# Learning and Verification of Feedback Control Systems using Feedforward Neural Networks.

**Souradeep Dutta** [*] **Susmit Jha** [**] **Sriram Sankaranarayanan** [*]
**Ashish Tiwari** [**]

[*] *University of Colorado, Boulder, (first.lastname@colorado.edu)*
[**] *SRI International, ({susmit.jha,tiwari}@sri.com)*

Abstract We present an approach to learn and formally verify feedback laws for data-driven models of neural networks. Neural networks are emerging as powerful and general data-driven representations for functions. This has led to their increased use in data-driven plant models and the representation of feedback laws in control systems. However, it is hard to formally verify properties of such feedback control systems. The proposed learning approach uses a receding horizon formulation that samples from the initial states and disturbances to enforce properties such as reachability, safety and stability. Next, our verification approach uses an over-approximate reachability analysis over the system, supported by range analysis for feedforward neural networks. We report promising results obtained by applying our techniques on several challenging nonlinear dynamical systems.

*Keywords:* Reachability and safety analysis, verification, formal synthesis, neural networks

## 1. INTRODUCTION

In this paper, we provide an approach to synthesize verified controllers for data-driven models of nonlinear systems towards objectives such as reachability and region stability. Our approach works for plant models defined using a feed forward neural network that has been inferred through regression on actual measurement data. The goal is to synthesize feedback laws that are also described by feedforward neural networks. Such data-driven models are now increasingly common for many types of systems (Hou et al. [2017b]). However, rigorous approaches to synthesis and verification are often needed to synthesize controllers.

In this work, we consider the problem of synthesizing neural networks that can serve as state and output feedback laws to achieve control objectives that can be specified as reachability of a target set, region stability of a target set around an equilibrium point, or in principle, logical formulas written in a temporal logic such as signal temporal logic(STL) (Donzé and Maler [2010]). Our approach incorporates two phases: **(a)** Data-Driven model predictive control (MPC) for training the weights of the feedback network from samples; and **(b)** Formal reachability analysis using range propagation as a primitive operation that computes reachable states enabling an over-approximate symbolic model checking of the resulting closed loop. The reachability analysis builds a post condition operator over template polyhedra with fixed LHS expressions, but variable constants on the RHS of inequalities. The key primitive for constructing template polyhedra uses recently available approaches such as ReluPlex that can check assertions involving neural networks, as well as our recent approach called SHERLOCK that computes intervals over output values of the network given a predicate constraining the inputs (Dutta et al. [2017], Katz et al. [2017]). We use SHERLOCK as a primitive to perform reachability analysis, and thus adapt standard approaches to build a symbolic model checker that can verify the resulting feedback controller (Frehse et al. [2011]).

We demonstrate our approach successfully over a series of challenging nonlinear control examples for underactuated systems with $2-4$ state variables showing how our approach can synthesize region stabilizing controllers and at the same time formally verify that these controllers satisfy the region stability property . The controllers synthesized involve 100s of neurons with intermediate verification problems that handle deep networks with as many as $\sim 100s$ of layers

Overall, the principal contribution of our approach is to establish evidence that a formal synthesis framework can combine classic backpropagation-based training from samples and verification using reachability analysis tools. In particular, the contributions include: (a) combination of backpropagation-based training techniques with classic receding horizon MPC; (b) use of range analysis as a primitive for symbolic model checking neural network based closed loops and (c) empirical evidence of feasibility of our approach especially over larger networks than previously possible. At the same time, the approach is limited to using template polyhedra with linear templates, has only been developed for reachability and region stability properties (though we make the case that other properties are possible) and furthermore, lacks a refinement loop that can translate failure of verification to test cases for the training. We continue to investigate approaches for these limitations.

## 2. RELATED WORK

Data driven approaches for verification has received some attention in literature. (Haesaert et al. [2017]), present a measurement driven approach which relies on Bayesian inference rules and reachability analysis to verify LTL properties. Another recent approach in using data driven techniques is the DRY-VR framework proposed in (Fan et al. [2017]), which uses simulation techniques combined with probabilistic verification

algorithms to reason about the system. Some verification tools which rely on over-approximate reachable set computation are, Flow*, (Chen et al. [2013]), CORA (Althoff [2015]) for non-linear systems and SpaceEx (Frehse et al. [2011]) for linear systems. Another class of approaches exists which uses simulation based techniques to prove safety properties, are presented in (Kanade et al. [2009]), (Deng et al. [2013]), Breach (Donzé [2010]), and S-Taliro (Annpureddy et al. [2011]).

But, when it comes to the problem of designing correct-by-synthesis controllers for data driven models, there is not much existing work. One of the very few approaches in data-driven control synthesis is Model Free Adaptive Control (MFAC) technique, discussed in (Hou et al. [2017a]). Dynamic linearization serves as a fundamental method in the MFAC scheme. The work in (Spall and Cristion [1998]) uses discrete time measurements of the system to compute the control actions. Our approach is different, since we do not use any complex online scheme to estimate system behavior and compute control actions. Another approach to assess stability based on the input-output system data was presented in (Wang and Liu [2013]). A survey of data-driven control techniques was presented in (Hou and Wang [2013]). A common problem with many of these approaches is that they are seldom scalable to higher dimensional systems, and can end up being computationally expensive for even smaller ones.

