Gray-box Adversarial Testing for Control Systems with Machine Learning Components

Shakiba Yaghoubi
SCIDSE, Arizona State University
Tempe, Arizona
syaghoub@asu.edu

Georgios Fainekos
SCIDSE, Arizona State University
Tempe, Arizona
fainekos@asu.edu

ABSTRACT

Neural Networks (NN) have been proposed in the past as an effective means for both modeling and control of systems with very complex dynamics. However, despite the extensive research, NN-based controllers have not been adopted by the industry for safety critical systems. The primary reason is that systems with learning based controllers are notoriously hard to test and verify. Even harder is the analysis of such systems against system-level specifications. In this paper, we provide a gradient based method for searching the input space of a closed-loop control system in order to find adversarial samples against some system-level requirements. Our experimental results show that combined with randomized search, our method outperforms Simulated Annealing optimization.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning**; • **Mathematics of computing** → *Calculus*; • **Theory of computation** → *Semantics and reasoning*; • **Computer systems organization** → *Embedded and cyber-physical systems.*

KEYWORDS

Testing and Verification, Optimization, neural network

ACM Reference Format:
Shakiba Yaghoubi and Georgios Fainekos. 2019. Gray-box Adversarial Testing for Control Systems with Machine Learning Components. 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, 6 pages. https://doi.org/10.1145/3302504.3311814

1 INTRODUCTION

There is a long history of investigating the application of Neural Networks (NN) in high assurance systems [19]. The advantages of including a NN in the control loop can be substantial. For example, a system may include components with complex dynamics that cannot be modeled by first principles and need to be learned. Most importantly, a high assurance system needs to be able to adapt in catastrophic situations. NNs provide such an adaptation mechanism with only limited assumptions on the structure of what is to be learned. However, even though there has been substantial progress in the stability analysis and verification of such systems [17], the problem of system level verification of transient system behaviors still remains a major challenge.

In this paper, we report on progress on the automatic generation of adversarial test cases (falsification) for nonlinear control systems with NN components in the loop. We assume that system properties -that can be specified using different logics (e.g, [4, 16])- are expressed in Signal Temporal Logic (STL) [4] and we develop a framework that searches for adversarial tests through functional gradient descent. In particular, we propose using a local optimal control based search combined with a global optimizer since the resulting optimization problem is non-convex.

We remark that our proposed approach neither requires analytical information about the system model nor the NN architecture. However, our framework requires information readily available by most model based development tools for control systems. Namely, it requires linearizations of the closed loop system at given operating points. The linearizations help us approximate the gradient descent directions without the need for computing sensitivity matrices or numerical approximations of the descent directions.

We assume that the NNs in the system include differentiable activation functions. This is not a restrictive assumption since most of the common approaches for training NNs are based on gradients which require differentiability, so activation functions are usually approximated to become smooth if they are not. For instance, Rectified Linear Unit (ReLU) is the rectifier function $f(x) = max(0, x)$ whose corner is smoothed out as $\tilde{f}(x) = ln(1 + e^x)$.

Our approach can be used for systems that contain Recurrent Neural Networks (RNN) which cannot be handled by the existing testing and verification methods. Finally, we remark that our proposed method could be extended to hybrid control systems with NNs if results similar to [23, 25] are utilized.

Summary of contributions: We develop an adversarial test generation (falsification) framework for control systems with RNN in the loop based on optimal control theory. Unlike our previous works [22, 24] in which the input signal is parameterized using finite number of parameters, in this work the input is calculated using an optimal-control approach which searches directly in the infinite search space of the input functions. We experimentally demonstrate that our framework vastly outperforms black-box system testing methods. Namely, in our case study, the proposed framework always returns falsifications when the black-box methods fail to do so.

2 RELATED WORK

First and foremost, we highlight that our work utilizes ideas from falsification methods based on optimal control [2, 15, 22, 25]. Similar

HSCC '19, April 16–18, 2019, Montreal, QC, Canada

ACM ISBN 978-1-4503-6282-5/19/04…$15.00
https://doi.org/10.1145/3302504.3311814

to the work in [15], we use well known results in optimal control theory but for systems equipped with NN and against properties that are not necessarily described in the form of standard optimal control cost functionals. In [2], authors study the falsification problem of white-box nonlinear systems with piece-wise constant inputs. They calculate gradient descent directions for system's initial conditions and input parameters using sensitivity analysis. This work has been extended in [22] to include gray-box nonlinear systems using the linearizations of the system model, and in [23] to include hybrid systems. Another search-based method for falsification of Hybrid systems is multiple shooting optimization technique which is studied in [25].

In terms of testing and verification for NN, in [14], a verification framework based on Satisfiability Modulo Theory (SMT) for feed-forward multi-layer neural networks is developed. The framework aims to evaluate the robustness of image classifiers to manipulations. The work is limited to feed-forward networks and proves properties statically: it does not consider the NN in a closed loop system.

