\tau_c$. We use Flow* to compute the reachable set at each time interval $[(j-1)\tau_c, j\tau_c]$. The system trajectories are over-approximated by some Taylor model flowpipes, and the continuous dynamics is updated according to the flowpipe in the last step.

For the j^{th} control step such that $j = 1, \ldots, N$, our algorithm performs the following steps.

(1) Compute the Taylor model overapproximation X_j for the reachable set at time $t = (j-1)\tau_c$.
(2) Next, given an X_j we compute an abstraction of the neural network function F_N, as a *polynomial rule* $q_j(\mathbf{x})$ plus an error interval J_j. This approximation is sound for any input $\mathbf{x} \in X_j$ of the neural network controller, i.e., $(\forall \mathbf{x} \in X_j).(F_N(\mathbf{x}) \in q_j(\mathbf{x}) + J_j)$. Details of this step will be presented in the main poster.
(3) Next, we compute the control input as $\mathbf{u}_j = q_j(X_j) + J_j$ for the current control step by Taylor model arithmetic with the order k.
(4) Finally, the continuous dynamics are updated to $\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}_j, \mathbf{w})$, and compute Taylor model flowpipes for the current control step. The new flowpipe computed is appended to the resulting list.

The main reason this approach is useful, for a large range of applications, is that the dependencies among the state variables of the system is preserved between the continuous and discrete components. This in turn reduces the overestimation in the reach set computation by a large margin.

2.3 Training the Neural Network

The neural networks were initially trained using the techniques presented in [3]. The approach involves a data driven MPC method to train neural network controllers, using simple back-propagation like techniques. A natural extension of using the reach sets computed is to tune the controller in order to make sure that there are no safety violations. Since, the method of over-approximation can lead to false alarms, we first confirm that the faulty behaviors correspond to concrete trajectories. This is accomplished via the use of gradient-descent like search techniques, and random restarts. Once we have the system traces which violate the specifications, we feed them back into the training samples, and retrain the controller, thus completing the design-verify-design loop.

3 RESULTS

We implemented the prototype tool for our neural rule generation inside Sherlock, and used it together with Flow* for computing the reach sets of the system. We used a set of 11 benchmarks of non-linear systems, with a neural network controller. The networks were trained using standard MPC control schemes. Our largest benchmark had upto 17 dimensions, which modelled a quadrotor. One of the neural networks involved had around 500 neurons, and some of the networks were up to 6 layers deep. We were able to successfully compute the reach sets for the benchmarks, and verify the behaviors for the corresponding initial states. Sherlock can be downloaded from *https://github.com/souradeep-111/sherlock*

REFERENCES

[1] X. Chen, E. Ábrahám, and S. Sankaranarayanan. 2013. Flow*: An Analyzer for Non-linear Hybrid Systems. In *Proc. of CAV'13 (LNCS)*, Vol. 8044. Springer, 258–263.
[2] Souradeep Dutta, Susmit Jha, Sriram Sankaranarayanan, and Ashish Tiwari. [n. d.]. Output Range Analysis for Deep Feedforward Neural Networks. NFM 2018 (to appear), Cf. https://arxiv.org/pdf/1709.09130.pdf.
[3] Souradeep Dutta, Susmit Jha, Sriram Sankaranarayanan, and Ashish Tiwari. 2018. Learning and Verification of Feedback Control Systems using Feedforward Neural Networks. *IFAC-PapersOnLine* 51, 16 (2018), 151 – 156. https://doi.org/10.1016/j.ifacol.2018.08.026 6th IFAC Conference on Analysis and Design of Hybrid Systems ADHS 2018.
[4] Rüdiger Ehlers. 2017. Formal Verification of Piece-Wise Linear Feed-Forward Neural Networks. In *ATVA (Lecture Notes in Computer Science)*, Vol. 10482. Springer, 269–286.
[5] Guy Katz, Clark Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. 2017. *Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks.* Springer International Publishing, Cham, 97–117. https://doi.org/10.1007/978-3-319-63387-9_5