A data driven approach to synthesize a high level demand-response strategy for cyber-physical energy systems was presented in (Behl et al. [2016]), (Jain et al. [2016]). Our approach is different from the above data driven approach, since here we are trying to solve a more low-level control law computation problem for data-driven system models.

The closest technique that tries to accomplish control law computation by merely keeping track of input-output relations is that of Model Based Reinforcement Learning (Williams et al. [2017]). Though reinforcement learning techniques have proved to be quite successful recently, there isn't much work when it comes to *proving* properties on these closed loop systems to make them more reliable. This is mainly due to the difficulty imposed by the use of neural networks as function approximators. Our verification approach proposed here can be complementary to such techniques, since the reasoning methods proposed here can be easily used to prove stability and safety properties of closed loop systems learnt using other techniques.

## 3. PRELIMINARIES

We first define the neural network models used, and the process of training these models. Next, we discuss a recent approach for finding the output range of a network given assertions over its input. We refer the reader to a standard textbook for details on neural networks Goodfellow et al. [2016].

A feedforward neural network $N$ is a directed acyclic graph whose nodes may represent "hidden neurons", inputs or outputs. Each neuron implements a nonlinear function $y = \sigma(x)$, wherein the function $\sigma$ for a "ReLU" (recurrent linear unit) considered in this paper is $\sigma(x) : \max(x, 0)$. The connections between neurons have a weight $w_{ij}$ an the set of weights are collectively written as $\mathscr{W}$. Each neuron's input is defined as the weighted sum of the outputs from its incoming edges and the output s obtained using the function $\sigma$. It is well known that neural networks compute a function $F_W(\mathbf{x})$ over the inputs $\mathbf{x}$ and weights given by $W$. This function is continuous and differentiable almost everywhere over $\mathbf{x} \in \mathbb{R}^n$.

*Training a Neural Network:* Training a network starts from given input-output pairs $(\mathbf{x}_i, \mathbf{y}_i)_{i=1}^N$ and a network with a known graph structure but with unknown weights $\mathscr{W}$. The goal of training is to find weights $\mathscr{W}$ that minimize a predefined loss function $\mathscr{L}(\mathbf{y}', \mathbf{y})$ that measures the deviation from two sets of outputs $\mathbf{y}', \mathbf{y}$. The overall optimization problem is written as that of minimizing the overall loss over the given training data.

$$\mathscr{W}^* : \operatorname*{argmin}_{\mathscr{W}} \sum_{j=1}^{N} \mathscr{L}(F(\mathbf{x}_j; \mathscr{W}), \mathbf{y}_j)\,.$$

Finding $\mathscr{W}^*$ is a hard problem in practice since it involves a large nonconvex optimization problem. In practice, approaches such as stochastic gradient descent have been employed to find $\mathscr{W}^*$ that is a local minimum of the overall optimization problem. This process does not have any guarantees but is known to perform quite well for many types of modeling problems. Furthermore, there are many heuristics ranging from randomly choosing starting points, fixing large number of weights in $\mathscr{W}$ to 0 (thus insisting on a sparse model) and selecting random subsets of data to calculate the gradient (Bottou [2010]). This addresses the problem of overfitting.

*Range Analysis for Neural Networks:* The problem of range analysis for a neural network starts from a network $\mathscr{N}$ and an assertion $\varphi[\mathbf{x}]$ over the inputs to the network. The goal is to find an interval $[\ell, u]$ such that

$$\varphi[\mathbf{x}] \;\models\; F_W(\mathbf{x}) \in [\ell, u]\,.$$

Often, we are interested in ensuring that the interval is tight. Finding such an interval over the outputs is performed by solving a series of $2m$ optimization problems, i.e., two problems for each output variable $y_j$:

$$\max(\min)\; y_j \;\mathsf{s.t}\; \varphi[\mathbf{x}] \;\wedge\; \mathbf{y} = F_W(\mathbf{x})\,,$$

However, the problem of solving optimization problems with neural network constraints is highly nonlinear. Using the properties of ReLU function, it can be encoded as a large mixed integer linear program (MILP). However, this approach seems to be quite expensive in practice. Our recent work instead uses a combination of local and "global" search steps alternating between a simple local iterative step that uses easy to compute gradient information to improve the current solution. While we do use an MILP solver to perform global search, it is only asked to provide a small $\varepsilon$ improvement to an existing local solution. We find that the combined approach is faster and more effective for many of the networks tested. This approach has been implemented inside the tool SHERLOCK (Dutta et al. [2017]). Throughout this paper, we will use SHERLOCK as a black-box solver with the following specification:

**INPUT:** A neural network $\mathscr{N}$, linear assertions over the input variables $\mathbf{x}$ and a tolerance parameter $\varepsilon$.

