Available online at www.sciencedirect.com

ScienceDirect

IFAC Papers Online CONFERENCE PAPER ARCHIVE

IFAC PapersOnLine 58-11 (2024) 183–188

LB4TL: A Smooth Semantics for Temporal Logic to Train Neural Feedback Controllers

Navid Hashemi * **Samuel Williams** * **Bardh Hoxha** **
Danil Prokhorov ** **Georgios Fainekos** **
Jyotirmoy Deshmukh *

* *University of Southern California, CA 90007 USA (e-mail: navidhas, samwilliams, jdeshmuk @usc.edu).*
** *Toyota NA R&D, Ann Arbor, MI 48105 USA (e-mail: {first_name.last_name}@toyota.com)*

Abstract:
This paper presents a framework for training neural network (NN)-based feedback controllers for autonomous agents with deterministic nonlinear dynamics to satisfy task objectives and safety constraints expressed in discrete-time Signal Temporal Logic (DT-STL). Control synthesis that uses the robustness semantics of DT-STL poses challenges due to its non-convexity, non-differentiability, and recursive definition, in particular when it is used to train NN-based controllers. We introduce a smooth neuro-symbolic computation graph to encode DT-STL robustness to represent a smooth approximation of the robustness, enabling the use of powerful stochastic gradient descent and backpropagation-based optimization for training. Our approximation guarantees that it lower bounds the robustness value of a given DT-STL formula, and shows orders of magnitude improvement over existing smooth approximations when applied to control synthesis. We demonstrate our approach on planning to satisfy complex spatio-temporal and sequential tasks, and show scalability with formula complexity.

Keywords: Neural Networks, Learning, STL, Smoothing, Nonlinear Control.

1. INTRODUCTION

Learning-based approaches to synthesize controllers for highly nonlinear dynamical systems are used across domains: from autonomous vehicles to robots. Popular ways to train NN-based controllers include deep reinforcement learning (RL)(Li et al., 2017; Chua et al., 2018; Mnih et al., 2013) and deep imitation learning (Fang et al., 2019). Techniques to synthesize neural controllers (including deep RL methods) largely focus on optimizing user-defined rewards or costs, but do not directly address specific spatio-temporal task objectives. For example, consider the objective that the system must reach region R_1 before reaching region R_2, while avoiding an obstacle region. spatio-temporal objectives can be easily expressed in the formalism of Signal Temporal Logic (STL) (Maler and Nickovic, 2004). The discrete-time variant of STL has found applications in robotics and control synthesis (Raman et al., 2014; Pant et al., 2018) through quantitative semantics (Donzé and Maler, 2010; Fainekos and Pappas, 2006).

The use of DT-STL-based objectives has seen considerable recent interest in data-driven methods for training controllers for dynamical systems that can be described by (stochastic) difference equations. This literature brings together two separate threads: (1) smooth approximations to the robustness degree of DT-STL specifications (Gilpin et al., 2020; Pant et al., 2017) enabling the use of STL robustness in gradient-based learning of control policies, and (2) efficient representation of the robustness computation allowing its use in training neural controllers using backpropagation (Yaghoubi and Fainekos, 2019; Leung et al., 2019, 2021; Hashemi et al., 2023a,b). However, with few exceptions, *existing literature largely focuses on open-loop control policies*. Closed-loop feedback control has the obvious advantage that it is robust to perturbations in the system state; on the other hand, the control synthesis problem is significantly harder.

In this paper, we revisit our prior work (Hashemi et al., 2023a) that proposed a ReLU-based neural network encoding (called STL2NN) of DT-STL formulas to exactly compute the robustness value. We show how we can significantly extend this computation graph to obtain *smooth, guaranteed underapproximations* of the DT-STL robustness value. We use backpropagation-based methods that treat the one-step environment dynamics and the neural feedback controller as a recurrent unit that is then unrolled as many times as required by the temporal horizon of the DT-STL specification φ. We make the following contributions:

(1) We propose smooth versions of computation graphs representing the robustness degree computation of a DT-STL specification over the trajectory of a dynamical system. Our computation graph guarantees that it lower bounds the robustness value with a tunable degree of approximation.
(2) We develop a backpropagation framework which leverages the new differentiable structure, and we show how we can handle DT-STL specifications.

10.1016/j.ifacol.2024.07.445

Related Work. The use of temporal logic specifications for controller synthesis is a well-studied problem. Recent years have also seen growing interest in data-driven techniques (Li et al., 2018) for control synthesis. In (Balakrishnan and Deshmukh, 2019), the authors replace handcrafted reward functions with the STL robustness within single-agent or multi-agent deep RL frameworks. The overall approach of this paper is the closest to the work in (Yaghoubi and Fainekos, 2019; Leung et al., 2019, 2021; Hashemi et al., 2023a,b), where DT-STL robustness is used in conjunction with backpropagation to train neural network-based controllers. A significant difference between some of the previous approaches (Leung et al., 2019, 2021) and this work is the smooth semantics we introduced and utilized for training the optimal policy.

2. PRELIMINARIES

We denote the set, $\{1, 2, \cdots, n\}$ with $[n]$. A feed forward neural network (NN) with ℓ hidden layers is denoted by the array $[n_0, n_1, \cdots, n_{\ell+1}]$, where n_0 denotes the number of inputs, $n_{\ell+1}$ is the number of outputs and for $i \in [\ell]$, n_i denotes the width of i^{th} hidden layer.