In [9], a tool for calculating the reachable set of ReLU[1] feed-forward NNs (FNNs) based on mixed integer linear programs is introduced. This tool is used in [10] for verifying stability and reachability properties of systems with FNNs in a feedback loop. However, NNs in complex dynamical systems may contain more complicated activation functions like sigmoids or Gaussian functions.

Along these lines, the authors in [21] study the problem of safety verification for dynamical systems with FNNs with general activation functions. They develop an algorithm to compute an over-approximation of the reachable set of the neural network control system over a finite time horizon.

Due to the complexity and the growth in the model order, formal verification methods cannot currently be used for control systems with more general NNs than feed-forward. In these applications one need to resort to testing methods. The works in [1, 6, 7, 20] test autonomous vehicles equipped with NNs for perception, guided by system-level requirements.

3 PRELIMINARIES

3.1 Neural Networks

Neural Networks are brain-inspired functions/dynamical systems that can learn to replicate real systems if provided by enough data about that system. NN's consist of input, output and usually hidden layers that each include a number of nodes/neurons connected to transform the input into a suitable signal for replicating the desired output. The input layer passes the inputs to the network, where some computations are applied on them in the hidden layers, and the output layer consists of at least one node that generates the output vector. The inputs to each node are the outputs from other nodes, and the output of each node is computed by applying nonlinear functions to the weighted sum of its inputs. Many methods have been studied in literature to train a NN to replicate a system's behavior, most of which minimize a loss function, such as the mean-squared error of the output. We briefly introduce two types of the most generally used NNs in the following:

[1]ReLU in this work is assumed to be $f(x) = max(0, x)$

3.1.1 Feed forward Neural Networks (FNN). FNNs are the simplest type of NNs. They are static or memory-less networks with no feedback loops. Multi-layer perceptron (MLP) is the most general form of FNN, which has the ability to approximate any nonlinear function (Universal Approximation Theorem [13]). Assuming l layers in the FNN, the ith layer applies the following function to its inputs $u_i \in \mathbb{R}^{m_i}$,

$$y_i = \phi_i(W_i^T u_i + b_i) \quad i \in \{1, 2..., l\} \tag{1}$$

where assuming that the layer has n_i outputs $y_i \in \mathbb{R}^{n_i}$ (usually $n_i = m_{i+1}$), W_i is a $\mathbb{R}^{m_i} \times \mathbb{R}^{n_i}$ weight matrix, $b_i \in \mathbb{R}^{n_i}$ is a bias vector, and $\phi_i : \mathbb{R}^{m_i} \to \mathbb{R}^{n_i}$ is an activation function which is usually one of the continuous nonlinear functions: ReLU, tanh, arctan, logistic or sigmoid. The weight matrices W_i and the bias vectors b_i should be adjusted using a training approach [12]. After the training phase, the function $FNN : \mathbb{R}^{m_1} \to \mathbb{R}^{n_l}$ formed by neurons of Eq. (1), calculates the final output of the feed forward neural net at time t given the input at that time: $y(t) = FNN(u(t))$

3.1.2 Recurrent Neural Networks (RNN). Unlike FNNs, RNNs are dynamic networks. The feedback loops between neurons equip the network with long/short term memory. The output at each time t represented as $y(t) = \text{RNN}(t, u(.))$ is a function of the vectorized input signal/sequence $u(.)$[2] and is a solution to the following continuous or discrete system of equations:

$$\begin{gathered} \dot{x}_{nn} = f_c^r(x_{nn}, u), \quad \text{or} \\ x_{nn}(t) = f_d^r(x_{nn}(t-1), x_{nn}(t-2), ..., u(t)) \\ y(t) = g(x_{nn}(t)) \end{gathered} \tag{2}$$

where x_{nn} is the internal state (memory) of the *RNN* which is usually initially zero ($x_{nn}(0) = 0$). These states are the outputs of the delay/integrator blocks whose inputs are calculated using the functions f_c^r or f_d^r given the input and (previous) states. Note that despite FNN formulation in Eq. (1), the above formulation describes the overall input output relationship of the *RNN* rather than the individual neurons. The *RNN* output at each time t is a function of the states x_{nn} at t.

We denote the solution of an arbitrary NN at time t as $NN(t, u(.))$.