**OUTPUT:** Interval $[\ell, u]$ over the outputs $\mathbf{y}$ such that (a) $\varphi[\mathbf{x}] \models F_W(\mathbf{x}) \in [\ell, u]$; and (b) for each variable $y_j$, we have $\ell_j \geq \ell_j^* - \varepsilon$ and $u_j \leq u_j^* + \varepsilon$. The first condition says that the range $[\ell, u]$ is a valid overapproximation. The second asserts that $[\ell, u]$ is not more than $\varepsilon$ away from the tightest range possible.

Furthermore, corresponding to the bounds $\ell_j, u_j$, the approach provides witnesses $\mathbf{x}_{\ell,j}, \mathbf{x}_{u,j}$ such that $F(\mathbf{x}_{\ell,j}) \leq \ell_j + \varepsilon$ and $F(\mathbf{x}_{u,j}) \geq u_j - \varepsilon$.

By adjusting $\varepsilon$, it is possible to trade off the precision vs. running time of our approach.

## 4. PROBLEM STATEMENT

In this section, we will formally describe the overall problem statement for our approach. The problem consists of a given discrete-time plant model $\mathscr{P}$ over state variables $\mathbf{x} \in \mathbb{R}^n$ and control inputs $\mathbf{u} \in \mathbb{R}^l$, and a plant function $f_p : \mathbb{R}^n \times \mathbb{R}^l \to \mathbb{R}^n$. Further, we define an output function $g_p : \mathbb{R}^n \to \mathbb{R}^m$ that maps state variables $\mathbf{x} \in \mathbb{R}^n$ to outputs $\mathbf{y} : g_p(\mathbf{x})$.

*Definition 4.1.* (Plant Model). Given a sequence of control inputs $\mathbf{u}(0), \ldots, \mathbf{u}(n)$ and initial plant state $\mathbf{x}(0)$, the resulting sequence of plant states $\mathbf{x}(0), \ldots, \mathbf{x}(n)$ and outputs $\mathbf{y}(0), \ldots, \mathbf{y}(n)$ are defined recursively as

$$\mathbf{x}(t+1) = f_p(\mathbf{x}(t), \mathbf{u}(t)), \;\; \mathbf{y}(t+1) = g_p(\mathbf{x}(t+1)) \text{ for } t \geq 0.$$

Note that, $\mathbf{y}(0) : \; g_p(\mathbf{x}(0))$.

The overall goal is to synthesize a control feedback function $h : \mathbb{R}^m \to \mathbb{R}^l$ that maps output values to control inputs such that the closed loop defined by $f_p$ and $h$ satisfies some desired properties. The closed loop function is defined by

$$\mathscr{C}_{f_p, g_p, h}(\mathbf{x}) : \; f_p(\mathbf{x}, h(g_p(\mathbf{x}))) \, .$$

We will drop the subscript whenever $f_p, g_p$ and $h$ are known from the context.

We would like to design $h$ so that the resulting closed loop $\mathscr{C}$ satisfies some desirable properties such as safety, reachability of a given target, region stability (Podelski and Wagner [2007]), or a general property defined in a suitable logic such as *Signal Temporal Logic* (STL) (Donzé and Maler [2010]). In this paper, we restrict our attention to basic properties of reachability and region stability.

*Definition 4.2.* (Reachability and Region Stability). Let $T \subseteq \mathbb{R}^n$ be a given *target* set of states and $X_0 \subseteq \mathbb{R}^n$ be the initial set of states. We say that the closed loop $\mathscr{C}$ satisfies the reachability property for $T$ starting from $X_0$ iff for all $\mathbf{x}_0 \in X_0$, the resulting execution trace $\sigma : \; \mathbf{x}(0), \mathbf{x}(1), \cdots, \mathbf{x}(t), \cdots$ has a state $\mathbf{x}(t) \in T$ for some $t \geq 0$. In temporal logic, we write this property as $\mathscr{C} \models X_0 \Rightarrow \Diamond(T)$.

Region stability extends the notion of reachability by also requiring that the system reach the target set $T$ and remain in $T$ forever. In temporal logic, we express this property as $\mathscr{C} \models X_0 \Rightarrow \Diamond \Box(T)$.

We consider the following problem statement:

**Inputs:** Neural networks $\mathscr{F}_p$ and $\mathscr{G}_p$ representing the functions $f_p$ and $g_p$, respectively; initial condition $X_0$ and target set $T$.

**Output:** A feedback function $h$ denoted by a neural network $\mathscr{H}$ such that the resulting closed loop satisfies the reachability property $X_0 \Rightarrow \Diamond(T)$ (or region stability $X_0 \Rightarrow \Diamond \Box(T)$).

The subsequent sections show how to (a) define the neural network using a data-driven MPC scheme and (b) verify that the networks satisfy reachability and region stability properties using the range propagation algorithm as a primitive.

## 5. DATA DRIVEN MODEL-PREDICTIVE CONTROL

In this section, we introduce data-driven MPC approach that learns a NN model for the controller using the idea of finite-horizon lookahead from model-predictive control. For simplicity, we explain our ideas using state-feedback controllers, i.e, the function $g_p$ is the identity function. However, our ideas can extend, in principle, to output feedback, as well.

