Worst-case Satisfaction of STL Specifications Using Feedforward Neural Network Controllers: A Lagrange Multipliers Approach

SHAKIBA YAGHOUBI and GEORGIOS FAINEKOS, CIDSE, Arizona State University

In this paper, a reinforcement learning approach for designing feedback neural network controllers for nonlinear systems is proposed. Given a Signal Temporal Logic (STL) specification which needs to be satisfied by the system over a set of initial conditions, the neural network parameters are tuned in order to maximize the satisfaction of the STL formula. The framework is based on a max-min formulation of the robustness of the STL formula. The maximization is solved through a Lagrange multipliers method, while the minimization corresponds to a falsification problem. We present our results on a vehicle and a quadrotor model and demonstrate that our approach reduces the training time more than 50 percent compared to the baseline approach.

CCS Concepts: • **Theory of computation → Adversarial learning**; • **Computer systems organization → Robotic control**;

Additional Key Words and Phrases: Reinforcement learning, signal temporal logic, neural network controller

ACM Reference format:
Shakiba Yaghoubi and Georgios Fainekos. 2019. Worst-case Satisfaction of STL Specifications Using Feedforward Neural Network Controllers: A Lagrange Multipliers Approach. *ACM Trans. Embed. Comput. Syst.* 18, 5s, Article 107 (October 2019), 20 pages.
https://doi.org/10.1145/3358239

1 INTRODUCTION

The impressive performance of Artificial Neural Networks (NN) in perception and more general artificial intelligence applications has led researchers to revisit their application to control systems as well. NN can play a critical role in autonomous ground and aerial vehicles, robots, and energy systems [10–12, 41, 59]. They can be used for identification and behavior prediction of complicated plants, or learning complex control laws in a reinforcement learning fashion. Classic nonlinear control methods [33], typically require the dynamical systems to operate near their stability region which reduces their maneuverability. On the other hand, Model Predictive Control (MPC) and other optimal control methods which need to be designed online, usually do not scale to higher

This work was partially supported by the NSF awards CNS 1350420, IIP-1361926, and the NSF I/UCRC Center for Embedded Systems.
This article appears as part of the ESWEEK-TECS special issue and was presented at the International Conference on Embedded Software (EMSOFT) 2019.
Authors' addresses: S. Yaghoubi and G. Fainekos, School of Computing, Informatics, and Decision Systems Engineering, Arizona State University, Tempe, Arizona, 85281; emails: {syaghoub, fainekos}@asu.edu.

1539-9087/2019/10-ART107 $15.00
https://doi.org/10.1145/3358239

dimensional systems and computing and storing them offline is not possible either as it requires plenty of memory. However, neural network controllers can be computed offline and used online, and since their complexity is not directly dependent on the system dimension or the complexity of their task, they can be used to accomplish complex tasks on embedded computers.

In all these applications, the closed-loop system (including the NN) should exhibit some desired properties to be reliable. The growing use of NNs especially Deep Neural Networks (DNN) in domains such as safety-critical systems, where reliability is a big concern, creates the need for testing and verification of these data-driven controllers.

Designing provably correct NN-based systems is notoriously difficult. One of the difficulties arises from the complexity of the underlying NN in terms of the architecture and the number of parameters. This complexity makes characterizing all the behaviors of the NN difficult. As a result, training complex neural networks (especially in a reinforcement learning fashion) and verifying the closed-loop system including them is theoretically challenging. As a result, the interest in testing (a semi-formal verification method) systems with machine learning components has increased significantly lately [51, 57]. Most of the previous works that study testing and verification of systems with NNs focus on the ML components only (e.g. [20, 46] and the references therein), and do not consider the interconnected closed-loop behavior of the system. However, usually, the ultimate goal is to evaluate system-level behaviors. In contrast, in this paper, we study the system-level behaviors of NN-based control systems in closed loop.

To evaluate the system-level behaviors, system properties should be specified in a logic formalism (e.g. [34, 37, 38]). In this paper we use Signal Temporal Logic (STL) [37] which is very powerful for specifying complicated properties on dense time, real-valued signals. These specifications can be much more complicated than simple stability, or reachability properties.

In this paper, we propose a framework to design state/output feedback neural network controllers that satisfy system specifications described in STL. The NN is trained to maximize the worst-case robustness of the STL formula [15, 22] in 2 phases. In the first phase, it is trained using a set of random samples. Given the large space of the NN parameters, and the non-convexity of the objective function, in order to balance the exploration vs exploitation, we use a combination of a global search approach to explore the search space, and a local search approach based on gradients to further improve the parameters.

The challenges in using gradient-based approaches for NN parameter tuning were two-fold:

(1) The objective function (robustness function) is inherently non-differentiable. We managed this issue by using the smooth approximation of the robustness function [42].
(2) Backpropagation using automatic differentiation based on the chain rule along long temporal sequences causes the problem of vanishing and exploding gradients [44]. This is due to the fact that in each temporal step, the number of the gradients of the system flow ($\frac{\partial f}{\partial x}$, $\frac{\partial f}{\partial u}$), which are multiplied by each-other, grows. Since these terms are usually not equal to one, their iterative multiplication eventually grows very large or approaches zero. To avoid this issue, we calculate the desired changes in the NN parameters using a method based on state-dependent Lagrange multipliers [6].

In order to increase the system's reliability, in the second phase, the closed-loop system is tested against the STL specification using a falsification tool (S-TaLiRo [4]). Falsifying samples are added to the sample set and the NN is retrained on the updated sample set. The overall learning framework is shown in Figure 1.

Our contributions can be summarized as follows:

- We design state/output feedback NN controllers to satisfy system properties specified in STL.

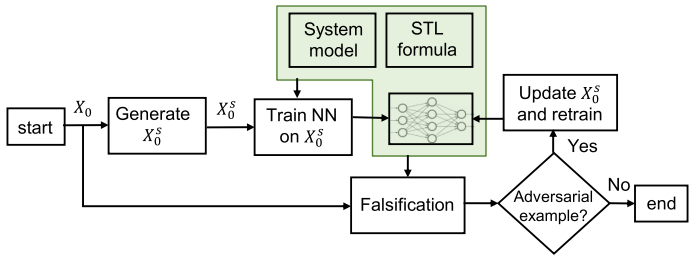

Fig. 1. Training framework.

- To improve the performance of the NN, we test the closed-loop system (including the NN) against the STL requirements. Any discovered adversarial examples in the testing phase are added into the training set and used to retrain the NN. This will allow efficient retraining using rare and important samples which may not be captured even with a dense sampling.
- We provide a formulation for gradient-based improvement of the NN parameters which is numerically stable, i.e, it is not prone to the problem of vanishing and exploding gradients. This can be considered as the main contribution of the paper.

The application of our approach is shown using examples of a 3 dimensional model of a vehicle and a 6 dimensional model of a quadrotor that each need to accomplish missions specified in an STL formula.

2 RELATED WORK

Designing controllers for satisfying system properties specified in STL has been considered before. In [47], a method for designing open-loop controllers for STL specifications based on MPC using Mixed Integer Linear Program (MILP) solvers is introduced. Authors design open-loop strategies that satisfy STL specifications using randomized tree search in [52]. The work in [42] introduces a smooth approximation to the robustness function of the STL formula, which allows using gradient-based optimizers to design robust controllers for the satisfaction of STL formulas. Sequential Quadratic programs are used in [43] in order to design motion plans for quadrotors that satisfy missions specified in STL. In [9], an MPC controller is learned to satisfy STL specifications with non-strict compliance. In general, open-loop controllers need to be designed and stored in advance, and MPC controllers need to be computed at the run-time. Computing MPC controllers for high-order systems include solving large optimization problems at each step [28], so in general, they do not scale to higher-order systems since their run-time computation is challenging. On the other hand, storing the results of the offline computation of open-loop or MPC controllers (e.g., as a lookup table) requires too much memory [32], so using Neural networks (NN) to approximate controllers [31, 49], learn optimal policies offline [25], or to improve baseline controllers [59] is very common. The application of NNs in high assurance systems has been studied in [50]. In this work, we will present a framework to learn optimal policies offline using a Neural Network controller which does not require much memory.

Model-based Reinforcement Learning methods are also another line of work close to our work [35, 54, 58]. Recently, Reinforcement Learning methods have been developed that make policy learning w.r.t temporal logic specifications possible, see e.g., [24, 36]. Despite the recent advances using Reinforcement Learning methods, providing guarantees for the behavior of systems trained using them is challenging. This is due to the complicated behavior of the Neural Networks which are used in these systems as function approximators. These systems sometimes have safety-critical roles, and as a result, the problem of testing and system-level verification is of utmost importance.