3.2 Closed-loop Control Systems Description

In this paper NNs can be combined with a system plant in a general way. Many of the dynamical systems in which NNs are used for controls (in feedback, feedforward or end-to-end), unmodeled dynamics estimation or predictions, can lie under the class of systems that we consider (shown in Fig. 1). The system is studied in the bounded time interval $[0, T]$ and described in the following.

$$\Sigma : \ \dot{x}_p = f_p(x_p, w, \text{NN}(t, x_p(.), w(.))) \tag{3}$$

where $x_p \in X \subset \mathbb{R}^n$, $x_p(0) \in X_0$, and $w \in U \subset \mathbb{R}^m$ are the system states, state initial values, and inputs, respectively. Also, $x(.)$, $w(.)$ are the state and input trajectories, $\text{NN} : \mathbb{R}_+ \times X^{[0,T]} \times U^{[0,T]} \to \mathbb{R}^k$, and $f_p : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^k \to \mathbb{R}^n$ are C^1 functions. The solution to system (3) at time t with initial condition $x_p(0)$ and input w is denoted by $s_p(t, x_p(0), w)$.

[2]Because of the properties of RNN, the output at each time t' can be dependent on the values of the input signal at any time $t < t'$, so instead of using $u(t)$ that represents the value of signal at time t, we use $u(.)$ to represent the input trajectory

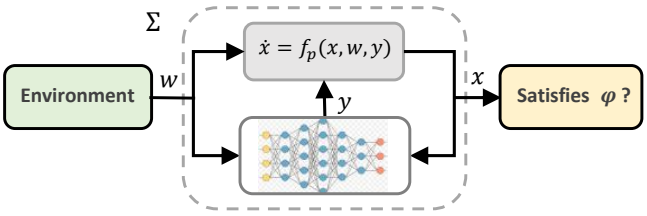

Figure 1: A close-loop control system containing a NN

3.3 Specifications

Desired system behaviors can be specified using Signal Temporal logic (STL) formulas which are reviewed in [4]. These formulas are created by combining *atomic propositions* or *predicates* using logical and temporal operators. Logical operators include: *and* ($\wedge$), *or* ($\vee$), and *not* ($\neg$), and temporal operators include: *always* ($\Box$), *eventually* ($\Diamond$), and *until* ($\mathcal{U}$) that can be combined with time intervals to specify when operators are active.

Given the system state trajectory $s_p(t, x_0, w)$, a robustness value can be calculated with respect to an STL formula φ (see [11]), which shows how well the trajectory satisfies the formula. Positive values indicate satisfaction and negative values indicate violation. The absolute value of the robustness shows how far the trajectory is from being satisfied/falsified.

The robustness value is calculated using max and min functions over the distances of the points on the trajectory from sets that are defined by the formula predicates and as a result the robustness function is not differentiable. In [18] differentiable semantics of logic are defined approximately. The accuracy of the approximation however depends on various parameters and there is not a mature enough tool to calculate the robustness using them yet either. So in the following, we use the approach in [2] to deal with the non-differentiability of the robustness function:

It can be shown that the absolute value of the robustness of the trajectory $s_p(t, x_p(0), w)$ corresponds to the distance between a point $s_p(t_*, x_p(0), w)$ on the trajectory and a point z_* that belongs to a critical set. The critical set corresponds to a predicate in the STL formula φ, and t_* is called the critical time. The variables z_* and t_* are simply calculated using tools such as S-Taliro while evaluating the robustness. The robustness of neighboring trajectories $s_p(t, x'_p(0), w')$ where $x'_p(0) = x_p(0) + \delta x_p(0)$, and $w'(t) = w(t) + \delta w(t)$ is upper bounded by $||s_p(t_*, x'_p(0), w') - z_*||$ so minimizing the following cost w.r.t. $x'_p(0)$ and w' will locally minimize the robustness function. Note that the dependence of the cost function on $x_p(0)$ and w is through z_* and t_*.

$$J_{x_p(0),w} = \frac{1}{2}\big(s_p(t_*, x'_p(0), w') - z_*\big)^\top \big(s_p(t_*, x'_p(0), w') - z_*\big) \quad (4)$$

4 ANALYTICAL ADVERSARIAL TESTING

4.1 Problem Formulation

In adversarial testing, we are interested in finding adversarial $w \in U^{[0,T]}$ and $x_0 \in X_0$ for which the solution to the system (3) does not satisfy a given formula φ. The adversary can be used later to improve the system performance by adapting or retraining the NN (similar to [8]). We look at the problem as a constrained optimization problem in which we minimize the robustness function over X_0 and $U^{[0,T]}$ and under the dynamics of Eq. (3). This optimization problem can be locally solved by minimizing the cost in Eq. (4) instead of the robustness value. Also, we integrate the NN with the plant and rewrite the system in Eq. (3) as:

$$\dot{x} = f(x, w) \quad (5)$$

The solution to system (5) at time t with initial condition $x(0)$ and input w is denoted by $s(t, x(0), w)$. Note that the states of the closed loop system above (x) include the states of the plant ($x_p \in \mathbb{R}^n$) and possible states of the neural network ($x_{nn} \in \mathbb{R}^b, b \geq 0$)[3]. However the system requirements are usually on the plant states rather than the NN states, so the value of the neural net states x_{nn} do not affect the robustness value directly. As a result $z_* \in \mathbb{R}^n$ only concerns x_p and any value of x_{nn} is considered to be desired for falsification. In the rest of the paper, the superscript i shows the variables corresponding to the *i-th* iteration.