NN Feedback Control Systems (NNFCS). Let $\mathbf{s}$ and $\mathbf{a}$ denote the state and action variables that take values from compact sets $\mathcal{S} \subseteq \mathbb{R}^n$ and $\mathcal{A} \subseteq \mathbb{R}^m$, respectively. We use $\mathbf{s}_k$ (resp. $\mathbf{a}_k$) to denote the value of the state (resp. action) at time k. We define a neural network controlled system (NNFCS) as a recurrent difference equation

$$\mathbf{s}_{k+1} = \mathbf{f}(\mathbf{s}_k, \mathbf{a}_k), \tag{1}$$

where $\mathbf{a}_k = \pi_\theta(\mathbf{s}_k, k)$ is the control policy. We assume that the control policy is a parameterized function π_θ, where θ is a vector of parameters that takes values in Θ. Later in the paper, we instantiate the specific parametric form using a neural network for the controller. That is, given a fixed vector of parameters θ, the parametric control policy π_θ returns an action $\mathbf{a}_k$ as a function of the current state $\mathbf{s}_k \in \mathcal{S}$ and time $k \in \mathbb{Z}^{\geq 0}$, i.e., $\mathbf{a}_k = \pi_\theta(\mathbf{s}_k, k)$.

Closed-loop Model Trajectory. For a discrete-time NNFCS, for a set of designated initial states $\mathcal{I} \subseteq \mathcal{S}$ under a pre-defined feedback policy π_θ, (1) represents an autonomous discrete-time dynamical system. For a given initial state $\mathbf{s}_0 \in \mathcal{I}$, a system trajectory $\sigma^\theta_{\mathbf{s}_0}$ is a function mapping time instants in $[0, K]$ to $\mathcal{S}$, where $\sigma^\theta_{\mathbf{s}_0}(0) = \mathbf{s}_0$, and for all $k \in [0, K-1]$, $\sigma^\theta_{\mathbf{s}_0}(k+1) = \mathbf{f}(\mathbf{s}_k, \pi_\theta(\mathbf{s}_k, k))$ [1]. The computation graph for this trajectory is a recurrent structure. In this paper, we provide algorithms to learn a policy $\pi_{\theta^\star}$ that maximizes the degree to which certain task objectives and safety constraints are satisfied.

Task Objectives and Safety Constraints. We assume that task objectives and safety constraints are specified using the syntax of Signal Temporal Logic (STL)(Maler and Nickovic, 2004) in positive normal form

$$\varphi \;=\; h(\mathbf{s}) \bowtie 0 \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \mathbf{U}_I \varphi_2 \mid \varphi_1 \mathbf{R}_I \varphi_2 \tag{2}$$

where $\mathbf{U}_I$ and $\mathbf{R}_I$ are the timed until and release operators, $\bowtie \in \{\leq, <, >, \geq\}$, and h is a function from $\mathcal{S}$ to $\mathbb{R}$. In this work, since we use discrete-time semantics for STL (referred to as DT-STL), the time interval I is a bounded interval of integers, i.e., $I = [a, b] \subseteq [0, K]$. The timed eventually ($\mathbf{F}_I$) and always ($\mathbf{G}_I$) operators can be syntactically defined through until and release. That is, $\mathbf{F}_I\varphi \equiv \top \mathbf{U}_I \varphi$ and $\mathbf{G}_I\varphi \equiv \bot \mathbf{R}_I \varphi$ where $\top$ and $\bot$ represent true and false. We briefly recall the formal semantics of DT-STL over discrete-time trajectories previously presented in (Fainekos and Pappas, 2006). Note that without timestamps, DT-STL is just a regular language; nevertheless, in this work, we use robust semantics which are not defined over automata or regular expressions.

[1] If obvious from the context, we drop θ in the notation $\sigma^\theta_{\mathbf{s}_0}$.

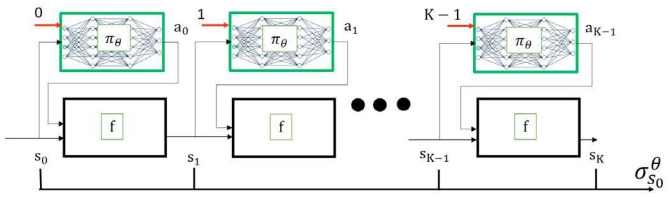

Fig. 1. Shows an illustration for computation graph of trajectory states in our control feedback system. The controller is a feed forward neural network that receives the time and state and returns a decision. The parameters of this controller will be trained to satisfy a temporal property, formulated in STL framework.