Some classic tools for verification that rely on over-approximation of reachable sets include Flow* [8], CORA [3], and SpaceEx [23]. Verification methods for systems including neural networks are studied in [21, 30, 55]. However, verification approaches are usually unsuccessful when the complexity and dimension of the system increases.

Because of the limitations that verification approaches usually have in the application, testing methods are used to find falsifying system behaviors. S-TaLiRo [4], and Breach [14] are general-purpose tools for falsification. The problem of testing autonomous vehicles equipped with NNs for perception, guided by system-level requirements are studied in [1, 16, 51].

The falsifying/adversarial samples that are found during the testing process can be used for retraining the Neural nets, in order to improve their accuracy. Adversarial training is used in [17, 18] for improving the performance of the NNs used in an autonomous vehicle for perception. We will also use adversarial training to improve the Neural Network controller which is initially trained using a set of random samples. The problem of adversarial training in our paper can be looked at as a min-max game, in which adversaries and the NN play against each other: adversaries minimize the robustness of the STL formula to falsify it, and the NN tries to adapt and increase the robustness when it is retrained accordingly. A zero-sum game is formulated in [48, 53] to design STL controllers that maximize the worst-case robustness value.

The work in [21] can be viewed as the closest work to our work wherein Neural network controllers are designed to satisfy reachability and region stability properties. Using a smooth approximation of the STL robustness function, we will design controllers for general STL formulas. Another difference is that in order to guarantee safe worst-case performance, our neural networks are trained to maximize the worst-case robustness values. Besides, in a post-training procedure, our algorithm searches for adversarial samples and retrain the NN using them. Another main contribution of the paper over the aforementioned work is providing a gradient-based backpropagation formulation which is not prone to vanishing and exploding gradients while backpropagation using automatic differentiation is.

3 PRELIMINARIES

We consider a discrete time dynamical system Σ

$$\begin{cases} x_{t+1} = f(x_t, u_t) \\ y_t = g(x_t) \end{cases} \tag{1}$$

where $x_t \in X \subset \mathbb{R}^{n_x}$ is the system state, $y_t \in \mathbb{R}^{n_y}$ is the system output, $u_t \in U \subset \mathbb{R}^{n_u}$ is the control input, $f : X \times U \to X$ is a differentiable function of its arguments, and $x_0 \in X_0 \subset X$ is the initial state. Given an initial state $x_0 \in X_0$ and a control sequence $\mathbf{u}^{\mathrm{N}} = u_0, u_1, \dots u_N$, s.t, $\forall t : u_t \in U$, the bounded time solution of the system is denoted as $\mathbf{x}(x_0, u^N) = x_0, x_1, \dots, x_N$, where $\forall t : x_t \in X$ and $x_{t+1} = f(x_t, u_t)$.

3.1 Controllers for Robust Satisfaction of Signal Temporal Logic Specifications

The controller that computes u_t should be designed such that the closed-loop system of Equation (1) robustly satisfies a specification expressed as a Signal Temporal Logic (STL) [37] formula.

3.1.1 Signal Temporal Logic (STL). STL is a logical formalism that allows efficient and unambiguous specifications of a wide variety of desired system properties beyond basic properties like stability and reachability. An STL formula is a composition of temporal and Boolean operations over predicates over signals. A predicate p_k represents a set of states defined using a real-valued function $\mu : X \to \mathbb{R}$ as $P_k = \{x \in X \mid \mu(x) > 0\}$[1]. Let $P = \{p_1, \dots p_L\}$ be the set of all predicate

[1]The STL formula and its predicates can be defined over the system outputs, as well.

expressions of interest for an STL formula and $\mathcal{I} \subset \mathbb{R}_+$ be any non-empty interval. The set of all well-formed STL formulas φ is inductively defined as

$$\varphi \ := \ \top \mid p \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \, \mathcal{U}_{\mathcal{I}} \psi,$$

where $p \in P$, and $\top$, $\neg$ and $\wedge$ are Boolean *true*, *negation* and *and* operations, and $\mathcal{U}$ is the until temporal operator, which requires ψ to be satisfied at some time in $\mathcal{I}$ and until then, φ needs to be satisfied. The validity of a formula φ with respect to a signal $\mathbf{x}$ at time t is defined inductively as follows:

$$\begin{aligned}
(\mathbf{x}, t) &\models p_k \Leftrightarrow \mu_k(x_t) > 0 \\
(\mathbf{x}, t) &\models \neg\varphi \Leftrightarrow \neg((\mathbf{x}, t) \models \varphi) \\
(\mathbf{x}, t) &\models \varphi \wedge \psi \Leftrightarrow (\mathbf{x}, t) \models \varphi \wedge (\mathbf{x}, t) \models \psi \\
(\mathbf{x}, t) &\models \varphi \, \mathcal{U}_{\mathcal{I}} \psi \Leftrightarrow \exists t' \in (t + \mathcal{I}) \ s.t. \ (\mathbf{x}, t') \models \psi \wedge \forall t'' \in [t, t'), (\mathbf{x}, t'') \models \varphi
\end{aligned}$$

where $(t + \mathcal{I}) = \{t + c \mid c \in \mathcal{I}\}$. Other operators like disjunction ($\vee$), Always ($\Box$) and Eventually ($\Diamond$) can be defined using the above operators (see [5]). The trajectory $\mathbf{x}$ satisfies φ if $(\mathbf{x}, 0) \models \varphi$. An STL formula is bounded-time if all its temporal intervals are bounded.

Definition 3.1. The maximum trajectory length N required to decide satisfiability of an STL formula is called the formula horizon. The formula horizon is the maximum over the sums of all the nested upper bounds on the temporal operators.

For instance the horizon of the formula $\Box_{[0,t_1]}(\Diamond_{[0,t_2]} p)$ is equal to $t_1 + t_2$. In this paper, we consider bounded time STL formulas.

A desired controller for the system, maximizes the satisfaction of an STL in order to account for suitable reactions to disturbances and other uncertainties. The degree of satisfaction of an STL formula [15] which we call the robustness value, can be calculated using a real-valued function ρ_φ of a trajectory $\mathbf{x}$, and t such that $(\mathbf{x}, t) \models \varphi \equiv \rho_\varphi(\mathbf{x}, t) > 0$.

$$\begin{aligned}
\rho_{p_k}(\mathbf{x}, t) &= \mu_k(x_t) \\
\rho_{\neg\varphi}(\mathbf{x}, t) &= -\rho_\varphi(\mathbf{x}, t) \\
\rho_{\varphi\wedge\psi}(\mathbf{x}, t) &= \min(\rho_\varphi(\mathbf{x}, t), \rho_\psi(\mathbf{x}, t)) \\
\rho_{\varphi \, \mathcal{U}_{\mathcal{I}} \psi}(\mathbf{x}, t) &= \max_{t' \in (t+\mathcal{I})} \left(\rho_\psi(\mathbf{x}, t'), \min_{t'' \in [t, t')} \rho_\varphi(\mathbf{x}, t'') \right)
\end{aligned}$$

Temporal operators like Always and Eventually, can be treated as conjunctions and disjunctions along the time axis.

Remark. Note that the interpretation of robustness as defined above (and in [15]) can differ from the notion of the robustness as defined in [22] for general functions of the form $\mu_k(x) > 0$. Using the interpretation in [22], the robustness w.r.t a predicate p_k, is the signed distance (see [22] for more information) of x from the predicate set $P_k = \{x \in X | \mu_k(x) > 0\}$. However in [15], the robustness w.r.t this predicate is simply defined as $\mu_k(x)$ which fails to define a robustness tube around the signal within which all the trajectories satisfy the property. That is we should expect the same robustness value for x no matter $\mu_k(x) = x$ or $\mu_k(x) = 2x$. Nevertheless, for both semantics, positive sign of the robustness implies Boolean satisfaction and negative sign implies falsification.

In spite of the above discussion, in this paper, we choose to work with the STL semantics [15] as they're easier to analyze and differentiate. In order to get robustness values that reflect the radius of robustness tubes around trajectories, we only use predicate functions of the form $\mu_k(x) = x - c$

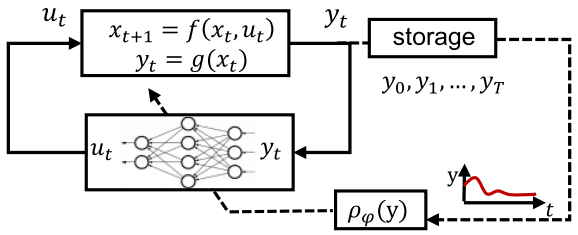

Fig. 2. The closed-loop system with its NN controller to satisfy an STL formula φ.

where c is a constant. The robustness interpretation over the predicates of this form are the same using both methods.

