Neural Abstraction-Based Controller Synthesis and Deployment

RUPAK MAJUMDAR and MAHMOUD SALAMATI, Max Planck Institute for Software Systems, Kaiserslautern, Germany
SADEGH SOUDJANI, Newcastle University, Newcastle, United Kingdom

Abstraction-based techniques are an attractive approach for synthesizing correct-by-construction controllers to satisfy high-level temporal requirements. A main bottleneck for successful application of these techniques is the memory requirement, both during controller synthesis (to store the abstract transition relation) and in controller deployment (to store the control map).

We propose memory-efficient methods for mitigating the high memory demands of the abstraction-based techniques using *neural network representations*. To perform synthesis for reach-avoid specifications, we propose an on-the-fly algorithm that relies on compressed neural network representations of the forward and backward dynamics of the system. In contrast to usual applications of neural representations, our technique maintains soundness of the end-to-end process. To ensure this, we correct the output of the trained neural network such that the corrected output representations are sound with respect to the finite abstraction. For deployment, we provide a novel training algorithm to find a neural network representation of the synthesized controller and experimentally show that the controller can be correctly represented as a combination of a neural network and a look-up table that requires a substantially smaller memory.

We demonstrate experimentally that our approach significantly reduces the memory requirements of abstraction-based methods. We compare the performance of our approach with the standard abstraction-based synthesis on several models. For the selected benchmarks, our approach reduces the memory requirements respectively for the synthesis and deployment by a factor of 1.31×10^5 and 7.13×10^3 on average, and up to 7.54×10^5 and 3.18×10^4. Although this reduction is at the cost of increased off-line computations to train the neural networks, all the steps of our approach are parallelizable and can be implemented on machines with higher number of processing units to reduce the required computational time.

CCS Concepts: • **Theory of computation** → *Abstraction*; • **Computing methodologies** → *Representation of mathematical functions*; **Computational control theory**;

Additional Key Words and Phrases: Abstraction-based control, neural networks, compact representations, formal synthesis

1 INTRODUCTION

Designing controllers for safety-critical systems with formal correctness guarantees has been studied extensively in the past two decades, with applications in robotics, power systems, and medical devices [1, 24, 26]. Abstraction-based controller design (ABCD) has emerged as an approach that can algorithmically construct a controller with formal correctness guarantees on systems with non-linear dynamics and bounded adversarial disturbances [3, 29, 31, 37, 40] and complex behavioral specifications. ABCD schemes construct a *finite abstraction* of a dynamical system that has continuous state and input spaces, and solve a two-player graph game on the abstraction. When the abstraction is related to the original system through an appropriate behavioral relation (alternating bisimulation or feedback refinement [31]), the winning strategy of the graph game can be refined to a controller for the original system. Finite abstractions can be computed analytically when the system dynamics are known and certain Lipschitz continuity properties hold. Even when the system dynamics are unknown, one can use data-driven methods to learn finite abstractions that are correct with respect to a given confidence [8, 20, 30].

A main bottleneck of ABCD is the memory requirement, both in representing the finite abstract transition relation and in representing the controller. First, the state and input spaces of the abstraction grow exponentially with the system and input dimensions, respectively, and the size of the abstract transition relation grows quadratically with the abstract states and linearly with the input states. While symbolic encodings using BDDs can be used, in practice, the transition relation very quickly exceeds the available RAM. Memory-efficient methods sometimes exploit the analytic description of the system dynamics or growth bounds [16, 28, 34], but these techniques are not applicable when the finite abstractions are learned directly from sampled system trajectories, or when a compact analytical expression of the growth bound is not available. Second, the winning strategy in the graph game is extracted as a look-up table mapping winning states to one or more available inputs. Thus, the controller representation is also exponential in the system dynamics. Such controllers cannot be deployed on memory-constrained embedded systems.