Boolean Semantics and Formula Horizon. We denote φ being true at time k in trajectory $\sigma^\theta_{\mathbf{s}_0}$ by $\sigma^\theta_{\mathbf{s}_0}, k \models \varphi$. We say that $\sigma^\theta_{\mathbf{s}_0}, k \models h(\mathbf{s}) \bowtie 0$ iff $h(\sigma^\theta_{\mathbf{s}_0}(k)) \bowtie 0$. The semantics of the Boolean operations ($\wedge$, $\vee$) follow standard logical semantics of conjunctions and disjunctions, respectively. For temporal operators, we say $\sigma^\theta_{\mathbf{s}_0}, k \models \varphi_1 \mathbf{U}_I \varphi_2$ is true if there is a time k', s.t. $k' - k \in I$ where φ_2 is true and for all times $k'' \in [k, k')$, φ_1 is true. Similarly, $\sigma^\theta_{\mathbf{s}_0}, k \models \varphi_1 \mathbf{R}_I \varphi_2$ is true if for all times k' with $k' - k \in I$, φ_2 is true, or there exists some time $k'' \in [k, k')$ such that φ_1 was true. The temporal horizon of a DT-STL formula defines the number of time-steps required in a trajectory to evaluate the formula, $\sigma^\theta_{\mathbf{s}_0}, 0 \models \varphi$ (see (Maler and Nickovic, 2004)).

Quantitative Semantics (Robustness value) of DT-STL. Quantitative semantics of DT-STL roughly define a signed distance of a given trajectory from the set of trajectories satisfying or violating the given DT-STL formula. There are many alternative semantics proposed in the literature (Donzé and Maler, 2010; Fainekos and Pappas, 2006); in this paper, we focus on the semantics from (Donzé and Maler, 2010) that are shown below. The robustness value $\rho(\varphi, \sigma^\theta_{\mathbf{s}_0}, k)$ of a DT-STL formula φ over a trajectory $\sigma^\theta_{\mathbf{s}_0}$ at time k is defined recursively as follows [2].

φ	$\rho(\varphi, k)$
$h(\mathbf{s}_k) \geq 0$	$h(\mathbf{s}_k)$
$\varphi_1 \wedge \varphi_2$ (resp. $\vee$)	$\min(\rho(\varphi_1, k), \rho(\varphi_2, k))$ (resp. max)
$\varphi_1 \mathbf{U}_{[a,b]} \varphi_2$	$\max\limits_{k' \in [k+a, k+b]} \left(\min \left(\rho(\varphi_2, k'), \min\limits_{k'' \in [k, k')} \rho(\varphi_1, k'') \right) \right)$
$\varphi_1 \mathbf{R}_{[a,b]} \varphi_2$	$\min\limits_{k' \in [k+a, k+b]} \left(\max \left(\rho(\varphi_2, k'), \max\limits_{k'' \in [k, k')} \rho(\varphi_1, k'') \right) \right)$

(3)

[2] For brevity, we omit the trajectory from the notation, as it is obvious from the context.

We note that if $\rho(\varphi, k) > 0$ the DT-STL formula φ is satisfied at time k, and we say that the formula φ is satisfied by a trajectory if $\rho(\varphi, 0) > 0$.

Discrete Time STL Robustness as a ReLU *NN.* The quantitative semantics of DT-STL is based on min/max operators. Therefore, the robust interpretation of a DT-STL specification is difficult to be used in gradient-based method for learning. However, min / max operators can be expressed using ReLU functions as follows (see (Hashemi et al., 2023a)):

$$\begin{aligned} \min(a_1, a_2) &= a_1 - \mathsf{ReLU}(a_1 - a_2), \\ \max(a_1, a_2) &= a_2 + \mathsf{ReLU}(a_1 - a_2). \end{aligned} \tag{4}$$

This allows the computation graph representing the robustness of a DT-STL formula w.r.t. a given trajectory to be expressed using repeated application of the ReLU function (with due diligence in balancing min, max computations over several arguments into a tree of at most logarithmic height in the number of operands). We call this ReLU-based computation graph as STL2NN.

3. TRAINING NN CONTROL POLICIES

Problem Definition. We wish to learn a neural network (NN) control policy π_θ (or equivalently the parameter values θ), s.t. for any initial state $\mathbf{s}_0 \in \mathcal{I}$, using the control policy π_θ, the trajectory obtained, i.e., $\sigma^\theta_{\mathbf{s}_0}$, satisfies a given DT-STL formula φ.

Our solution strategy is to treat each time-step of the given dynamical equation in (1) as a recurrent unit. We then sequentially compose or unroll as many units as required by the horizon of the DT-STL specification. The unrolled structure essentially represents the symbolic trajectory, where each recurrent unit shares the NN parameters of the controller (see Figure 1). By composing this structure with the neural network representing the given DT-STL specification φ; for instance, the STL2NN computation graph introduced in the previous section, then we have an NN that maps the initial state of the system (1) to the robustness degree of φ. Thus, training the resulting NN to guarantee that its output is positive over a sufficient sample of initial conditions is the first step towards computing the policy π_θ. The second step is to verify that the computed π_θ guarantees the satisfaction of the specification φ for any initial condition $\mathbf{s}_0 \in \mathcal{I}$. This can be achieved by using reachability computation methods (Dutta et al., 2019) or conformal prediction (Vovk et al., 2005) based probabilistic guarantees (see Sec. 5.1).