3.1.2 Neural Network Controllers for Robust Satisfaction of STL. Consider $h : \mathbb{R}^{n_y} \rightarrow U$ to be a differentiable function. The controller $u_t = h(y_t)$ should be designed in order to maximize the robustness value. Since the function spaces are infinite-dimensional, the function h needs to be parameterized in order to decrease the complexity of the optimization problem. As neural networks are known to be universal function approximators [29], we consider h to be a Feedforward Neural Network (FNN) parameterized using its weights.

Definition 3.2. Feedforward Neural Networks (FNN) FNNs are static or memory-less networks. The most general FNN is Multi-Layer Perceptron (MLP), which can approximate any nonlinear function. We can define FNNs using the number of their inputs n_I, outputs n_O, layers n_L, and weights $(W_i, b_i)_{i=1}^{n_L}$ which connect the i^{th} layer to the $i^{th}+1$ layer or the outputs. The i^{th} layer applies the following function to its inputs $\mathcal{U}_i \in \mathbb{R}^{m_i}$:

$$\mathcal{U}_{i+1} = \zeta_i(W_i^T \mathcal{U}_i + b_i) \quad i \in \{1, 2 \ldots, l\} \tag{2}$$

where ϕ_i is an activation function chosen to be 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 in order to maximize the robustness value. The function $\mathcal{N}$ formed by the decomposition of the neurons in Equation (2), calculates the final output of the FNN at time t given the inputs at that time and $(W_i, b_i)_{i=1}^{n_L}$. The weights W_i and the bias vectors b_i are collectively denoted as W. We write the output of the NN as $u_t = \mathcal{N}(y_t, W)$. As a result, the closed loop system (shown in Figure 2) can be described as:

$$\begin{cases} x_{t+1} = f(x_t, \mathcal{N}(y_t, W)) \\ y_t = g(x_t) \end{cases} \tag{3}$$

Given an initial condition $x_0 \in X_0$, the bounded time solution of the closed loop system (3) is $\mathbf{x}(x_0, W) = x_0, x_1, \ldots x_N$, where $\forall t : x_t \in X$, $y_t = g(x_t)$, and $x_{t+1} = f(x_t, \mathcal{N}(y_t, W))$.

3.1.3 Smooth Approximation of STL Robustness. The robustness function is a non-differentiable function of its inputs, as it consists of composition of max and min functions. As a result, in order to avoid less efficient optimizers like non-smooth optimizers, stochastic heuristics, or Mixed-Integer Programming solvers, we work with the smooth approximation [42] of the robustness function $\tilde{\rho}_\varphi$ which replaces the min and max functions with soft min ($\overline{\min}$) and soft max ($\overline{\max}$) functions:

$$\overline{\min}(x, \alpha) = -\frac{1}{\alpha} ln\left(\Sigma_{i=1}^{N} e^{-\alpha x_t}\right) \tag{4}$$

$$\overline{\max}(x, \alpha) = -\overline{\min}(-x, \alpha) \tag{5}$$

where x_t is the t^{th} argument of x. This function approaches the min function as $\alpha \to \infty$. It's been shown in [42] that an ϵ can be computed such that $|\rho_\varphi - \tilde{\rho}_\varphi| < \epsilon$. As a result, $\tilde{\rho}_\varphi > \epsilon$ guarantees satisfaction of φ.

4 PROBLEM STATEMENT

Consider the dynamical system $\mathcal{S} = (\Sigma, X_0, U)$, where Σ is described in Equation (1), X_0 is a set of initial conditions, and U is the set of admissible control inputs. We would like to design a NN controller $\mathcal{N}$ for the system $\mathcal{S}$ that satisfies the input constraint such that the minimum robustness value over the set of initial conditions X_0 is maximized. The problem is formally defined as follows:

Problem 1. *Given the dynamical system $\mathcal{S}$, an STL formula φ whose horizon is N, and a fixed architecture for the neural network $\mathcal{N}$, solve the following optimization problem:*

$$W^* = \underset{W}{argmax} \min_{x_0 \in X_0} \rho_\varphi(\mathbf{x}(x_0, u^N)) \tag{6}$$

$$s.t \quad \begin{cases} x_{t+1} = f(x_t, u_t) \\ y_t = g(x_t) \\ u_t = \mathcal{N}(y_t, W) \\ \mathbf{x}(x_0, u^N) = x_0, x_1, \ldots, x_N \end{cases}$$

5 TRAINING THE NEURAL NETWORK CONTROLLER FOR STL SATISFACTION

The NN weights W have a complex chained effect on the cost function of Problem 1 through the system state trajectory. Based on our experiments, direct policy search variants of reinforcement learning approaches based on evolution strategies [27], like covariance matrix adaptation evolution strategy (CMA-ES) algorithm, were not able to find a good set of parameters (W) without additional support. Nonetheless, their exploration power is usually able to bring the parameters close to their optimal value. On the other hand, with a good initialization, gradient based approaches were found to be very effective in training neural networks using back propagation approaches [7].

In this paper, we use the best of two worlds: the exploration power of the evolution based methods with a large enough population size, and the exploitation power of gradient based methods.

5.1 Neural Network Controller Architecture

We assume a given neural network architecture $\mathcal{N}$ for the controller with $n_I = n_y$, $n_O = n_u$, which satisfies the input constraints. In other words, $\mathcal{N}$ is a function with a range U ($\mathcal{N} : \mathbb{R}^{n_y} \to U$). In order to satisfy the input constraints, the output layer can apply a scaled Hyperbolic tangent sigmoid transfer function (see Figure 3) to its inputs. Assuming $\overline{u}$ and $\underline{u}$ to be the upper and lower bounds on U respectively, using the following activation function ensures the satisfaction of the input constraint:

$$\overline{tansig}(x, U) = \underline{u} + \frac{\overline{u} - \underline{u}}{2}(tansig(x) + 1) \tag{7}$$

Remark. If an output feedback control law exists that satisfies the STL formula $\forall x_0 \in X_0$, then provided enough data, and using an appropriate neural network architecture, optimal weights W can be found such that the system of Equation (3) satisfies φ for all $x_0 \in X_0$. Note however that, in general, for satisfying the temporal and reactive properties of STL formulas, more features than the system's current states/outputs, or a controller with memory may be required.

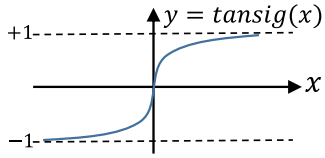

Fig. 3. Tangent Sigmoid function.

5.2 NN Weights Initialization Using Global Optimization

In this step of the training, we look for a set of parameters that on average maximizes ρ_φ for a set of randomly selected initial states $x_0 \in X_0$. We denote this finite set of initial states by X_0^s and its size, i.e, number of samples with $|X_0^s|$. Starting from a set of randomized weights and bias values W, a global optimizer, minimizes the following loss (cost) function

$$C(W) = \sum_{x_0 \in X_0^s} c_\varphi(\mathbf{x}(x_0, W)) \tag{8}$$

where $c_\varphi = -\rho_\varphi$.

In the initialization stage, the above cost function can be minimized using any method for global optimization. In this paper, we use the CMA-ES algorithm [27], which is a derivative-free, evolution based numerical optimization method. CMA-ES is primarily a local optimization approach, but it has also been reported to be reliable and highly competitive for global optimization when using larger population sizes [26]. In our experiments, we use the following relation[2] in order to choose the population size

$$\text{population size} = 10 \times (4 + round(3 \times ln(|W|))) \tag{9}$$

where ln is the natural logarithm function and $|W|$ is the number of NN weights and bias parameters.

Note that a global minimizer W_G of the cost function in Equation (8) does not necessarily maximize the worst case robustness value as required in Problem 1. The reason is that, firstly, a finite number of randomly selected initial conditions are considered in the cost formulation, and secondly, W_G^* is trained on the average performance rather than the worst case performance. Specifically, using $\mathcal{N}(y_t, W_G^*)$ as the controller:

(1) There may exist $x_0 \in X_0 - X_0^s$ such that the solution to the closed loop system (3) does not satisfy the specification, since the training is done using a finite set of initial states.
(2) Or, there may exist adversarial samples $x_0 \in X_0^s$, as the cost formulation is over the average robustness value and not the minimum (worst-case) robustness value.

As a result, in Section 5.5, we design a two-player game that uses adversarial examples to improve the weights of the neural network.