### 5.1 Receding Horizon MPC

Receding horizon MPC uses a given depth $k > 0$ and considers the next $k$ steps of the system execution starting from the current state $\mathbf{x}$. Further, we will have a (positive definite) cost function $c(\mathbf{x}_0, \ldots, \mathbf{x}_k, \mathbf{u}_0, \cdots, \mathbf{u}_{k-1})$, wherein $\mathbf{x}_0, \ldots, \mathbf{x}_k$ are the states and $\mathbf{u}_0, \ldots, \mathbf{u}_{k-1}$ are the control inputs applied over the next $k$ steps.

Model-predictive control (MPC) uses a $k$-step horizon lookahead to determine the best control actions at each step. Specifically, at any given state $\mathbf{x}$, the control action $\mathbf{u}$ is computed by solving the following optimization problem:

$$\begin{array}{l} \min \; c(\mathbf{x}, \mathbf{x}_1, \ldots, \mathbf{x}_k, \mathbf{u}_0, \ldots, \mathbf{u}_{k-1}) \\ \mathsf{s.t.} \; \mathbf{x}_1 = f_p(\mathbf{x}, \mathbf{u}_0) \\ \qquad \vdots \\ \qquad \mathbf{x}_k = f_p(\mathbf{x}_{k-1}, \mathbf{u}_{k-1}) \end{array} \tag{1}$$

The fixed data is the current state $\mathbf{x}$. The remaining variables $\mathbf{x}_1, \ldots, \mathbf{x}_k$ and $\mathbf{u}_0, \ldots, \mathbf{u}_{k-1}$ are the unknown decision variables. The result is a sequence of optimal $k-$step control inputs $\mathbf{u}_0, \ldots, \mathbf{u}_{k-1}$. The feedback law simply selects the first control point to define the feedback law $h(\mathbf{x}) : \; \mathbf{u}_0$, discarding the remaining control inputs.

Typically, MPC is used in an online receding horizon fashion (so-called implicit MPC). The optimization problem is solved at each time step to compute $h(\mathbf{x})$. The control is applied and in the next step, a fresh state measurement $\mathbf{x}(t+1)$ is obtained for which the process is repeated again in the next time step. In this paper, however, we will learn a neural network model for the complex feedback function $h$ in a data-driven fashion.

### 5.2 Data-Driven MPC

Our goal is to compute the controller explicitly as a function from the state $\mathbf{x}$ of the system to the control inputs $\mathbf{u}$. Our approach is based on learning the controller using reinforcement learning where the reward is given by the cost function $c$.

The data-driven approach is initialized with three inputs:

**Neural Network for Plant:** We are provided a neural network $\mathscr{N}_f$ for the plant model $f_p$ (and in the general output-feedback scheme, a network $\mathscr{N}_g$ for the output).

**Generate Samples:** We sample initial states $\mathbf{x}_0^{(1)}, \ldots, \mathbf{x}_0^{(N)}$ from the set $X_0$ of possible initial states.

**Neural Network Architecture:** We assume a given neural network structure $\mathscr{N}_h$ for the desired feedback law $h$ with unknown weights collectively written as $\mathscr{W}_h$.

The overall goal of this approach is to learn the unknown $\mathscr{W}_h$ using the back-propagation algorithm with the MPC cost $c(\mathbf{x}_0, \ldots, \mathbf{x}_k, \mathbf{u}_0, \ldots, \mathbf{u}_{k-1})$ as the loss function. Figure 1 shows the setup of the overall training MPC loop. The approach composes the neural networks $\mathscr{N}_h$ for the controller and the network $\mathscr{N}_f$ for the plant, depicting an unwinding of $k$ steps.

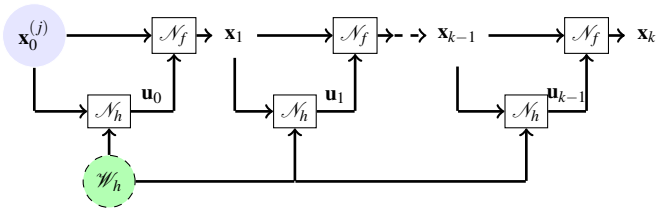


Figure 1. Unwinding of the closed loop model used to train the unknown weights $\mathscr{W}_h$ of the feedback neural network. Note that $\mathbf{x}_0^{(j)}$ is a fixed sample, and $\mathbf{u}_0,\ldots,\mathbf{u}_{k-1},\mathbf{x}_1,\ldots,\mathbf{x}_k$ are outputs that feed into the cost function evaluation. Finally, $\mathscr{W}_h$ are the unknown weights of the network $\mathscr{N}_h$.