Problem 1. *At the ith iteration, given an STL formula φ, an initial condition $x^i_p(0)$[4], and an input signal w^i, find the solution to the system of Eq. (5): $s(t, x^i(0), w^i) \triangleq x^i = [x^i_p, x^i_{nn}]$, where $x^i(0) = [x^i_p(0), zeros(b)]$[5]. Calculate for the formula φ, the critical time t^i_* and the critical point z^i_* corresponding to x^i_p. Let $r^i_* \triangleq [z^i_*, x^i_{nn}(t^i_*)]$, and solve the following constrained minimization problem:*

$$\begin{aligned} \underset{x_p(0), w}{Minimize} \quad & J^i = \frac{1}{2}\big(x(t^i_*) - r^i_*\big)^\top \big(x(t^i_*) - r^i_*\big) \\ s.t \quad & \dot{x} = f(x, w) \\ & x_p(0) \in X_0, w \in U \end{aligned} \quad (6)$$

4.2 Specification Falsification Attack

Due to the nonlinear constraints, finding the global minimizer to Problem (1) cannot be guaranteed. However, taking a small enough step in the direction of the negative of the gradient of the cost function (6) w.r.t x_0 and w, will decrease the cost locally. Using the well known method of the Lagrange multipliers, Problem 1 can be reduced to the problem of minimizing the following cost function:

$$\bar{J}^i = \frac{1}{2}\big(x(t^i_*) - r^i_*\big)^\top \big(x(t^i_*) - r^i_*\big) + \int_0^{t^i_*} \lambda^\top \big(f(x, w) - \frac{dx}{dt}\big) dt$$

Forming the Hamiltonian as $H(x, w) = \lambda^\top f(x, w)$, and $\phi^i(x) = \frac{1}{2}\big(x - r^i_*\big)^\top \big(x - r^i_*\big)$, $\bar{J}^i$ can be written as:

$$\bar{J}^i = \phi^i(x(t^i_*)) + \lambda(0)^\top x(0) - \lambda(t^i_*)^\top x(t^i_*) + \int_0^{t^i_*} \big(H(x, w) + \frac{d\lambda}{dt}^\top x\big) dt$$

As a result, the gradient of the cost function $\bar{J}^i$ is:

$$\begin{aligned} \delta \bar{J}^i = & \Big(\frac{d\phi^i\big(x^i(t^i_*)\big)}{dx} - \lambda^\top(t^i_*)\Big)\delta x(t^i_*) + \lambda^\top(0)\delta x(0) \\ & + \int_0^{t^i_*} \Big(\big(\frac{\partial H}{\partial x} + \dot{\lambda}^\top\big)\delta x + \frac{\partial H}{\partial w}\delta w\Big) dt \end{aligned}$$

[3] These states exist only for RNN (See Sec. 3.1.2).
[4] The initial condition for neural network states x_{nn} are usually considered as zero.
[5] zeros(b) is a vector of b zeros

By updating the co-states λ backward in time with the following final value ordinary differential equation,

$$\dot{\lambda} = -\frac{\partial H}{\partial x}^{\top} = -\frac{\partial f}{\partial x}\Big|_{x^i, w^i}^{\top} \lambda \tag{7}$$

$$\lambda(t_*^i) = \Big(\frac{d\phi^i\big(x^i(t_*^i)\big)}{dx}\Big)^{\top} = x^i(t_*^i) - r_*^i \tag{8}$$

$\delta \bar{J}^i$ is reduced to $\delta \bar{J}^i = \lambda^{\top}(0)\delta x(0) + \int_0^{t_*^i} \frac{\partial H}{\partial w}\delta w \, dt$. The following choices of $\delta x(0)$ and δw with a small enough positive step size h will result in a negative $\delta \bar{J}^i$ and as a result a decrease in $\bar{J}^i$:

$$\delta x^i(0) = -\lambda(0) \tag{9}$$

$$\delta w^i(t) = -\frac{\partial H}{\partial w} = -\frac{\partial f}{\partial w}\Big|_{x^i, w^i}^{\top} \lambda(t) \tag{10}$$