5.3 NN Weight Update Using Lagrange Multipliers for Incremental Robustness

In model-based Reinforcement Learning, backpropagation of the desired change of the objective function to the weights of the NN is used for policy improvement [13].

[2]This is suggested in the Matlab's implementation of CMA-ES.

In practice, direct backpropagation into the policy using automatic differentiation methods (through the unwinding of the states as described in [21]) has shortcomings like numerical instability. This numerical instability causes vanishing or exploding gradients, especially when working with long temporal sequences of states[3]. Mathematically, the reason is that when calculating the gradient of the cost function w.r.t W using the chain rule, in each time step, the derivative calculated in the previous time step should be multiplied with the Jacobin matrix of the dynamics $\frac{\partial f}{\partial x}$. If the eigenvalues of the Jacobian matrix are smaller than one, gradients will vanish over time, and if they are greater than 1 gradient will explode over time. As a result, researchers usually solve the constrained – by the system dynamics – problem using relaxations [35, 39].

When training Recurrent NNs (RNN) that approximate time series, one way to deal with this problem is redesigning them to have fewer layers and memory blocks. However, in our problem, the memory comes from physical laws and system dynamics that cannot be altered. Also, we may require long sequences to decide on the satisfiability of the STL formula (see Definition 3.1 on the formula horizon) when designing a controller that satisfies the specification. Consequently, training algorithms based on automated differentiation may face numerical instability which prevents effective training and wastes computation effort. As a result, in what follows, we derive a weight update law based on an optimal control approach which does not need recursive multiplication of the gradients - as required in direct differentiation based on the chain rule - and incrementally improves the robustness value.

As gradient-based approaches require differentiability of the objective function, we alter the optimization problem of Equation (6) in Problem 1, as follows:

$$\max_{W} \overline{\min_{x_0 \in X_0^s}} \tilde{\rho}_\varphi(\mathbf{x}(x_0, u^N)) \tag{10}$$
$$s.t \quad \begin{cases} x_{t+1} = f(x_t, \mathcal{N}(g(x_t), W)) \\ \mathbf{x}(x_0, u^N) = x_0, x_1, \ldots, x_N \end{cases}$$

Consider the following objective function for one of the trajectories starting from $x_0 \in X_0^s$ and subject to the dynamical constraints:

$$\begin{cases} J_{x_0}(W) = \tilde{\rho}_\varphi(x_0, x_1, \ldots, x_N), \\ x_{t+1} = f(x_t, \mathcal{N}(g(x_t), W)) \end{cases}$$

using the Lagrange multipliers associated with the state equations which are called co-states, the dynamical constraints can be incorporated into the objective function, as follows:

$$\begin{aligned} \bar{J}_{x_0}(W) = \tilde{\rho}_\varphi(x_0, x_1, \ldots, x_N) + \sum_{t=0}^{N-1} \lambda_{t+1}^\top \Big(f(x_t, \mathcal{N}(g(x_t), W))) - x_{t+1} \Big) \\ = \tilde{\rho}_\varphi(x_0, x_1, \ldots, x_N) + H_0 - \lambda_N^\top x_N + \sum_{t=1}^{N-1} \Big(H_t(x_t, W) - \lambda_t^\top x_t \Big) \end{aligned} \tag{11}$$

where λ_ts are the co-states, and $H_t = \lambda_{t+1}^\top f(x_t, \mathcal{N}(g(x_t), W))$ is the Hamiltonian. Thus, we have:

$$\delta \bar{J}_{x_0}(W) = \sum_{t=1}^{N-1} \left(\frac{\partial H_t}{\partial x_t} + \frac{\partial \tilde{\rho}_\varphi}{\partial x_t} - \lambda_t^\top \right) \delta x_t + \left(\frac{\partial \tilde{\rho}_\varphi}{\partial x_N} - \lambda_N^\top \right) \delta x_N + \sum_{t=0}^{N-1} \frac{\partial H_t}{\partial W} \delta W \tag{12}$$

[3]The same problem happens when training Recurrent Neural networks with backpropagation through time [45].

The following equations enforce $\delta \bar{J}_{x_0}(W) \geq 0$:

$$\lambda_N^\top = \frac{\partial \tilde{\rho}_\varphi}{\partial x_N}, \tag{13}$$

$$\lambda_t^\top = \frac{\partial H_t}{\partial x_t} + \frac{\partial \tilde{\rho}_\varphi}{\partial x_t} = \lambda_{t+1}^\top \left(\frac{\partial f}{\partial x_t} + \frac{\partial f}{\partial u_t} \frac{d\mathcal{N}}{dx_t} \right) + \frac{\partial \tilde{\rho}_\varphi}{\partial x_t}, \tag{14}$$

$$\delta W = \sum_{t=0}^{N-1} \frac{\partial H_t}{\partial W} = \sum_{t=0}^{N-1} \lambda_{t+1} \frac{\partial f}{\partial u_t} \frac{\partial \mathcal{N}}{\partial W} \tag{15}$$

where $\frac{\partial f}{\partial x_t} = \frac{\partial f}{\partial x_t}\Big|_{(x_t, u_t)}$, is the Jacobian matrix of the open loop system, $\frac{\partial f}{\partial u_t} = \frac{\partial f}{\partial u_t}\Big|_{(x_t, u_t)}$ is the derivative of the open loop system w.r.t its inputs, $\frac{\partial \tilde{\rho}_\varphi}{\partial x_t}$ is the derivative of the smooth robustness function to its t^{th} argument, $\frac{d\mathcal{N}}{dx_t} = \frac{\partial \mathcal{N}}{\partial y_t} \frac{\partial g}{\partial x_t}$ is the derivative of the NN to its inputs multiplied by their derivative to the states, and $\frac{\partial \mathcal{N}}{\partial W}$ is the derivative of the NN w.r.t its weights which is usually readily available. Equations (13) and (14) provide the terminal condition and the backward dynamics for the co-states, respectively. Equation (15) provides the desired change in the weights of the neural network in order to increase the objective function $J_{x_0}(W)$. As a result, changing the weights of the NN in the direction δW using a small enough step size will increase the robustness function ρ_φ for a single trajectory starting from x_0 - assuming the soft min and max constant α is large enough. To emphasize the dependency of the desired change in the weights on the initial condition x_0, we will denote it as δW_{x_0}. As a result, in order to improve the objective function in Equation (10) which depends on all the samples in X_0^s rather than just one sample, the weights of the NN should be changed in the following direction:

$$\Delta W = \sum_{x_0 \in X_0^s} k_{x_0}(i) \delta W_{x_0} \tag{16}$$

where k_{x_0} is a vector of length $|X_0^s|$ whose i^{th} element is proportional to the effect of the robustness value $\tilde{\rho}_\varphi(\mathbf{x}(x_0, W))$ in the value of the overall objective function $J = \overline{\min}_{x_0 \in X_0^s} \tilde{\rho}_\varphi(\mathbf{x}(x_0, u^N))$. We write these robustness values collectively as r_{x_0}, as a result:

$$k_{x_0} = \frac{\partial \overline{\min}(r_{x_0})}{\partial x} \tag{17}$$

where $\frac{\partial \overline{\min}}{\partial x}$ is the derivative of the soft min function in Equation (4).

Note that changing W in the direction $\delta W_{\underline{x}_0}$, where $\underline{x}_0$ corresponds to the minimum robustness value in X_0^s, may result in a decrease in the worst case robustness value. The reason is that the change in the robustness value for the other samples in X_0^s are not considered and there could exist some $x_0 \in X_0^s, x_0 \neq \underline{x}_0$ which may worsen the minimum robustness value. Considering only the worst sample in the weight update law breaks the completeness of the method, and may cause oscillations between improving the policy for two $x_0 \in X_0^s$. However, using Equation (16) for updating the weights ensures an improvement in the worst-case behavior in X_0^s under mild assumptions, as it considers the effect of the weight perturbation on all the samples. In practice, in order to reduce the computational complexity, one can remove the terms for which $k_{x_0}(i) << max(k_{x_0})$ in Equation (16) and avoid calculating δW_{x_0} for them.