4. SMOOTH UNDER-APPROXIMATION FOR STL2NN

The existing smooth semantics for gradient computation (Gilpin et al., 2020; Pant et al., 2017; Leung et al., 2019) perform backward computation on a computation graph that is generated based on dynamic programming. Although these computation graphs are efficient for forward computation, they may face computational difficulty for backward computation over the robustness when the specification is highly complex. However, STL2NN, directly utilizes the STL tree (Donzé and Maler, 2010) to generate a feedforward ReLU neural network as a computation graph whose depth grows logarithmically with the complexity of DT-STL specification. This makes back-propagation more feasible for complex specifications. On the other hand, the way it formulates the robustness (Feedforward NN) facilitates the back-propagation process, by enabling vectorized computation of the gradient. However, considering that STL2NN is exactly identical to the non-smooth robustness introduced in Eq. (3), using smooth approximations, as suggested in previous studies (Gilpin et al., 2020; Pant et al., 2017), has proven to improve the efficiency, particularly in gradient-based techniques. Therefore, we approximate STL2NN with a smooth function. It is also preferable that this smooth approximation also acts as a guaranteed lower bound for the robustness. Ensuring its positivity guarantees that the real robustness is also positive. Thus, we approximate STL2NN with a smooth under-approximator, and we call this smooth function as LB4TL. We also propose a thorough and clear comparison between the performance of LB4TL and the previous smooth semantics, available in the literature. To generate LB4TL, we firstly replace ReLU activations in the min() operation Eq. (4) with the softplus activation function defined as:

$$\mathsf{softplus}(a_1 - a_2\ ; b) = \frac{1}{b}\log\left(1 + e^{b(a_1 - a_2)}\right),\ b > 0.$$

Similarly, we replace the ReLU activation functions contributing in max() operation Eq. (4) with the swish activation function:

$$\mathsf{swish}(a_1 - a_2\ ; b) = \frac{a_1 - a_2}{1 + e^{-b(a_1 - a_2)}},\ b > 0.$$

Next, we show that LB4TL is a guaranteed lower-bound for STL2NN. To that end, we start with the following proposition,

Proposition 1. For any two real numbers $x, y \in \mathbb{R}$:
$y + \mathsf{swish}(x - y) \leq \max(x, y)$, $x - \mathsf{softplus}(x - y) \leq \min(x, y)$.

Proof. We know for all $x, y \in \mathbb{R}$, $\max(x, y) = y + \mathsf{ReLU}(x - y)$ and $\min(x, y) = x - \mathsf{ReLU}(x - y)$. We also know, for all $z \in \mathbb{R}$, $\mathsf{swish}(z) < \mathsf{ReLU}(z)$ and $\mathsf{softplus}(z) > \mathsf{ReLU}(z)$ (Ramachandran et al., 2017).

The result of the Proposition. 1, can be utilized to propose the following result,

Proposition 2. Assume φ_1 and φ_2 are two different DT-STL formulas, and assume $L_1 = \mathsf{LB4TL}(\sigma^\theta_{\mathbf{s}_0}, \varphi_1,\ 0\ ; b) \leq R_1 = \rho(\sigma^\theta_{\mathbf{s}_0}, \varphi_1, 0)$ and $L_2 = \mathsf{LB4TL}(\sigma^\theta_{\mathbf{s}_0}, \varphi_2,\ 0\ ; b) \leq R_2 = \rho(\sigma^\theta_{\mathbf{s}_0}, \varphi_2, 0)$. Then we can conclude,

$$\begin{aligned} &\mathsf{LB4TL}(\sigma^\theta_{\mathbf{s}_0}, \varphi_1 \vee \varphi_2,\ 0\ ; b) \leq \rho(\sigma^\theta_{\mathbf{s}_0}, \varphi_1 \vee \varphi_2, 0), \\ &\mathsf{LB4TL}(\sigma^\theta_{\mathbf{s}_0}, \varphi_1 \wedge \varphi_2,\ 0\ ; b) \leq \rho(\sigma^\theta_{\mathbf{s}_0}, \varphi_1 \wedge \varphi_2, 0). \end{aligned}$$

Proof. Based on Proposition. 1, we know $\mathsf{LB4TL}(\sigma^\theta_{\mathbf{s}_0}, \varphi_1 \vee \varphi_2,\ 0\ ; b) = L_2 + \mathsf{swish}(L_1 - L_2\ ; b) \leq \max(L_1, L_2)$ and $\rho(\sigma^\theta_{\mathbf{s}_0}, \varphi_1 \vee \varphi_2, 0) = \max(R_1, R_2)$ We also know $L_1 \leq R_1$, and $L_2 \leq R_2$ which implies $\max(L_1, L_2) \leq \max(R_1, R_2)$. Therefore, we can conclude $\mathsf{LB4TL}(\sigma^\theta_{\mathbf{s}_0}, \varphi_1 \vee \varphi_2,\ 0\ ; b) \leq \rho(\sigma^\theta_{\mathbf{s}_0}, \varphi_1 \vee \varphi_2, 0)$. Likewise, from Proposition. 1, we know $\mathsf{LB4TL}(\sigma^\theta_{\mathbf{s}_0}, \varphi_1 \wedge \varphi_2,\ 0; b) = L_1 - \mathsf{softplus}(L_1 - L_2\ ; b) \leq \min(L_1, L_2)$ and $\rho(\sigma^\theta_{\mathbf{s}_0}, \varphi_1 \wedge \varphi_2, 0) = \min(R_1, R_2)$. We also know $L_1 \leq R_1$, and $L_2 \leq R_2$ which implies $\min(L_1, L_2) \leq \min(R_1, R_2)$. Therefore, we can conclude $\mathsf{LB4TL}(\sigma^\theta_{\mathbf{s}_0}, \varphi_1 \wedge \varphi_2,\ 0\ ; b) \leq \rho(\sigma^\theta_{\mathbf{s}_0}, \varphi_1 \wedge \varphi_2, 0)$.