Let $S$ be the finite set of $N$ state samples. For each sample $\mathbf{x}_0 \in S$, and given current weights $\mathscr{W}_h$ for the feedback network, the sample loss is written as

$$\hat{C}(\mathbf{x}_0,\mathscr{W}_h):\ c(\mathbf{x}_0,\mathbf{x}_1,\ldots,\mathbf{x}_k,\mathbf{u}_0,\ldots,\mathbf{u}_{k-1}),$$

wherein $\mathbf{x}_1,\ldots,\mathbf{x}_k,\mathbf{u}_0,\ldots,\mathbf{u}_{k-1}$ are computed from $\mathbf{x}_0,\mathscr{W}_h$, as specified in Figure 1. The overall loss function for the network training is represented as

$$\mathscr{L}_S(\mathscr{W}_h):\ \sum_{\mathbf{x}_0^{(j)}\in S}\hat{C}(\mathbf{x}_0,\mathscr{W}_h).$$

### *5.3 Backpropagation-based Training*

Given the setup involving a loss function $\mathscr{L}(\mathscr{W}_h)$ with sample set $S$, the core idea is to train the network weights $\mathscr{W}_h$ using backpropagation. This process requires the repeated calculation of a gradient $\nabla_{\mathscr{W}_h}\mathscr{L}_S$. This is performed systematically by composing gradients for elementary operations. In fact, frameworks such as `TensorFlow` to automate this computation using automatic differentiation techniques, ensuring that the computation is performed accurately (Abadi, M. et al. [2016]). We will therefore, skip the details of the gradient calculations, noting that it is easily automated.

Once the gradient calculation is automated, the approach uses stochastic (batched) gradient descent wherein a batch $S_i$ with $|S_i| \simeq 50$ samples are drawn on the fly from the initial set and a single setp of gradient is perfomed by computing $\nabla_{\mathscr{W}_h}\mathscr{L}_{S_i}$. This is once again automated in the `TensorFlow` framework. We also note that the gradient calculation and training can be performed quite efficiently using GPUs.

### *5.4 Defining a Loss Function*

Finally, we consider the process of systematically defining a loss function from the property at hand, focussing mostly on reachability and stability properties. In general, the loss function for a given STL formula can be written using a robustness metric (Donzé and Maler [2010], Fainekos and Pappas [2009]). However, computing derivatives of this metric with respect to trajectory perturbations can lead to technical problems (Abbas and Fainekos [2011]). For one, robustness is defined using a combination of max/min operators on top of basic primitives that include point to set distances. As such, the function is continuous under certain mild assumptions, but not differentiable everywhere. One approach involves redefining this under softmax/softmin operators to define a smooth and differentiable robustness metric

In what follows, we will describe simple cost functions for reachability and region stability properties.

*Reachability Property:* Let us consider the reachability property for a given target set $T$. We will assume for simplicity that $T$ is described as non-empty set $T:\{\mathbf{x}\mid t(\mathbf{x})\le t_0\}$ wherein $t$ is a smooth (convex) function and $t_0$ is a level. The robustness for a trace with states $\mathbf{x}_0,\ldots,\mathbf{x}_k$ is defined as (Cf. Donzé and Maler [2010], Fainekos and Pappas [2009])

$$\rho_{\diamond T}(\mathbf{x}_0,\ldots,\mathbf{x}_k):\ \min_{j=0}^{k}(t(\mathbf{x}_j)-t_0).$$

A negative value of the robustness function indicates that the property is true whereas a positive value indicates that it is false.

Let $\mathbf{x}_0,\ldots,\mathbf{x}_k$ represent a trajectory. We wish to write down a cost function $\mathsf{reachCost}_T(\mathbf{x}_0,\ldots,\mathbf{x}_k)$ that approximates the robustness function:

$$\mathsf{reachCost}_T(\mathbf{x}_0,\ldots,\mathbf{x}_k):\ -\mathsf{softMax}\begin{pmatrix}t_0-t(\mathbf{x}_0)\\ \vdots\\ t_0-t(\mathbf{x}_k)\end{pmatrix}.$$

The softmax of a vector $\mathbf{z}\in\mathbb{R}^n$ is defined as

$$\mathsf{softMax}_\alpha(\mathbf{z}):\ \frac{\sum_{i=1}^n \mathbf{z}_i\exp(\alpha\mathbf{z}_i)}{\sum_{i=1}^n\exp(\alpha\mathbf{z}_i)}.$$

as $\alpha\to\infty$, we have that $\mathsf{softMax}\to\max$ uniformly. Note that softMax is a smooth function that is also concave (thus its negation is convex).

*Region Stability:* Let us consider region stability with respect to the set $T$ defined by predicate $t(\mathbf{x})\le t_0$. There are two ways of defining robustness. The first approach simply uses the robustness definition for the $\diamond\Box(T)$ property by using a min over max formulation. A simpler cost function is to simply use a quadratic cost function:

$$\mathsf{stabilityCost}_T(\mathbf{x}_0,\ldots,\mathbf{x}_k):\ \sum_{i=0}^{k}\mathsf{softMax}_\alpha(t(\mathbf{x}_i)-t_0,0)^2.$$

Such a function encourages the value of $t(\mathbf{x}_i)$ to be as smaller than $t_0$, "as soon as possible" in the trace.

Another important step is to combine the cost function over states with a corresponding cost that encourages the optimizer to consider small values of the control input.

Therefore, the overall cost for reachability will be given by

$$c(\mathbf{x}_0,\ldots,\mathbf{x}_k,\mathbf{u}_0,\ldots,\mathbf{u}_{k-1}):\ \sum_{i=0}^{k-1}||\mathbf{u}_i||_2^2+\mathsf{reachCost}(\mathbf{x}_0,\ldots,\mathbf{x}_k).$$

We augment the stability cost similarly.