Proposition 5.1. *Assuming that the set of weights W is not a local minimizer of $\overline{min}_{x_0 \in X_0^s} \tilde{\rho}_\varphi$, there exists a small enough step size $\bar{h}_w > 0$ such that for all $0 < h_w \leq \bar{h}_w$:*

$$\overline{\min_{x_0 \in X_0^s}} \tilde{\rho}_\varphi(\mathbf{x}(x_0, W + h_w \Delta W) > \overline{\min_{x_0 \in X_0^s}} \tilde{\rho}_\varphi(\mathbf{x}(x_0, W))$$

Proof. This is a direct result of the design choice, since by design, ΔW is a descent direction of the cost function in Equation (10). □

5.3.1 Choosing the Step Size. Theoretically, the ideal step size for updating W is the upper-bound for all the $\bar{h}_w$ values that satisfy the condition in Problem 5.1. In practice, however, finding such an ideal step size is not possible. Choosing a proper step size h_w is important for convergence of the gradient-based policy search methods. Choosing very large step sizes can result in passing better level sets and local extremes, and choosing very small step sizes will cause insignificant and slow improvements. As a result, in gradient-based optimization, the step size is usually changed adaptively.

In this work, we choose step sizes proportional to the norm of the weight matrix $|W|$, $h_w = k|W|$. At the first iteration, we pick $k = k_0 << 1$ and change it adaptively later, as described in Algorithm 1. The value returned by Algorithm 1 in one iteration is used as the initial step size in the next iteration.

5.4 Training on X_0^s

In order to find a controller that maximizes the worst-case robustness value in the set of randomly selected initial conditions $X_0^s \subset X_0$, we use a global optimizer to explore the search space and find a set of weights W that optimize the average robustness value on X_0^S. Given the limitations of randomized search methods, the resulting weights from the previous stage are used to initialize the gradient-based local search described in Section 5.1 which is used later to improve the worst-case performance in X_0^s. This is described in Algorithm 2. If the purpose is to only satisfy the requirement, then the while loop can be exited as soon as the worst-case robustness value on X_0^s becomes positive. Note that the solution to the optimization problem of Equation (10) can be negative in general: there may exist $x_0 \in X_0^s$ for which the specification is not satisfied even with the best set of weights, either since φ is hard, or since the controller choice is not suitable.

5.5 Adversarial Guided Training

As described earlier, finding a set of weights which maximally satisfies the specification φ on the set X_0^s, does not guarantee the worst-case optimality for all $x_0 \in X_0$, nor does it guarantee the satisfaction of the formula in X_0. As a result, after training the weights on X_0^s, we use a falsification paradigm in order to find worst-case adversarial samples for which $\rho_\varphi < 0$ and $|\rho_\varphi|$ is maximized. Given the NN controller and its weights W, a falsification method, aims to solve the following problem [56, 57]:

$$\begin{aligned} &\underset{x_0 \in X_0}{\operatorname{argmin}} \ \rho_\varphi(\mathbf{x}(x_0, W)) \\ s.t \quad &\begin{cases} x_{t+1} = f(x_t, \mathcal{N}(g(x_t), W)) \\ \mathbf{x}(x_0, W) = x_0, x_1, \ldots, x_N \end{cases} \end{aligned} \tag{18}$$

If an adversarial sample x_0^a was found, we will add the sample to X_0^s, and use the Lagrange multipliers approach to update the NN weights W until no more progress can be achieved (Algorithm 2 without the initial global training). Once a set of weights W was found that maximally satisfies φ on the updated set X_0^s, we look for another adversarial sample (See Algorithm 3). If we were not

ALGORITHM 1: Step size adaptive update

Data: A dynamical system $\mathcal{S}$, W, ΔW, initial step size k_0, maximum number of iterations I_m, increase and decrease rates $\overline{\alpha}$ and $\underline{\alpha}$, and the specification φ

.**Result**: The final step size k, and the improved weights W^* if W is not a local optimizer, and a local optima flag 'LO' o.w

$dec\_flag \leftarrow$ false, $inc\_flag \leftarrow$ false, $LO \leftarrow$ false ;

$k = k_0$, $h_w = k|W|$, $W' = W + h_w \Delta W$, $J^* = \overline{\min_{x_0 \in X_0^s}} \tilde{\rho}_\varphi(\mathbf{x}(x_0, W))$;

for $i = 1, \ldots, I_m$ **do**

  $\Delta J = \overline{\min_{x_0 \in X_0^s}} \tilde{\rho}_\varphi(\mathbf{x}(x_0, W')) - J^*$;

  if $\Delta J > 0$ **then**

    $J^* = \overline{\min_{x_0 \in X_0^s}} \tilde{\rho}_\varphi(\mathbf{x}(x_0, W'))$;

    if *dec_flag* **then**

      return k, W^*, LO, and exit

    end

    $k \leftarrow \overline{\alpha} k$, $W^* \leftarrow W'$;

    $h_w = k|W|$, $W' = W + h_w \Delta W$;

    $inc\_flag \leftarrow$ true;

  else

    if *inc_flag* **then**

      return k, W^*, LO, and exit

    end

    $k \leftarrow \underline{\alpha} k$;

    $h_w = k|W|$, $W' = W + h_w \Delta W$;

    $dec\_flag \leftarrow$ true;

  end

end

$LO \leftarrow$ true;

return k, W, LO;

able to satisfy φ for x_0^a the algorithm returns failure. The reason for a failure can be one of the following:

- φ is not satisfiable on X_0.
- A state feedback controller is not enough for decision making. More information/features or controllers with memory might be required.
- The NN architecture is not suitable.
- There is no suitable policy in a small neighborhood around W.

Figure 1 shows the overall learning framework.

If the falsification method provides testing coverage guarantees (e.g, see [2]), once no more adversarial samples were found, φ is satisfied on X_0 almost surely. One can also try to verify the property φ over the set X_0 using a verification method for dynamical systems including neural networks [19, 21, 30].

6 EXPERIMENTAL RESULTS

The following experimental results were performed in MATLAB 2018b. The simulations were run on an Intel(R)Core(TM) i7-4790 CPU @3.6 GHz with 16 GB memory processor with Windows

ALGORITHM 2: Training on X_0^s

Data: A dynamical system $\mathcal{S}$, an STL specification φ, a randomly selected set of initial conditions X_0^s, an NN architecture $\mathcal{N}$, and the inputs to Algorithm 1.
Result: Set of parameters W that solve the problem in Equation (10)
$W \leftarrow$ Use a global optimizer to find the set of weights W that optimizes the average robustness on X_0^S (Section 5.2);
$LO \leftarrow$ false;
while $\neg LO$ **do**
  $\Delta W \leftarrow$ Use Equation (16) to find the ascent direction for W;
  $(W, LO) \leftarrow$ Use Algorithm 1 to change W in the ΔW direction
end
return W;

ALGORITHM 3: Retraining using adversarial samples

Data: A dynamical system $\mathcal{S}$, an STL Specification φ, and a randomly selected set of initial conditions X_0^s, an NN architecture $\mathcal{N}$ and its parameters W, and the inputs to Algorithm 1.
Result: Set of the optimal parameters W
while *true* **do**
  $x_0^a \leftarrow$ Use a falsification method to solve problem 18;
  if $\rho_\varphi(\mathbf{x}(x_0^a, W)) < 0$ **then**
    $X_0^s \leftarrow X_0^s \cup x_0^a$;
    $LO \leftarrow$ false;
    while $\neg LO$ **do**
      $\Delta W \leftarrow$ Use Equation (16) to find the desired direction;
      $(W, LO) \leftarrow$ Use Algorithm 1 to update W
    end
    if $\rho_\varphi(\mathbf{x}(x_0^a, W)) < 0$ **then**
      return failure;
      break
    end
  else
    return W;
    break
  end
end

10 Enterprise. We use the Matlab implementation of CMA-ES [27] for global optimization. The S-TaLiRo toolbox [4] is used to test the closed-loop system against the specification φ. The tool has different optimization algorithms. Specifically, in this work, we use the Simulated Annealing (SA) optimization method for finding the adversarial samples.

6.1 Vehicle Navigation

In this section, in order to evaluate the performance of the Lagrange multipliers approach, we use it without the help of the global search for updating the weights of an NN, which is used to guide a vehicle to accomplish a reach-avoid requirement formalized in STL. The vehicle navigates in a

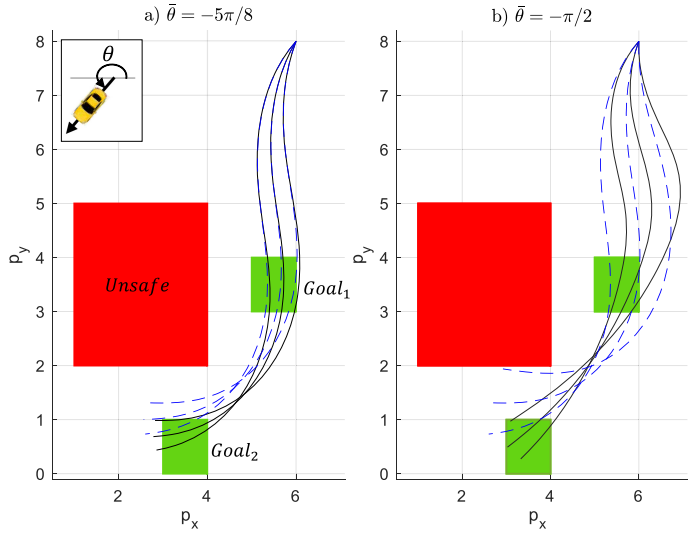

Fig. 4. Vehicle navigation: The vehicle should visit Goals 1 and 2 in this order while avoiding the unsafe set, starting from a set of different steering angles.