The result of Proposition. 2 can also be utilized to introduce the following result.

Lemma 3. For any formula φ belonging to DT-STL framework in positive normal form, and $b > 0$, for a given trajectory $\sigma^{\theta}_{\mathbf{s}_0} = \mathbf{s}_0, \mathbf{s}_1, \ldots, \mathbf{s}_K$, if $\mathsf{LB4TL}(\sigma^{\theta}_{\mathbf{s}_0}, \varphi,\ 0; b) > 0$, then $\sigma^{\theta}_{\mathbf{s}_0} \models \varphi$, where LB4TL is a computation graph for DT-STL robustness, but with the softplus activation utilized in min operation and the swish activation employed in max.

Proof. Let's denote the set of predicates that are contributing to robustness computation of a DT-STL formula φ as, $\mathcal{A} = \{a_1 > 0, a_2 > 0, \cdots, a_N > 0\}$. The DT-STL formula, φ can be expanded in terms of $\vee$ and $\wedge$ operations applied to predicates, $a > 0$ where $(a > 0) \in \mathcal{A}$ (see (Donzé and Maler, 2010)). In addition, for all predicates $(a > 0) \in \mathcal{A}$, we have $\mathsf{LB4TL}(\sigma^{\theta}_{\mathbf{s}_0}, (a > 0),\ 0\,; b) \leq \rho(\sigma^{\theta}_{\mathbf{s}_0}, (a > 0), 0)$, since both are equal to a. Therefore, we can start from the predicates $(a > 0) \in \mathcal{A}$, and utilize the result of the Proposition. 2 to conclude for any DT-STL formula φ we have, $\mathsf{LB4TL}(\sigma^{\theta}_{\mathbf{s}_0}, \varphi,\ 0; b) \leq \rho(\sigma^{\theta}_{\mathbf{s}_0}, \varphi, 0)$.

4.1 Training with LB4TL

Algorithm 2 Neurosymbolic policy learning

1: Initialize variables
2: **while** $\left(\min_{\mathbf{s}_0 \in \widehat{\mathcal{I}}} \left(\rho(\varphi, \sigma^{\theta^j}_{\mathbf{s}_0}, 0)\right) < \bar{\rho}\right)$ **do**
3: $\quad \mathbf{s}_0 \leftarrow$ `Sample from` $\widehat{\mathcal{I}}$
4: $\quad \sigma^{\theta^j}_{\mathbf{s}_0} \leftarrow$ `Simulate using policy` π_{θ^j}
5: $\quad d \leftarrow \nabla_\theta \mathsf{LB4TL}(\sigma^{\theta^j}_{\mathbf{s}_o})$ `using` $\sigma^{\theta^j}_{\mathbf{s}_o}$
6: $\quad \theta^{j+1} \leftarrow \theta^j + \mathsf{Adam}(d)$
7: $\quad j \leftarrow j + 1$
8: **end while**

In order to train the controller for all initial states, $\mathbf{s}_0 \in \mathcal{I}$ we solve the following optimization problem:

$$\theta^* = \arg\max_{\theta} \left(\mathbb{E}_{\mathbf{s}_0 \overset{u}{\sim} \mathcal{I}} \left[\rho(\varphi, \sigma^{\theta}_{\mathbf{s}_0}, 0)\right]\right),$$
$$\textbf{s.t.}\ \ \sigma^{\theta}_{\mathbf{s}_0}(k+1) = \mathbf{f}\left(\sigma^{\theta}_{\mathbf{s}_0}(k), \pi_\theta\left(\sigma^{\theta}_{\mathbf{s}_0}(k), k\right)\right).$$

that aims to increase the expectation of the robustness for initial states uniformly sampled from the set of initial states. Solving this optimization problem is equivalent to training the NN controller using a gradient-based algorithm (shown in Alg. 2). However, we terminate the algorithm once the robustness is above a pre-specified lower threshold $\bar{\rho}$. We also generate a population of samples from the set of initial states of the system, i.e. $\mathcal{I}$, for training purposes, and denote this set by $\widehat{\mathcal{I}}$.

5. EXPERIMENTAL EVALUATION

We now present an evaluation of the performance of our proposed method [3] on two case studies.

[3] To increase the efficiency of our training process, we check for $\min_{\mathbf{s}_0 \in \widehat{\mathcal{I}}} \left(\rho(\varphi, \sigma^{\theta^j}_{\mathbf{s}_0}, 0)\right)$ once every 50 gradient steps to make a decision on terminating the training algorithm.

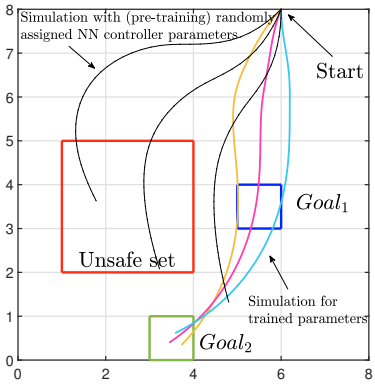

Fig. 2. Simulated trajectories of the NNFCS representing control for the simple car dynamics for the trained controller, in contrast with those for a controller initialized with random parameter values. The trajectories are initiated from the set of sampled initial conditions, which is $\theta = \frac{-3\pi}{4}, \frac{-5\pi}{8}, \frac{-\pi}{2}$.