In order to find $\delta x(0)$ and $\delta w(t)$ using Eq. (7-10), we either need to differentiate f w.r.t x and w which requires knowledge about f (or f_p and NN) or we can use modified version of the successive linearization approach introduced in [22]. Recall that linear approximations of f around operating points can usually be provided. Given $x_p^i(0)$ and $w^i(t)$ assume that we take N time samples on the corresponding trajectory and the following is a linear approximation of Eq. (5) at sample time $t_k \in [0, T]$ ($t_1 = 0, t_N = T$)

$$\dot{x} = A_k^i x + B_k^i w \qquad k = 1, ..N$$

where A_k^i, B_k^i are constant matrices. For each time $t \in [t_k, t_{k+1}]$, we calculate the time-varying functions $A^i(t)$ and $B^i(t)$ as follows:

$$\alpha_k = \frac{t_{k+1} - t}{t_{k+1} - t_k}, \quad \alpha_{k+1} = \frac{t - t_k}{t_{k+1} - t_k}$$

$$A^i(t) = \alpha_k A_k^i + \alpha_{k+1} A_{k+1}^i, \quad B^i(t) = \alpha_k B_k^i + \alpha_{k+1} B_{k+1}^k \tag{11}$$

and calculate $\delta x(0)$ and $\delta w(t)$ using the following equations

$$\lambda(t_*^i) = x^i(t_*^i) - r_*^i \tag{12}$$

$$\dot{\lambda} = -A(t)^{\top}\lambda \tag{13}$$

$$\delta x^i(0) = -\lambda(0) \tag{14}$$

$$\delta w^i(t) = -B(t)^{\top}\lambda(t) \tag{15}$$

The linearization matrices A_k^i, B_k^i can be computed analytically or approximated numerically. Similar to the work in [15], using numerical approximaions, our approach can be applied to blackbox systems too. The MATLAB `'Linearize'` command that we use in our implementation can compute the linearizations analytically (using a block-by-block approach) or numerically (using perturbetions) for Simulink models. However, Mathworks strongly recommends that the analytical approach is used as it is faster and more accurate.

Algorithm 1 describes the process of finding adversarial inputs and initial conditions. In this algorithm, InBox is a function that saturates its first input argument to lie in the set which is specified in its second input argument. Note that we can stop the algorithm based on different criteria. The algorithm can be stopped if:

- A maximum number of iterations is reached.
- The change in the robustness is less than a minimum value.
- The changes in the initial conditions and inputs are less than a minimum value.

[6] $\delta x_p^i(0)$ is the non NN part of $\delta x^i(0)$

Algorithm 1 Optimal input and initial condition for falsification

Require: TL formula φ, $x_p^1(0)$, $w^1(t)$, X_0, U, and a tool to extract linearizations of f, and initial step size h_0, and constant $c > 1$.
Ensure: local optimal initial condition x_p^*, local optimal input w^*.

1: Initialize $i = 1, d_* = \infty, h = h_0$
2: Evaluate the system response $x^i(t)$, and find the corresponding robustness value d, and t_*^i, r_*^i.
3: If $d < d_*$ let $d_* = d$, $x_p^*(0) = x_p^i(0)$, $w^* = w^i$, and $h = ch$, otherwise let $h = h/c$ and go to step 6.
4: If $d < 0$ (φ is falsified): stop and return the corresponding $x_p^*(0), w^*$.
5: Linearize the system around sample times taken in $[0, t_*^i]$ and evaluate $\delta x^i(0)$ and δw^i using equations (11-15)
6: While the stop condition is not active, let $x_p^i(0) = InBox(x_p^i(0) + h\,\delta x_p^i(0),\ X_0)$ [6] and $\forall t \in [0, t_*^i]: \ w^i(t) = InBox(w^i(t) + h\,\delta w^i(t),\ U)$ and go back to step 2.
7: Let $i = i + 1$, $\delta x^i(0) = \delta x^{i-1}(0)$ and $\delta w^i = \delta w^{i-1}$

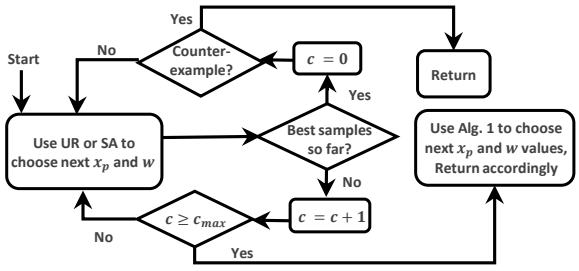

Figure 2: The falsification framework

4.3 Framework

The robustness function is a non-convex non-differentable function in nature. In order to locally solve the problem we defined the function $J_{x_p(0),w}$. However, in order to search for the global minimizer of the robustness function, the gradient based local search still needs to be combined with a "sampling method for coverage" or a "stochastic global optimization" approach. In what follows we combine the local search with Uniform Random Sampling (UR) and Simulated Annealing optimization (SA). The framework is shown in Fig. 2, where $c = 0$ in the beginning and c_{max} is a design choice.

5 CASE STUDIES

In this section we study two systems containing NNs. The NNs serve as controllers and they are trained to replicate the behavior of well-known controllers. Motivated by the fact that Simulink models are widely used in industry for modeling complicated systems, both of our case studies are Simulink models that are treated as gray-box, and the only information that we extract from their model, is the dynamical model linearizations along systems' trajectories that are anyway extractable using the Simulink's linear analysis toolbox.