2D environment according to the following dynamics

$$\dot{x} = \begin{bmatrix} \dot{p_x} \\ \dot{p_y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v cos(\theta) \\ v sin(\theta) \\ \frac{v}{L} tan(\gamma) \end{bmatrix} \tag{19}$$

where (p_x, p_y) is the x-y position of the center of the vehicle and θ is the angle that the vehicle's heading has with the x-axis. The inputs v, γ are the forward driving speed and the steering angle of the front wheels. The inputs are limited to the sets $[0, 5]$ and $[-\pi/4, \pi/4]$, respectively. We assume that initially the car is at $(p_{x0}, p_{y0}) = (6, 8)$, but the heading angle can vary in the set $\theta_0 \in [-3\pi/4, \bar{\theta}]$. We choose $\bar{\theta} = -5\pi/8$ once and $\bar{\theta} = -\pi/2$ later. The system is simulated using a discretized step size $\Delta t = 0.05$ for 40 steps. The vehicle needs to visit goals 1 and then 2 in this order while avoiding an obstacle [36]. This can be expressed in STL using the following nested formula:

$$\varphi = \Diamond_{[1,40]} \Big(Goal_1 \wedge \Diamond Goal_2 \Big) \wedge \Box_{[1,40]} \neg Unsafe$$

where the *Unsafe*, $Goal_1$ and $Goal_2$ sets are 2D sets: $[1, 4] \times [2, 5]$, $[3, 4] \times [0, 1]$ and $[5, 6] \times [3, 4]$, respectively. These sets are shown in the left plot of Figure 4. We design an NN with 3 inputs (the inputs are the system states), 2 hidden layers each with 5 neurons, and 2 outputs that are applied to the system as the control inputs v, γ. The NN has tangent Sigmoid activation functions and the input constraints are enforced by using the scaled tangent Sigmoid functions on the outputs. Since the weight update law leads to local improvements in the weights, a good weight initialization is important. In order to initialize the weights, we picked the weights randomly between -1 and 1 for 20 times and kept the one that maximizes the minimum robustness for samples $\theta_0 = -3\pi/4, -5\pi/8, -\pi/2$.

The initial set was chosen as $X_0^s = \{-3\pi/4, \bar{\theta}, (\bar{\theta} - 3\pi/4)/2\}$. The soft min and max function's constant was picked as $\alpha = 100$ in the optimization problem. The initial step size is set to $k = 0.05$. For $\bar{\theta} = -5\pi/8$, system trajectories starting from initial conditions $x_0 \in X_0^s$ using the initial NN weights are shown with dashed blue lines in Figure 4(a). For $\bar{\theta} = -\pi/2$, they are shown with dashed

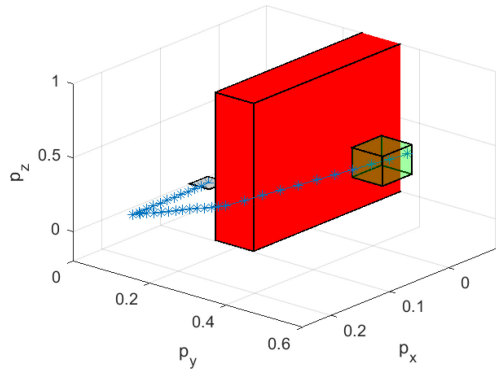

Fig. 5. Quadrotor needs to avoid the Unsafe set and reach the Goal set during a short interval of time. The mission is shown in blue.

blue lines as well, in Figure 4(b). When $\bar{\theta} = -5\pi/8$, a set of weights that could satisfy φ in the worst case, was found using only 8 iterations of the while loop in Algorithm 2. When the initial condition set grows larger, training the NN becomes harder as it needs to be able to generalize in a larger environment. In our experiment, when the initial condition set grows larger by choosing $\bar{\theta} = -\pi/2$, the objective function (worst-case robustness value) increases in the ascent direction only with much smaller step sizes h_w, and the satisfying set of weights is found after 576 iterations of the while loop in Algorithm 2. After training over the sample set X_0^s, using Algorithm 3, no adversarial examples were found in the the initial condition set for $\bar{\theta} = -5\pi/8$. However, for $\bar{\theta} = -\pi/2$, adversarial samples were found 2 times, and the weights were updated accordingly using Algorithm 3. Satisfying trajectories starting from $x_0 \in X_0^s$ are shown with solid black lines in Figure 4.

6.2 Quadrotor Mission

In this case study, we consider a 6 dimensional model of a quadrotor, as follows [30]:

$$\dot{x} = \begin{bmatrix} \dot{p_x} \\ \dot{p_y} \\ \dot{p_z} \\ \dot{v_x} \\ \dot{v_y} \\ \dot{v_z} \end{bmatrix} = \begin{bmatrix} v_x \\ v_y \\ v_z \\ g\tan(\theta) \\ -g\tan(\phi) \\ \tau - g \end{bmatrix} \tag{20}$$

where (p_x, p_y, p_z) and (v_x, v_y, v_z) are the quadrotor's position and velocity along x, y, z axis, θ, ϕ, τ are the control inputs (for pitch, roll and thrust), and $g = 9.81$ is the gravity. The input constraints are $\theta, \phi \in [-0.1, 0.1]$ and $\tau \in [7.81, 11.81]$. We assume that initially the quadrotor is still, $(v_{x0}, v_{y0}, v_{z0}) = (0, 0, 0)$, and at zero altitude $x_{z0} = 0$. The initial x-y position of the quadrotor can vary in $(p_{x0}, p_{y0}) \in [0.02, 0.05] \times [0, 0.05]$. The system is simulated using a discrete step size $\Delta t = 0.05$ for N steps, where N is the STL formula horizon. The quadrotor should visit a Goal set which is blocked by a tall wall (the '*Unsafe*' set) during a short amount of time while avoiding the wall. Formally, it needs to satisfy the following formula:

$$\varphi = \Box_{[1,35]} \neg \mathit{Unsafe} \wedge \Diamond_{[32,35]} \mathit{Goal}$$

where the time intervals correspond to the discrete time steps. The projection of the *Unsafe* and *Goal* sets into the quadrotor's position states are $[-\infty, 0.17] \times [0.2, 0.35] \times [0, 1.2]$ and $[0.05, 0.1] \times [0.5, 0.58] \times [0.5, 0.7]$, respectively. These sets are shown in Figure 5 with red and green boxes respectively. The set of initial positions is shown in grey.