Here we apply our technique on two examples. The problem setting and training results for experiments are provided in Table 1, and the simulation of their trained controllers are available in Figures 2,3. In these figures, the simulation for the random initial guess of control parameters are also presented in black color that clearly violate the specifications. The experiments are explained as follows, and the sampling time is $\delta t = 0.05$ sec.

Experiment 1: [Sequential Goals for a Simple Car Navigation Task]. We use a standard 3-dimensional model from (Yaghoubi and Fainekos, 2019) to represent the dynamics of a simple car as follows:

$$\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{\theta}\end{bmatrix} = \begin{bmatrix} v\cos(\theta)\\ v\sin(\theta)\\ \frac{v}{L}\tan(\gamma)\end{bmatrix}, \quad \begin{aligned} v &\leftarrow 2.5\tanh(0.5a_1) + 2.5,\\ \gamma &\leftarrow \pi/4\tanh(0.5a_2), \quad a_1,\ a_2 \in \mathbb{R}.\end{aligned} \tag{5}$$

Here (x, y) is the position, and θ is the heading angle. The controllers v, γ are the velocity and steering angle, respectively. Assuming the outputs of the NN controller are the (unbounded) values, $[a_1(k), a_2(k)]^\top$ we secure satisfaction of our controller bounds via Eq. (5)(see Table 1). The value of b in LB4TL is 10.

The task objective is for the car to first visit the goal region $Goal_1$ and then visit the region $Goal_2$. Further, we require this sequential task to be finished in 40 time steps. However, the car should always avoid the unsafe set Unsafe (see Figure 2). This temporal task can be formalized in DT-STL framework as follows:

$$\varphi_7 := \mathbf{F}_{[0,40]}\left[Goal_1 \wedge \mathbf{F}[Goal_2]\right] \wedge \mathbf{G}_{[0,40]}\left[\neg\ \text{Unsafe set}\right]$$

Experiment 2: [Reach/Avoid Tasks for a Quadrotor]. We use the 6-dimensional model for a quadrotor from (Yaghoubi and Fainekos, 2019) presented as follows:

$$\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{z}\\ \dot{v}_x\\ \dot{v}_y\\ \dot{v}_z\end{bmatrix} = \begin{bmatrix} v_x\\ v_y\\ v_z\\ \mathrm{g}\tan(u_1)\\ -\mathrm{g}\tan(u_2)\\ \mathrm{g} - u_3\end{bmatrix}, \quad \begin{aligned} u_1 &\leftarrow 0.1\tanh(0.1a_1),\\ u_2 &\leftarrow 0.1\tanh(0.1a_2),\\ u_3 &\leftarrow \mathrm{g} - 2\tanh(0.1a_3),\\ a_1, a_2, a_3 &\in \mathbb{R}, \mathrm{g} = 9.81.\end{aligned} \tag{6}$$

Expt.	Controller bounds	Initial states $\mathcal{I}$	Sampled initial states $\widehat{\mathcal{I}}$	Controller NN	Runtime (secs) LB4TL build	Runtime (secs) Training	Num. iterations
1	$(v,\gamma)\in[0,5]\times[-\frac{\pi}{4},\frac{\pi}{4}]$	$(x,y,\theta)(0)\in(6,8)^2\times[\frac{-3\pi}{4},\frac{-\pi}{2}]$	corners, center	$[4,10,2]$	1.31	72.9	750
2	$(u_1,u_2)\in[-0.1,0.1]^2$ $u_3\in[7.81,\ 11,81]$	$(x,y,z,v_x,v_y,v_z)(0)\in$ $[0.02,\ 0.05]\times[0,\ 0.05]\times[0]^4$	corners, center	$[7,10,3]$	0.13	354.36	16950

Table 1. Description of the case studies and the training results. Both used tanh activation.

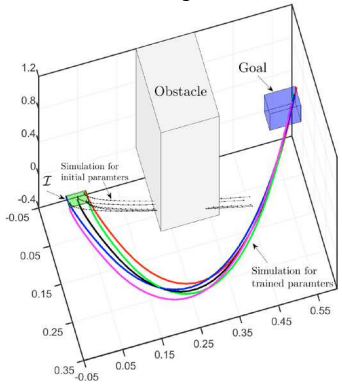

Fig. 3. Simulated trajectories for the trained controller in comparison to the trajectories for the NN controller initialized with random parameter values for the quadrotor case study. Trajectories are initiated from the set of sampled initial conditions consisting of the corners of $\mathcal{I}$ and its center.

Here, $\mathbf{x}=(x,y,z)$ respectively denotes the quadrotor's positions and $\mathbf{v}=(v_x,v_y,v_z)$ denote its velocities along the three coordinate axes. The control inputs u_1,u_2,u_3 respectively represent the pitch, roll, and thrust inputs. Assuming the outputs of the NN controller are the (unbounded) values, $[a_1(k),a_2(k),a_3(k)]^\top$ we secure satisfaction of our controller bounds (see Table 1) in Eq. (6). The value of parameter b in LB4TL is 20.