5.1 Nonlinear system with FNN controller

Consider the following nonlinear system under a FNN controller that has 5 layers and tangent-sigmoid activation functions. Also let

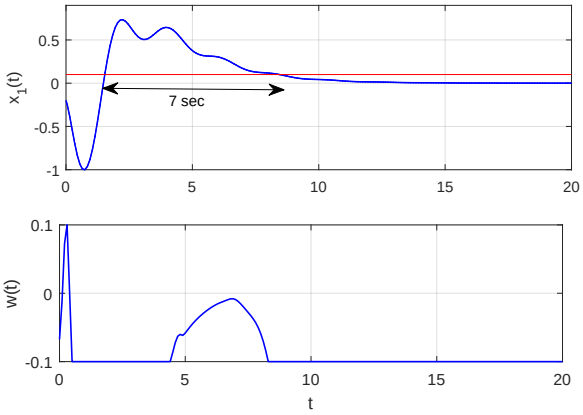

Figure 3: Falsifying Input and trajectory: The trajectory does not settle down below 0.1 within 7 seconds of the rise time.

$x_1(0) = -0.2, x_2(0) = 5$, and $w(t) \in [-0.1, 0.1]$:

$$\dot{x}_1 = -0.5x_1 - 2e^{-0.5t} sin(3t) + sin(x_2)$$

$$\dot{x}_2 = -x_2 + x_1^2(cos(x_2 + w(t)) + FNN(x_1, x_2)$$

The system is tested against the specification:

$$\square\big((x_1(t) < 0 \wedge \diamond_{[0,\epsilon]}\, x_1(t) > 0) \rightarrow \diamond_{[0,7]}\square(x_1(t) < 0.1)\big)$$

in which ϵ is a small positive constant. The requirement requires the signal to always stay below 0.1 within 7 second of the rise time. Starting from $w(t) = 0$ the local optimal search finds an input (shown in Fig. 3) that falsifies the requirement. The robustness for the falsifying trajectory is -7.7×10^{-7}.

5.2 Steam Condenser with RNN Controller

We studied a dynamic model of a steam condenser with 5 continuous states based on energy balance and cooling water mass balance [5] under an RNN controller with 6 discrete states and tangent-sigmoid activation functions. The Simulink model for the system is shown in Fig. 4. The steam flow rate $w(t)$ (Input 1 in Fig. 4) is allowed to vary in the set $[3.99, 4.01]$ and the system is tested for $T = 35$ seconds against the specification $\square_{[30,35]}\, p(t) \in [87, 87.5]$. Starting from a constant valued signal $w(t) = 4$ that results in a robustness value equal to 0.20633, the above approach finds a falsifying trajectory with robustness 0.00030222. The initial and final trajectories and inputs are shown in Fig. (5). Using $w(t) = 3.99$ and $w(t) = 4.01$ initially, the robustness values were reduced from 0.24131 to 0.00033674 and from 0.17133 to 0.0002290, respectively. These nearly falsifying trajectories are the result of very similar inputs with small differences in switch times. These small differences result in slightly different robustness values due to the hard timing constraints in the requirement. While the local search reduces the robustness values significantly in all the above 3 cases, in none of them a falsifying behavior is found. The importance of combining this local search with a global sampler/optimizer becomes more clear in the next section where the combination of the local search with uniform random sampling or Simulated Annealing method finds adversarial examples.

Note that, while the utilized NNs have fairly small number of layers (since they were found to perform good enough during the training phase), the scalability of the proposed approach was tested on the systems of Sec. 5.1 and 5.2 including NN controllers with larger number of layers (20 to 100) too. These experiments showed that the proposed approach scales well. Theoretically increasing the number of layers/neurons in FNNs or the number of non-recurrent layers (with no delay/memory) in RNNs will just increase the number of blocks in the Simulink model linearly. Since MATLAB analytical linearization is computed block-by-block, increasing the number of these kinds of layers (l) increases the linearization complexity by $O(l \times r)$ where r is the maximum number of neurons in layers. However increasing the size of state-space or the number of layers of the RNN with memory increases the linearziation complexity faster. Specifically the size of linearized matrices grow quadratically with the number of state-space plus RNN states. However, in practice, we observed much less increase in the computation time of the overall algorithm when increasing the size of the NN states.