## 6. REACHABILITY ANALYSIS

Given a closed loop $\mathscr{C}$ described by neural networks $\mathscr{N}_f$ for the plant model function $f_p$ and $\mathscr{N}_h$ for the feedback law $h$, and initial state $X_0$ (represented as a polyhedron over the state space), we wish to compute symbolic representations for sets $X_1,X_2,\ldots,X_K$ wherein $X_i$ represents the reachable states of the closed loop system given by the composition of the plant and the feedback law in $i$ steps. Here $K$ is some fixed time horizon. We will use reachability analyis as a primitive for checking reachability, invariance and stability properties.

### 6.1 Post-Condition Operator

First, we compute an over-approximation of the post operator:

$$\mathsf{post}(X; f_p, h):\ \{\mathbf{x} \in \mathbb{R}^n \mid (\exists \mathbf{x}_0 \in X)\ \mathbf{x} = f_p(\mathbf{x}_0, h(\mathbf{x}_0))\}.$$

In general, given a polyhedral set $X$ and neural networks $\mathscr{N}_p$ and $\mathscr{N}_h$ for the functions $f_p, h$, respectively, the precise post condition is a union of polyhedron, that can be exponential in the total number of neurons in the two given networks. This is prohibitively expensive to compute precisely. Therefore, we settle for a single polyhedron $P(X)$ that approximates the post condition. To do so, we employ template polyhedra:

*Definition 6.1.* (Template Polyhedra). A template $T$ is a set of expressions $T:\ \{\mathbf{e}_1, \ldots, \mathbf{e}_r\}$ wherein each $\mathbf{e}_i$ is an linear expression of the form $\mathbf{c}_i^t\mathbf{x}$ over the state variables. A template polyhedron $P$ over a template $T$ is of the form: $\bigwedge_{j=1}^{r} \ell_j \leq \mathbf{e}_j \leq u_j$, for bounds $\ell_j, u_j$ over each template expression $\mathbf{e}_j$.

We will fix a template $T$ and represent reachable sets by template polyhedra over these templates. The post condition operation is therefore replaced by a template-based post-condition operator $\mathsf{post}_T(X; f_p, h)$ that yields bounds $\ell_j, u_j$ for each $\mathbf{e}_j \in T$ by solving the following optimization problem:

$$\ell_j(u_j):\ \min(\max)\ \mathbf{e}_j[\mathbf{x}]\ \text{s.t.}\ \mathbf{x}_0 \in X_0,\ \mathbf{u} = h(\mathbf{x}_0),\ \mathbf{x} = f_p(\mathbf{x}_0, \mathbf{u}).$$

Note however, that this optimization involves functions $h$ and $f_p$ defined by neural networks. However, the combination of local search and MILP encoding used in our tool SHERLOCK can be modified almost trivially to solve this optimization problem. Furthermore, the guarantees used in SHERLOCK extend. Thus, we guarantee that the reported result is no more than $\varepsilon$ away from the true value, for the given tolerance parameter $\varepsilon$.

### 6.2 Reachable Set Computation and Acceleration

We have described how to compute a single step post-condition using a range analysis tool SHERLOCK as a primitive. The next step is to perform a $k$ step reachability analysis that repeatedly uses the $\mathsf{post}_T$ operator to compute $X_{i+1}:\ \mathsf{post}_T(X_i; f_p, h)$. Here, each $X_i$ for $i > 0$ is a template polyhedron given by the template $T$ and bounds $\ell_j^{(i)}, u_j^{(i)}$ for each expression $\mathbf{e}_j \in T$. Because SHERLOCK guarantees the $\varepsilon$ tolerance factor in $T$, let us denote the exact reachable sets as $R_{i+1}:\ \mathsf{post}(R_i; f_p, h)$, with $R_0:\ X_0$. Let $\ell_j^{(*,i)}$ (and $u_j^{(*,i)}$) denote the optimal value

$$\min(\max)\ \mathbf{e}_j[\mathbf{x}]\ \text{s.t.}\ \mathbf{x} \in R_i.$$

*Lemma 6.1.* The reachable bounds for expression $\mathbf{e}_j$ at the $i^{th}$ step $(\ell_j^{(i)}, u_j^{(i)})$ differ from the best possible bounds by at most $i\varepsilon$: $\ell_j^{(i)} \geq \ell_j^{(*,i)} - i\varepsilon, u_j^{(i)} \leq u_j^{(*,i)} + i\varepsilon$.

In other words, as the size of the template $|T| \to \infty$ (assuming that the directions are sampled uniformly from the set of unit vectors), and furthermore, $\varepsilon \to 0$, our approach calculates the precise convex hull of the reachable set in the limit.