The quadrotor launches from a position in the set $\mathcal{I}$, and the task objective is to visit the goal set while avoiding obstacles. The projections of the obstacle 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. This temporal task can be formalized in DT-STL framework as, $\varphi_2=\mathbf{G}_{[0,35]}[\neg\text{ Obstacle}]\wedge\mathbf{F}_{[32,35]}[\text{Goal}]$.

Comparison. We now compare against the smooth semantics in (Gilpin et al., 2020; Pant et al., 2017; Leung et al., 2019) and empirically demonstrate that LB4TL outperforms them when used for training NN controllers. We also show that with increasing complexity of the DT-STL formula, the other smooth semantics show significant increases in runtime during gradient computation while LB4TL scales well. Given a fixed initial guess for the control parameters and a fixed set of sampled initial states $\widehat{\mathcal{I}}$, we run this algorithm, 4 times, and we utilize the following smooth semantics, each time ($b=10$).

(1) The first one is the smooth semantics proposed in (Pant et al., 2017) that replaces the min()/ max() operators in Eq. (3) with:

$$\widetilde{\min}(a_1,\cdots,a_\ell)=-\frac{1}{b}\log\left(\sum_{i=1}^{\ell}e^{-ba_i}\right),\quad \widetilde{\max}(a_1,\cdots,a_\ell)=\frac{1}{b}\log\left(\sum_{i=1}^{\ell}e^{ba_i}\right).$$

(2) The second one is the smooth semantics proposed in (Gilpin et al., 2020) that replaces min()/ max() operators in Eq. (3) with:

θ^0	(Gilpin et al., 2020) Runtime(sec)	(Pant et al., 2017) Runtime(sec)	STLCG Runtime(sec)	LB4TL Runtime(sec)
1^{st} guess	1465	626	NF[−1.3057]	104
2^{nd} guess	1028	1017	NF[−1.3912]	580
3^{rd} guess	2444	NF[−0.0821]	2352	419
4^{th} guess	NF[−0.1517]	NF[−0.9681]	NF[−0.4281]	429
5^{th} guess	617	NF[−1.2562]	1946	124
Average	2159	2488	3019	331

Table 2. Comparison between policy training with LB4TL and the other smooth robustness semantics.

$$\widetilde{\min}(a_1,\cdots,a_\ell)=-\frac{1}{b}\log\left(\sum_{i=1}^{\ell}e^{-ba_i}\right),\quad \widetilde{\max}(a_1,\cdots,a_\ell)=\sum_{i=1}^{\ell}\frac{a_ie^{ba_i}}{\sum_{i=1}^{\ell}e^{ba_i}}.$$

(3) The third one is the computation graph proposed in (Leung et al., 2019). This computation graph reformulates the robustness semantics in an RNN like structure and utilizes this graph for back-propagation.
(4) The last one is LB4TL that is introduced in this work.

We utilize Pytorch's automatic differentiation toolbox for all the examples. We also utilize the vehicle navigation case study with its provided specification as our case study. The mentioned DT-STL formula can be rephrased as:

$$\bigvee_{i=1}^{39}\left[\mathbf{F}_{[0,i]}\left[Goal_1\right]\wedge\mathbf{F}_{[i+1,40]}[Goal_2]\right]\wedge\mathbf{G}_{[0,40]}\left[\neg\text{ Unsafe set}\right],$$

that is a combination of 78 different future formulas, which results in a quite large size for LB4TL and is a great candidate to showcase the superiority of LB4TL comparing to the existing smooth semantics in terms of training runtime. We also assume a ReLU neural network with similar structure proposed in that example.

Since the runtime of the training algorithm is also highly related to the choice of initial guess for the controller, we repeat this experiment 5 times, and we assign a unique initial guess for the controller on all the 4 examples in a specific experiment. Table. 2 shows the report of training runtimes for all the experiments. In case the proposed example of smooth semantics in an experiment is unable to solve for a valid controller within 1 hour, we report it as NF[ρ^{end}], that implies the training did not finish. This also reports the minimum robustness $\rho^{end}=\min_{\mathbf{s}_0\in\widehat{\mathcal{I}}}\left(\rho(\varphi,\sigma^{\theta^{(j)}}_{\mathbf{s}_0},0)\right)$, where j is the last iteration before termination. Assuming the runtime for NF[.] to be 3600 sec, the average of the training runtime for the first, second, and third objective functions are 2159, 2488, and 3019 sec (via a Core i9 CPU), respectively. This is while the average of training runtime for our objective function (LB4TL) is 331 sec, which shows LB4TL is a more convenient choice for the training process when the specification becomes more complex. [4]

[4] Our experimental results show, for simple specifications, the performance of LB4TL and the previous smooth semantics are similar.

5.1 Controller Verification

In order to verify the results, we use a statistical verification technique based on conformal prediction (Vovk et al., 2005). Here, we pick 10^5 i.i.d. initial states from $\mathcal{I}$ and simulate their corresponding trajectories using the trained controller; then we validate the following guarantee for the NN controllers in our closed-loop system [5]: $\Pr\left[\Pr[\sigma^{\theta}_{\mathbf{s}_0} \models \varphi] \geq 99.98\%\right] \geq 99.5\%$.