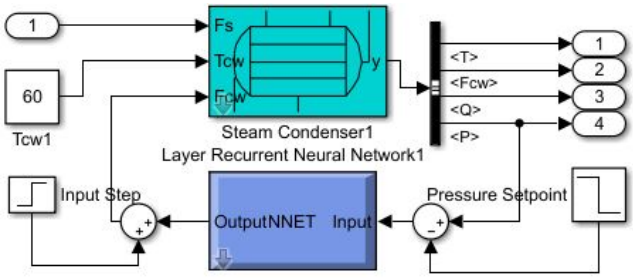

Figure 4: Simulink model for Steam Condenser with Feedback RNN Controller

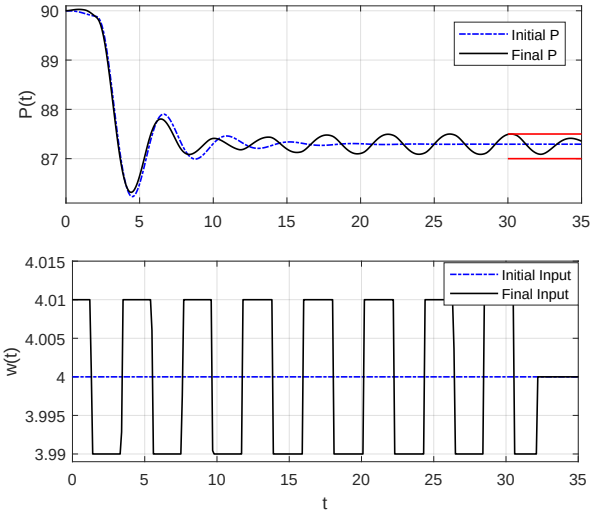

Figure 5: The system robustness is reduced from 0.20633 using a constant input $w(t) = 4$ to 0.00030222 using the local optimal input shown in the picture.

6 EXPERIMENTAL RESULTS

Experiments are conducted using MATLAB 2017a on an Intel(R) Core(TM) i7-4790 CPU @3.6 GHZ with 16 GB memory processor with Windows 10 Enterprise.

Table 1: Falsification Results of Steam Condenser system with RNN controller using different search methods.

	UR	SA	UR+GD	SA+GD
# falsifications	0/50	0/50	50/50	50/50
Avg min robustness	0.0843	0.0503	-0.0018	-0.0016
Avg execution time	>60	>60	15.7812	13.0688
Avg # simulations	600	600	87.48	62.26

We used Uniform the Random Sampling (UR) and Simulated Annealing (SA) implementations of S-Taliro [3] unaided and aided by the optimal local search (UR+GD and SA+GD, respectively) for finding adversarial inputs to the more difficult problem described in Sec. 5.2 with RNN in the loop. For sampling using SA and UR, inputs were (initially[7]) considered to be piece-wise constant signals with 12 control points with varying sample times (total of 24 variables). In the UR+GD implementation, local optimal search is performed when the sampler cannot find a sample with a less robustness value 50 times in a row, and in the SA+GD implementation it is applied when the optimizer cannot find a less robust sample 30 times in a row. We run the experiments 50 times, and in each run the maximum execution time is limited to 60 seconds[8]. The search is initialized with the same seed for all the experiments. The above search methods are compared against the number of falsifications found, average minimum robustness found, average execution time, and average total number of simulations before returning. The improvement in the results from left to right in Table 1 is evident and it motivates the use of the proposed local search. While SA and UR were not able to find any counterexamples in 50 runs, their combination with gradient based descent found an adversarial example in all the runs within a short amount of time and with less than 90 simulations on average.

7 CONCLUSIONS AND FUTURE WORK

In this paper, we proposed a gradient based local search approach for falsification using results from optimal control theory. We applied our method to a case study of a Simulink model of a steam condenser with an RNN controller. The results suggest that when combined with a sampler or a global optimizer, the gradient based local search is indeed very useful in detecting counterexamples to the requirements for control systems with RNNs.

As a future line of work, we will extend our results to falsification of dynamical systems with classifier neural networks. This can be done by manipulating the system's inputs in a way that the NN inputs approach the decision boundary of the falsifying NN output. In a different line of work we will try to augment the adversarial samples into the training set of the NN in order to adapt its gains and improve the system's performance.

ACKNOWLEDGMENTS

This research was partially funded by the NSF awards CNS 1350420 and IIP 1361926, and the NSF I/UCRC Center for Embedded Systems.

[7] when aided by the optimal local search, arbitrary perturbations were applied to inputs
[8] The next sample is taken only if the execution time so far is less than 60 seconds. The algorithm returns faster in case of finding a counter/adversarial example.

REFERENCES

[1] Houssam Abbas, Matthew O'Kelly, Alena Rodionova, and Rahul Mangharam. 2017. Safe At Any Speed: A Simulation-Based Test Harness for Autonomous Vehicles. (2017).

[2] Houssam Abbas, Andrew Winn, Georgios Fainekos, and A. Agung Julius. 2014. Functional Gradient Descent Method for Metric Temporal Logic Specifications. In *American Control Conference*. https://doi.org/10.1109/ACC.2014.6859453

[3] Yashwanth Annpureddy, Che Liu, Georgios Fainekos, and Sriram Sankaranarayanan. 2011. S-taliro: A tool for temporal logic falsification for hybrid systems. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 254–257.