Just as SHERLOCK can be used to compute a single step reachability relation, we can accelerate this computation by using the tool for a $k$ step reachablity $\mathsf{post}_T^{(k)}(X; f_p, h)$ with the tolerance factor $\varepsilon$. To achieve this, we calculate the bounds $\ell_j^{(k)}, u_j^{(k)}$ as

$$\ell_j^{(k)}(u_j^{(k)}):\ \begin{bmatrix} \min(\max)\ \mathbf{e}_j[\mathbf{x}_k] \\ \text{s.t.}\ \mathbf{x}_0 \in X, \\ \mathbf{x}_1 = f(\mathbf{x}_0, h(\mathbf{x}_0)), \\ \ldots,\ \mathbf{x}_k = f(\mathbf{x}_{k-1}, h(\mathbf{x}_{k-1})) \end{bmatrix}.$$

Table 1. Details of the experiments. **Legend:** *Acc*: Acceleration Factor, $Acc_T$: Acceleration Factor for the Target Set, $Reach_T$ : Reach Set Time , $Inv_T$ : Time taken to prove stability of the target set . All the tests were run on a Linux server running Ubuntu 17.04 with 24 cores, and 64GB RAM

| ID | NN Layer Sizes | *Acc* | $Reach_T$ | $Acc_T$ | $Inv_T$ |
|---|---|---|---|---|---|
| 1 | 2,52,3,4,3,4,3,200,2 | 1 | 7.53s | 2 | 2.8s |
| 2 | 2,102,52,3,4,3,4,3,250,2 | 2 | 2m25s | 2 | 1m3s |
| 3 | 3,103,53,4,5,4,5,4,600,3 | 2 | 2m33s | 5 | 3m10s |
| 4 | 3,103,53,4,5,4,5,4,300,3 | 1 | 48s | 3 | 17.89s |
| 5 | 3,103,4,5,4,5,4,300,3 | 5 | 63m6.4s | 16 | 111m45s |
| 6 | 3,303,203,4,252,3 | 2 | 16m25s | 4 | 9m19s |
| 7 | 4,104,5,6,5,6,5,600,4 | 3 | 19m42s | 8 | 22m1s |

Algorithm 1 shows the approach to check reachability of a target set $T$ by computing sets $X_i$ reached after $i \geq 0$ applications of the post operator. Likewise, to check for $\Diamond\Box T$, we use $(k,l)$ induction through repeated post condition computation as stated in Algorithm 2. Here the values of $k, l$ are incremented starting from $k = l = 1$.

**Algorithm 1** Check Reachability

```
1: procedure CHECKREACH
2:     X̂_0 ← X_0 \ T, i = 0
3: loop i:
4:     X_{i+1} ← post_T(X̂_i; f_p, h)
5:     X̂_{i+1} ← X_{i+1} \ T
6:     if X̂_i = ∅ then return Proved
7:     if i = N then return NotProved
```

**Algorithm 2** Check Stability Using k,l induction

```
1: procedure CHECKSTABLE
2:     X̂_0 ← X_0 \ T, i = 0
3: loop k,l:
4:     X_k ← post^(k)(X_0; f_p, h)
5:     X_{k+l}: post^(l)(X_k; f_p, h)
6:     if X_k ⊆ T and X_{k+l} ⊆ X_l then return ◇□(T)
```

## 7. EXPERIMENTAL RESULTS

We used the control synthesis technique explained above to train controller for a data-driven neural network model for various benchmark systems taken from the literature. A detailed descriptions of these benchmarks is available in our extended version. We used standard ODE simulation techniques to generate the data from the differential equations, and then use the data generated to learn a neural network model.

Our tool computes the reach-set overapproximations for various initial conditions, and prove its convergence to a target set around the origin. Table 1 reports on the sizes of the composed neural networks and the time taken to run for each of the benchmarks chosen. Plots of the reachable sets are shown in Fig 2 for a few of the benchmarks.

## 8. CONCLUSION

To conclude, we have presented a synthesis approach that can train the weights of a neural network feedback law using a receding horizon formulation. Furthermore, we show how properties such as region stability can be established for the resulting

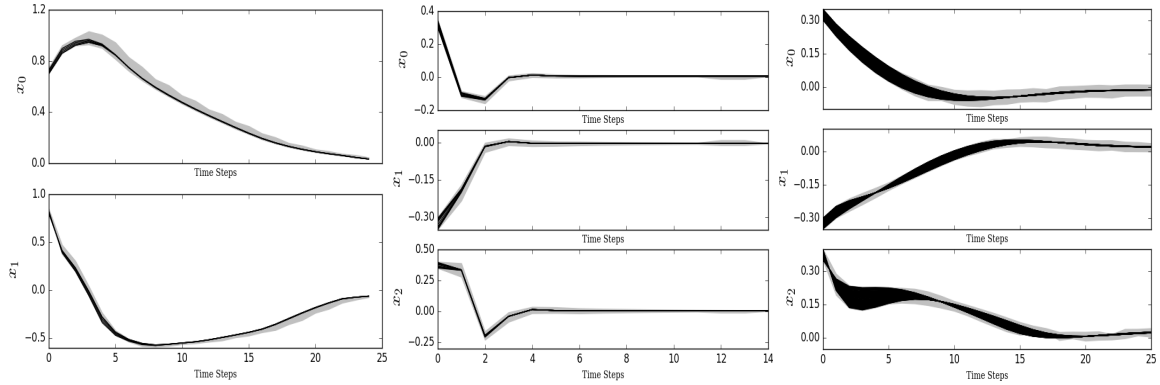


Figure 2. Plots for Example 2, 4, and 5 (left to right and, top to bottom). The limits of the reach sets computed along individual dimensions have been shown. The reach sets were overlaid with 100 real trajectories with randomly chosen initial points. The computation of reach sets was terminated once it was inside the target set $T$

controller. Future work will extend our work to more complex specifications such as guaranteed trajectory tracking, and integrate our work with neural network synthesis approaches such as deep reinforcement learning.