6. CONCLUSION

We introduce LB4TL, a smooth computation graph to lower bound the robustness degree of a DT-STL specification. We present a neurosymbolic algorithm that uses informative gradients from LB4TL to design NN controllers to satisfy DT-STL specifications. We show the efficacy of our training algorithm on a two case studies and present a comparison with existing work to demonstrate the significance of our proposed techniques.

ACKNOWLEDGEMENTS

This work was supported by the National Science Foundation through grants CAREER (SHF-2048094), CNS-1932620, CNS-2039087, FMitF-1837131, CCF-SHF-1932620; funding by Toyota R&D and Siemens Corporate Research through the USC Center for Autonomy and AI, and the Airbus Institute for Eng. Research.

REFERENCES

Balakrishnan, A. and Deshmukh, J.V. (2019). Structured reward shaping using signal temporal logic specifications. In *Proc. of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 3481–3486.

Chua, K., Calandra, R., McAllister, R., and Levine, S. (2018). Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *Advances in neural information processing systems*, 31.

Donzé, A. and Maler, O. (2010). Robust satisfaction of temporal logic over real-valued signals. In *International Conference on Formal Modeling and Analysis of Timed Systems*, 92–106. Springer.

Dutta, S., Chen, X., and Sankaranarayanan, S. (2019). Reachability analysis for neural feedback systems using regressive polynomial rule inference. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, 157–168.

Fainekos, G. and Pappas, G.J. (2006). Robustness of temporal logic specifications. In *Proc. of Formal Approaches to Testing and Runtime Verification*, 178–192.

Fang, B., Jia, S., Guo, D., Xu, M., Wen, S., and Sun, F. (2019). Survey of imitation learning for robotic manipulation. *International Journal of Intelligent Robotics and Applications*, 3, 362–369.

Gilpin, Y., Kurtz, V., and Lin, H. (2020). A smooth robustness measure of signal temporal logic for symbolic control. *IEEE Control Systems Letters*, 5(1), 241–246.

[5] See (Hashemi et al., 2024) for details on obtaining the guarantee, but it essentially involves estimating the quantile of the empirical distribution of the robustness values over the calibration set containing 10^5 trajectories, and using properties of the Beta distribution to gain confidence in this quantile.

Hashemi, N., Hoxha, B., Prokhorov, D., Fainekos, G., and Deshmukh, J. (2024). Scaling learning based policy optimization for temporal tasks via dropout. *arXiv preprint arXiv:2403.15826*.

Hashemi, N., Hoxha, B., Yamaguchi, T., Prokhorov, D., Fainekos, G., and Deshmukh, J. (2023a). A neurosymbolic approach to the verification of temporal logic properties of learning-enabled control systems. In *ICCPS*, 98–109.

Hashemi, N., Qin, X., Deshmukh, J.V., Fainekos, G., Hoxha, B., Prokhorov, D., and Yamaguchi, T. (2023b). Risk-awareness in learning neural controllers for temporal logic objectives. In *Proc. of the American Control Conference*, 4096–4103.

Leung, K., Aréchiga, N., and Pavone, M. (2019). Backpropagation for parametric stl. In *2019 IEEE Intelligent Vehicles Symposium (IV)*, 185–192. IEEE.

Leung, K., Arechiga, N., and Pavone, M. (2021). Backpropagation through signal temporal logic specifications: Infusing logical structure into gradient-based methods. In *Algorithmic Foundations of Robotics*, 432–449.

Li, X., Ma, Y., and Belta, C. (2018). A policy search method for temporal logic specified reinforcement learning tasks. In *American Control Conference (ACC)*, 240–245. IEEE.

Li, X., Vasile, C.I., and Belta, C. (2017). Reinforcement learning with temporal logic rewards. In *Proc. of IROS*, 3834–3839. IEEE.

Maler, O. and Nickovic, D. (2004). Monitoring temporal properties of continuous signals. In *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, 152–166. Springer.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Pant, Y.V., Abbas, H., and Mangharam, R. (2017). Smooth operator: Control using the smooth robustness of temporal logic. In *2017 IEEE Conference on Control Technology and Applications (CCTA)*, 1235–1240. IEEE.

Pant, Y.V., Abbas, H., Quaye, R.A., and Mangharam, R. (2018). Fly-by-logic: control of multi-drone fleets with temporal logic objectives. In *Proc. of Int. Conf. on Cyber-Physical Systems*, 186–197.

Ramachandran, P., Zoph, B., and Le, Q.V. (2017). Searching for activation functions. *arXiv preprint arXiv:1710.05941*.

Raman, V., Donzé, A., Maasoumy, M., Murray, R.M., Sangiovanni-Vincentelli, A., and Seshia, S.A. (2014). Model predictive control with signal temporal logic specifications. In *Proc. of CDC*, 81–87. IEEE.

Vovk, V., Gammerman, A., and Shafer, G. (2005). *Algorithmic learning in a random world*, volume 29.

Yaghoubi, S. and Fainekos, G. (2019). Worst-case satisfaction of STL specifications using feedforward neural network controllers: A Lagrange multipliers approach. *ACM Trans. on Embedded Computing Systems*, 18(5S).