[4] Ezio Bartocci, Jyotirmoy Deshmukh, Alexandre Donze, Georgios Fainekos, Oded Maler, Dejan Nivckovic, and Sriram Sankaranarayanan. 2018. Specification-based monitoring of cyber-physical systems: a survey on theory, tools and applications. In *Lectures on Runtime Verification*. Springer, 135–175.

[5] Yi Cao. [n. d.]. Dynamic Modelling of a Steam Condenser. ([n. d.]).

[6] Tommaso Dreossi, Alexandre Donzé, and Sanjit A Seshia. 2017. Compositional falsification of cyber-physical systems with machine learning components. In *NASA Formal Methods Symposium*. Springer, 357–372.

[7] T. Dreossi, S. Ghosh, A. Sangiovanni-Vincentelli, and S. A. Seshia. 2017. Systematic Testing of Convolutional Neural Networks for Autonomous Driving. In *Reliable Machine Learning in the Wild (RMLW)*.

[8] Tommaso Dreossi, Shromona Ghosh, Xiangyu Yue, Kurt Keutzer, Alberto Sangiovanni-Vincentelli, and Sanjit A Seshia. 2018. Counterexample-guided data augmentation. *arXiv preprint arXiv:1805.06962* (2018).

[9] Souradeep Dutta, Susmit Jha, Sriram Sanakaranarayanan, and Ashish Tiwari. 2017. Output range analysis for deep neural networks. *arXiv preprint arXiv:1709.09130* (2017).

[10] 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.

[11] Georgios Fainekos and George Pappas. 2009. Robustness of temporal logic specifications for continuous-time signals. *Theoretical Computer Science* 410, 42 (2009), 4262–4291.

[12] Simon S Haykin, Simon S Haykin, Simon S Haykin, and Simon S Haykin. 2009. *Neural networks and learning machines*. Vol. 3. Pearson Upper Saddle River.

[13] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. 1989. Multilayer feedforward networks are universal approximators. *Neural networks* 2, 5 (1989), 359–366.

[14] Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. 2017. Safety verification of deep neural networks. In *International Conference on Computer Aided Verification*. Springer, 3–29.

[15] Nan Li, Anouck Girard, and Ilya Kolmanovsky. [n. d.]. Optimal Control Based Falsification of Unknown Systems with Time Delays: A Gasoline Engine A/F Ratio Control Case StudyâŃĘ. ([n. d.]).

[16] Mohammadreza Mehrabian et al. 2017. Timestamp Temporal Logic (TTL) for Testing the Timing of Cyber-Physical Systems. *ACM Transactions on Embedded Computing Systems (TECS)* 16, 5s (2017), 169.

[17] Nhan T. Nguyen and Stephen A. Jacklin. 2010. *Stability, Convergence, and Verification and Validation Challenges of Neural Net Adaptive Flight Control*. SCI, Vol. 268. Springer, 77–110.

[18] Yash Vardhan Pant, Houssam Abbas, and Rahul Mangharam. 2017. Smooth operator: Control using the smooth robustness of temporal logic. In *Control Technology and Applications (CCTA), 2017 IEEE Conference on*. IEEE, 1235–1240.

[19] Johann Schumann and Yan Liu. 2010. *Applications of Neural Networks in High Assurance Systems*. SCI, Vol. 268. Springer.

[20] Cumhur Erkan Tuncali, Georgios Fainekos, Hisahiro Ito, and James Kapinski. 2018. Simulation-based Adversarial Test Generation for Autonomous Vehicles with Machine Learning Components. *arXiv preprint arXiv:1804.06760* (2018).

[21] Weiming Xiang and Taylor T Johnson. 2018. Reachability Analysis and Safety Verification for Neural Network Control Systems. *arXiv preprint arXiv:1805.09944* (2018).

[22] Shakiba Yaghoubi and Georgios Fainekos. 2017. Hybrid approximate gradient and stochastic descent for falsification of nonlinear systems. In *American Control Conference (ACC), 2017*. IEEE, 529–534.

[23] Shakiba Yaghoubi and Georgios Fainekos. 2017. Local Descent for Temporal Logic Falsification of Cyber-Physical Systems. In *Seventh Workshop on Design, Modeling and Evaluation of Cyber Physical Systems*.

[24] Shakiba Yaghoubi and Georgios Fainekos. 2018. Falsification of Temporal Logic Requirements Using Gradient Based Local Search in Space and Time. *IFAC-PapersOnLine* 51, 16 (2018), 103–108.

[25] Aditya Zutshi, Jyotirmoy V Deshmukh, Sriram Sankaranarayanan, and James Kapinski. 2014. Multiple shooting, cegar-based falsification for hybrid systems. In *Proceedings of the 14th International Conference on Embedded Software*. ACM.