**Acknowledgments:** . This work was funded in part by the US National Science Foundation (NSF) under award numbers CNS-1646556, CNS-1750009, CNS-1740079 and US ARL Cooperative Agreement W911NF-17-2-0196. All opinions expressed are those of the authors and not necessarily of the US NSF or ARL.

## REFERENCES

Abadi, M. et al. Tensorflow: A system for large-scale machine learning. In *OSDI'16*, pages 265–283. USENIX, 2016.

H. Abbas and G. Fainekos. Linear hybrid system falsification through local search. In *ATVA*, pages 503–510. Springer, 2011.

Matthias Althoff. An introduction to CORA 2015. In *ARCH Workshop*, pages 120–151, 2015.

Y. Annpureddy, C. Liu, G. Fainekos, and S. Sankaranarayanan. S-taliro: A tool for temporal logic falsification for hybrid systems. In *Proceedings of TACAS*, pages 254–257, 2011.

M. Behl, A. Jain, and R. Mangharam. Data-driven modeling, control and tools for cyber-physical energy systems. In *ICCPS*, pages 35:1–35:10. IEEE, 2016.

L. Bottou. Large-scale machine learning with stochastic gradient descent. COMPSTAT, pages 177–186, 2010.

X. Chen, E. Ábrahám, and S. Sankaranarayanan. Flow*: An analyzer for non-linear hybrid systems. In *CAV*, pages 258–263. Springer, 2013.

Y. Deng, A. Rajhans, and A. A. Julius. Strong: A trajectory-based verification toolbox for hybrid systems. In *QEST*, pages 165–168. Springer, 2013.

A. Donzé and O. Maler. Robust satisfaction of temporal logic over real-valued signals. In *FORMATS*, volume 6246 of *LNCS*, pages 92–106. Springer, 2010.

Alexandre Donzé. Breach, a toolbox for verification and parameter synthesis of hybrid systems. In *CAV*, pages 167–170. Springer, 2010.

S. Dutta, S. Jha, S. Sankaranarayanan, and A. Tiwari. Output range analysis for deep neural networks, 2017. arXiv:1709.09130.

G. Fainekos and G. J. Pappas. Robustness of temporal logic specifications for continuous-time signals. *Theoretical Computer Science*, 410:4262–4291, 2009.

C. Fan, B. Qi, S. Mitra, and M. Viswanathan. Dryvr: Data-driven verification and compositional reasoning for automotive systems. In *CAV*, pages 441–461. Springer, 2017.

G. Frehse, C. Le Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler. Spaceex: Scalable verification of hybrid systems. In *CAV*, pages 379–395. Springer, 2011.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

S. Haesaert, P. M. J. Van den Hof, and A. Abate. Data-driven and model-based verification via bayesian identification and reachability analysis. *Automatica*, 79:115–126, 2017.

Z. Hou, R. Chi, and Huijun Gao. An overview of dynamic-linearization-based data-driven control and applications. *IEEE Transactions on Industrial Electronics*, 64(C), 2017a.

Z. Hou, H. Gao, and F. L. Lewis. Data-driven control and learning systems. *IEEE Transactions on Industrial Electronics*, 64(5):4070–4075, May 2017b. ISSN 0278-0046. doi: 10.1109/TIE.2017.2653767.

Z.S. Hou and Z. Wang. From model-based control to data-driven control: Survey, classification and perspective. *Information Sciences*, 235(Supplement C):3 – 35, 2013.

A. Jain, R. Mangharam, and M. Behl. Data predictive control for peak power reduction. In *BuildSys*, pages 109–118. ACM, 2016.

A. Kanade, R. Alur, F. Ivančić, S. Ramesh, S. Sankaranarayanan, and K. C. Shashidhar. Generating and analyzing symbolic traces of simulink/stateflow models. In *CAV*, pages 430–445. Springer, 2009.

G. Katz, C. Barrett, D. Dill, K. Julian, and M. Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In *CAV*, pages 97–117, 2017.

A. Podelski and S. Wagner. Region stability proofs for hybrid systems. In *FORMATS*, pages 320–335. Springer, 2007.

J. C. Spall and J. A. Cristion. Model-free control of nonlinear stochastic systems with discrete-time measurements. *IEEE Transactions on Automatic Control*, 43:1198–1210, 1998.

Z. Wang and D. Liu. Data-based stability analysis of a class of nonlinear discrete-time systems. *Information Sciences*, 235 (C):36 – 44, 2013.

G. Williams, N. Wagener, B. Goldfain, P. Drews, J. M. Rehg, B. Boots, and E. A. Theodorou. Information theoretic mpc for model-based reinforcement learning. In *ICRA*, 2017.