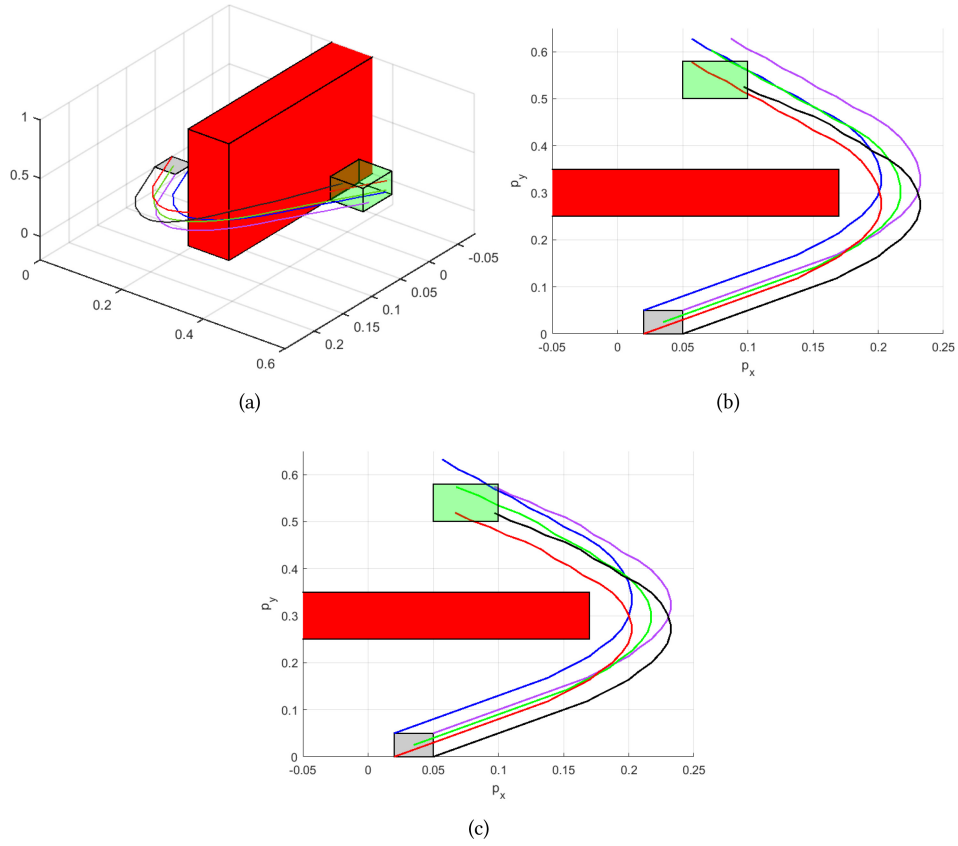

Fig. 6. Sample trajectories in X_0^s, (a): after global training, (b): projection of (a) into x-y plane (c): Trajectories after improving the NN weights using back propagation.

We pick a NN with 6 inputs (the states of the quadrotor), 3 outputs (the control inputs of the NN), 3 hidden layers with 6,10, and 6 neurons. The NN has tangent Sigmoid activation functions and a scaled tangent Sigmoid in the output to enforce the control input constraints. Initially, we pick the corners and the center of the initial set to create X_0^s. In the initial training phase, the weights of the NN are selected in order to minimize a cost function which depends on the average distance of the quadrotor's trajectories from a motion plan that satisfies the specification. This distance is measured using a simplified dynamic time warping metric [40]. The motion plan is a simple piece-wise constant trajectory that the quadrotor is not able to perfectly follow given its dynamics. This plan is shown in Figure 5. A final cost based on whether the trajectory reaches the goal set on time or not is added to the cost function. One hundred iterations of the CMA-ES with a population size of 200 were used for initial training of the neural network. This phase took about 3×10^4 seconds (about 8 hours) on our machine[4]. The trajectories of X_0^s after the initial training phase are shown in Figure 6(a,b). As it's clear from the figure, one of the trajectories (in purple) does not satisfy φ. As a result, we improve the set of NN weights W, using the Lagrange multiplier weight update law (see Algorithm 2). We pick $\alpha = 1000$ and an initial step size of $k = 0.1$. The trajectories after this step are shown in Figure 6(c). In the adversarial training phase (see Algorithm 3), 3 adversarial examples

[4]We did not use GPUs to reduce the training time.

Table 1. Comparison of Training Using the Proposed Approach (LM) and CMA-ES

	LM	CMA-ES
training time (sec.)	$3 \times 10^4 + 10^3$	8.6×10^4
num. of falsifications found	3	6

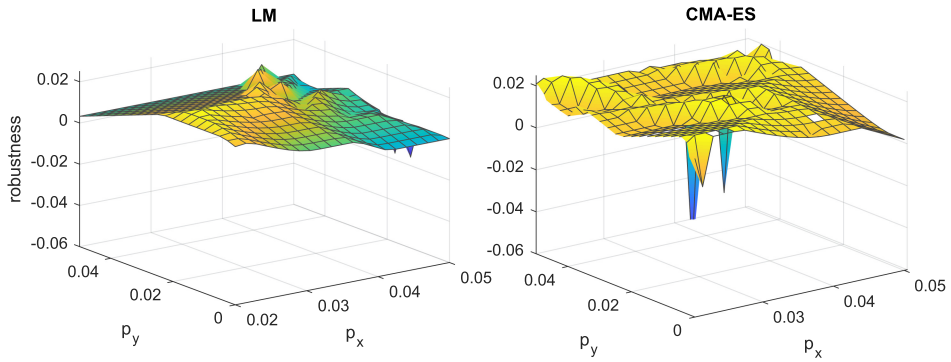

Fig. 7. Robustness surfaces using the controllers trained with LM and CMA-ES approach.

were found using S-TaLiRo's SA optimization, they were added to the samples set X_0^s, and the NN weights were adjusted using our weight update law accordingly until no further adversarial samples were found. The NN weight update using the Lagrange multipliers approach - on the initial set plus its integration with the adversarial samples - took about 1000 seconds. An interesting observation is that while the NN was not designed to output discretized values, after the training phase, the weights were tuned such that the NN outputs were almost always equal to the minimum or maximum allowed values of the control inputs. In other words, the output layer activation function was always saturated. This is interesting since as described in [30], the optimal policy is a bang-bang strategy.

The results from the above Lagrange Multipliers (LM) approach were compared with the results using a pure CMA-ES method as described in the following:

(1) The initial set X_0^s is selected as before.
(2) CMA-ES is used to maximize the minimum robustness value on X_0^s with a maximum of 100 iterations and it returns whenever the set of weights satisfies φ on X_0^s in the worst case.
(3) A falsification approach is used. If an adversarial sample is found, it will be added to X_0^s, and the algorithm goes back to step (2), otherwise, the algorithm returns.

Comparison results on the training time and the number of adversarial samples found before the algorithm returns can be found in Table 1. The training time for our approach is less than half the training time for the CMA-ES approach (training using CMA-ES approach took about a day). In order to compare the quality of the resulting controller using these approaches, we finely grid the set of initial conditions and evaluated the corresponding robustness values. Resulting surfaces for the two approaches are shown in Figure 7 which shows a better worst-case performance using the LM approach. Note that, in both cases, a few samples with negative robustness values were found since the falsification approach that we used did not have coverage guarantees. These negative robustness values are much smaller in amplitude when using the controller designed by the LM

approach while the training time was also much less. Based on the Goal set dimension, the upper bound to the robustness value is $\frac{0.1-0.05}{2} = 0.025$ which requires that the trajectory visits the center of the Goal set within the specified time interval.

7 CONCLUSIONS

Given a system model, we proposed a systematic approach to training neural network controllers that satisfy system properties given in STL. The loss functions for training the NNs are inspired by the robustness of the STL formula which is defined over temporal sequences corresponding to the closed-loop system response. Since usually the minimum length of the temporal sequences required for calculating the robustness value is large, training using automatic differentiation methods for backpropagation faces the problem of vanishing and exploding gradients, as a result, we provided a formulation for gradient-based training which solves this issue. Furthermore, after training the NN with random samples, we iteratively search for adversarial samples to the STL property, add them to the sample set and retrain the NN. We demonstrate our approach on a 6 dimensional model of a quadrotor that needs to accomplish a mission specified in an STL formula.

While we used state feedback neural network controllers, a controller might require more information than the current states in order to satisfy general STL properties. Specifying features that are required for satisfying different STL properties will be investigated in future work. NN controllers with memory for satisfying STL formulas will also be studied.

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] Arvind Adimoolam, Thao Dang, Alexandre Donzé, James Kapinski, and Xiaoqing Jin. 2017. Classification and coverage-based falsification for embedded control systems. In *International Conference on Computer Aided Verification*. Springer, 483–503.

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

[4] 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.

[5] 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.

[6] Dimitri P. Bertsekas. 2014. *Constrained Optimization and Lagrange Multiplier Methods*. Academic press.

[7] Dimitri P. Bertsekas. 2019. *Reinforcement learning and optimal control*. Athena Scientific.

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

[9] Kyunghoon Cho and Songhwai Oh. 2018. Learning-based model predictive control under signal temporal logic specifications. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 7322–7329.

[10] Arthur Claviere, Souradeep Dutta, and Sriram Sankaranarayanan. 2019. Trajectory tracking control for robotic vehicles using counterexample guided training of neural networks. In *Proceedings of the International Conference on Automated Planning and Scheduling*, Vol. 29. 680–688.

[11] Konstantinos Dalamagkidis, Kimon P Valavanis, and Les A. Piegl. 2010. Nonlinear model predictive control with neural network optimization for autonomous autorotation of small unmanned helicopters. *IEEE Transactions on Control Systems Technology* 19, 4 (2010), 818–831.

[12] M. Dehghani, M. Ahmadi, A. Khayatian, M. Eghtesad, and M. Farid. 2008. Neural network solution for forward kinematics problem of HEXA parallel robot. In *2008 American Control Conference*. IEEE, 4214–4219.

[13] Marc Deisenroth and Carl E. Rasmussen. 2011. PILCO: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*. 465–472.

[14] Alexandre Donzé. 2010. Breach, a toolbox for verification and parameter synthesis of hybrid systems. In *International Conference on Computer Aided Verification*. Springer, 167–170.

[15] Alexandre Donzé and Oded Maler. 2010. Robust satisfaction of temporal logic over real-valued signals. In *International Conference on Formal Modeling and Analysis of Timed Systems*. Springer, 92–106.
[16] 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.
[17] 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).
[18] Tommaso Dreossi, Somesh Jha, and Sanjit A. Seshia. 2018. Semantic adversarial deep learning. In *International Conference on Computer Aided Verification*. Springer, 3–26.
[19] Souradeep Dutta, Xin Chen, and Sriram Sankaranarayanan. 2019. Reachability analysis for neural feedback systems using regressive polynomial rule inference. In *International Conference on Hybrid Systems: Computation and Control (HSCC)*.
[20] Souradeep Dutta, Susmit Jha, Sriram Sanakaranarayanan, and Ashish Tiwari. 2017. Output range analysis for deep neural networks. *arXiv preprint arXiv:1709.09130* (2017).
[21] 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.
[22] Georgios E. Fainekos and George J. Pappas. 2009. Robustness of temporal logic specifications for continuous-time signals. *Theoretical Computer Science* 410, 42 (2009), 4262–4291.
[23] Goran Frehse, Colas Le Guernic, Alexandre Donzé, Scott Cotton, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang, and Oded Maler. 2011. SpaceEx: Scalable verification of hybrid systems. In *International Conference on Computer Aided Verification*. Springer, 379–395.
[24] Qitong Gao, Davood Hajinezhad, Yan Zhang, Yiannis Kantaros, and Michael M. Zavlanos. 2019. Reduced variance deep reinforcement learning with temporal logic specifications. (2019).
[25] Martin T Hagan, Howard B Demuth, and Orlando De Jesús. 2002. An introduction to the use of neural networks in control systems. *International Journal of Robust and Nonlinear Control: IFAC-Affiliated Journal* 12, 11 (2002), 959–985.
[26] Nikolaus Hansen and Stefan Kern. 2004. Evaluating the CMA evolution strategy on multimodal test functions. In *International Conference on Parallel Problem Solving from Nature*. Springer, 282–291.
[27] Nikolaus Hansen and Andreas Ostermeier. 2001. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation* 9, 2 (2001), 159–195.
[28] Michael Hertneck, Johannes Köhler, Sebastian Trimpe, and Frank Allgöwer. 2018. Learning an approximate model predictive controller with guarantees. *IEEE Control Systems Letters* 2, 3 (2018), 543–548.
[29] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. 1989. Multilayer feedforward networks are universal approximators. *Neural Networks* 2, 5 (1989), 359–366.
[30] Radoslav Ivanov, James Weimer, Rajeev Alur, George J. Pappas, and Insup Lee. 2019. Verisig: Verifying safety properties of hybrid systems with neural network controllers. (2019), 169–178.
[31] Kyle D. Julian and Mykel J. Kochenderfer. 2017. Neural network guidance for UAVs. In *AIAA Guidance, Navigation, and Control Conference*. 1743.
[32] Kyle D. Julian, Jessica Lopez, Jeffrey S. Brush, Michael P. Owen, and Mykel J. Kochenderfer. 2016. Policy compression for aircraft collision avoidance systems. In *2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)*. IEEE, 1–10.
[33] Hassan K. Khalil and Jessy W. Grizzle. 2002. *Nonlinear systems*. Vol. 3. Prentice hall Upper Saddle River, NJ.
[34] Ron Koymans. 1990. Specifying real-time properties with metric temporal logic. *Real-time Systems* 2, 4 (1990), 255–299.
[35] Sergey Levine and Pieter Abbeel. 2014. Learning neural network policies with guided policy search under unknown dynamics. In *Advances in Neural Information Processing Systems*. 1071–1079.
[36] Xiao Li, Yao Ma, and Calin Belta. 2018. A policy search method for temporal logic specified reinforcement learning tasks. In *2018 Annual American Control Conference (ACC)*. IEEE, 240–245.
[37] Oded Maler and Dejan Nickovic. 2004. Monitoring temporal properties of continuous signals. In *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*. Springer, 152–166.
[38] 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.
[39] William H Montgomery and Sergey Levine. 2016. Guided policy search via approximate mirror descent. In *Advances in Neural Information Processing Systems*. 4008–4016.
[40] Meinard Müller. 2007. Dynamic time warping. *Information Retrieval for Music and Motion* (2007), 69–84.
[41] K. Muralitharan, Rathinasamy Sakthivel, and R. Vishnuvarthan. 2018. Neural network based optimization approach for energy demand prediction in smart grid. *Neurocomputing* 273 (2018), 199–208.
[42] 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.

[43] Yash Vardhan Pant, Houssam Abbas, Rhudii A. Quaye, and Rahul Mangharam. 2018. Fly-by-logic: Control of multi-drone fleets with temporal logic objectives. In *Proceedings of the 9th ACM/IEEE International Conference on Cyber-Physical Systems*. IEEE Press, 186–197.

[44] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2012. Understanding the exploding gradient problem. *CoRR, abs/1211.5063* 2 (2012).

[45] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2013. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*. 1310–1318.

[46] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. 2017. Deepxplore: Automated whitebox testing of deep learning systems. In *proceedings of the 26th Symposium on Operating Systems Principles*. ACM, 1–18.

[47] Vasumathi Raman, Alexandre Donzé, Mehdi Maasoumy, Richard M. Murray, Alberto Sangiovanni-Vincentelli, and Sanjit A. Seshia. 2014. Model predictive control with signal temporal logic specifications. In *53rd IEEE Conference on Decision and Control*. IEEE, 81–87.

[48] Vasumathi Raman, Alexandre Donzé, Dorsa Sadigh, Richard M. Murray, and Sanjit A. Seshia. 2015. Reactive synthesis from signal temporal logic specifications. In *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control*. ACM, 239–248.

[49] Vicenc Rubies Royo, David Fridovich-Keil, Sylvia Herbert, and Claire J. Tomlin. 2018. Classification-based approximate reachability with guarantees applied to safe trajectory tracking. *arXiv preprint arXiv:1803.03237* (2018).

[50] Johann Schumann and Yan Liu. 2010. *Applications of neural networks in high assurance systems*. SCI, Vol. 268. Springer.

[51] Cumhur Erkan Tuncali, Georgios Fainekos, Hisahiro Ito, and James Kapinski. 2018. Simulation-based adversarial test generation for autonomous vehicles with machine learning components. In *2018 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 1555–1562.

[52] Cristian-Ioan Vasile, Vasumathi Raman, and Sertac Karaman. 2017. Sampling-based synthesis of maximally-satisfying controllers for temporal logic specifications. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 3840–3847.

[53] Marcell J. Vazquez-Chanlatte, Shromona Ghosh, Vasumathi Raman, Alberto Sangiovanni-Vincentelli, and Sanjit A. Seshia. 2018. Generating dominant strategies for continuous two-player zero-sum games. *IFAC-PapersOnLine* 51, 16 (2018), 7–12.

[54] Grady Williams, Nolan Wagener, Brian Goldfain, Paul Drews, James M. Rehg, Byron Boots, and Evangelos A. Theodorou. 2017. Information theoretic MPC for model-based reinforcement learning. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 1714–1721.

[55] Weiming Xiang, Patrick Musau, Ayana A. Wild, Diego Manzanas Lopez, Nathaniel Hamilton, Xiaodong Yang, Joel Rosenfeld, and Taylor T. Johnson. 2018. Verification for machine learning, autonomy, and neural networks survey. *arXiv preprint arXiv:1810.01989* (2018).

[56] 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.

[57] Shakiba Yaghoubi and Georgios Fainekos. 2019. Gray-box adversarial testing for control systems with machine learning components. In *Proceedings of the 22Nd ACM International Conference on Hybrid Systems: Computation and Control (HSCC'19)*. ACM, New York, NY, USA, 179–184. DOI: https://doi.org/10.1145/3302504.3311814

[58] Tianhao Zhang, Gregory Kahn, Sergey Levine, and Pieter Abbeel. 2016. Learning deep control policies for autonomous aerial vehicles with mpc-guided policy search. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 528–535.

[59] Siqi Zhou, Mohamed K. Helwa, and Angela P. Schoellig. 2017. Design of deep neural networks as add-on blocks for improving impromptu trajectory tracking. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*. IEEE, 5201–5207.

Received April 2019; revised June 2019; accepted July 2019