In this work, we address the memory bottleneck using approximate, compressed, representations of the transition relation and the controller map using neural networks. We learn an approximate representation of the abstract transition relation as a neural network with a fixed architecture. In contrast to the predominant use of neural networks to learn a generalization of an unknown function through sampling, we train the network on the entire data set (the transition relation or the controller map) offline. We store the transitions on disk, and train our networks in batch mode by bringing blocks of data into the RAM as needed. The trained network is small and fits into RAM. Since the training of the network minimizes error but does not eliminate it, we apply a *correction* to the output to ensure that the representation is *sound* with respect to the original finite abstraction, i.e., every trajectory in the finite abstraction is preserved in the compressed representation. We propose an on-the-fly synthesis approach that works directly on the corrected representation of the forward and backward dynamics of the system. Although we present our results with respect to reach-avoid specifications, our approach can be generalized to other classes of properties and problems (e.g., linear temporal logic specifications [2]) in which the solution requires the computation of the set of predecessors and successors in the underlying transition system.

Similarly, we store the winning strategy as a look-up table mapping states to sets of valid inputs on disk and propose a novel training algorithm to find a neural network representation of the synthesized controller. The network is complemented with a look-up table that provides "exceptions" in which the network deviates from the winning strategy. We experimentally demonstrate that a controller can be correctly represented as a combination of a neural network and a look-up table that requires a substantially smaller memory than the original representation.

An important aspect of our approach is that, instead of using neural networks for learning an unknown data distribution, we train them over the entire data domain. Therefore, in contrast to many other applications wherein neural networks provide function representation and generalization over the unseen data, we are able to provide *formal soundness guarantees* for the performance of the trained neural representations over the whole dataset.

Our compression scheme uses additional computation to learn a compressed representation and avoid the memory bottleneck. In our implementation, the original relations are stored on the hard drive and data batches are loaded sequentially into the RAM to perform the training. Hard drives generally have much higher memory sizes compared to the RAM, but reading data from the hard drive takes much longer. However, data access during training is predictable and we can perform prefetching to hide the latency. During the synthesis, the trained corrected neural representations fit into the RAM. In contrast, a disk-based synthesis algorithm does not have predictable disk access patterns and is unworkable. Similarly, the deployed controller only consists of the trained compact representation and (empirically) a small look-up table, which can be loaded into the RAM of the controlling chip for the real-time operation of the system.

We evaluate the performance of our approach on several examples of different difficulties and show that it is effective in reducing the memory requirements at both synthesis and deployment phases. For the selected benchmarks, our method reduces the space requirements of synthesis and deployment respectively by factors of 1.31×10^5 and 7.13×10^3 in average, and up to 7.54×10^5 and 3.18×10^4, compared to the abstraction-based method that requires storing the full transition system. Moreover, we empirically show that, unlike other encodings, the memory requirement of our method is not affected by the system dimension on the considered benchmarks.

In summary, our main contributions are:

- Proposing a novel and sound representation scheme for compressing finite transition systems using the expressive power of neural networks;
- Proposing a novel on-the-fly controller synthesis method using the corrected neural network representations of forward and backward dynamics;
- Proposing an efficient scheme for compressing the controller computed by abstraction-based synthesis methods;
- Demonstrating significant reduction in the memory requirements by orders of magnitude through a set of standard benchmarks.[1]

The rest of this paper is organized as follows. After a brief discussion of related works, we give a high-level overview of our proposed approach in Section 1.2. The preliminaries and the problem statements are given in Section 2. We provide the details of our synthesis and deployment algorithms in Sections 3 and 4, respectively. In Section 5, we provide experimental results of applying our approach to several examples. We state the concluding remarks in Section 6.

1.1 Related Work

Synthesis via reinforcement learning. The idea of using neural networks as function approximators to represent tabular data for synthesis purposes has been used in different fields such as reinforcement learning (RL) literature and aircraft collision avoidance system design. RL algorithms try to find an optimal control policy by iteratively guiding the interaction between the agent and the environment modeled as a Markov decision process [39]. When the space of the underlying model is finite and small, q-tables are used to represent the required value functions and the policy. When the space is large and possibly uncountable, such finite q-tables are replaced with neural networks

---

[1]Our implementations are available online at https://github.com/msalamati/Neural-Representation

as function approximators. Convergence guarantees that hold with the q-table representation [4] are not valid for non-tabular setting [5, 6, 43]. A similar behavior is observed in our setting: we lose the correctness guarantees in our approach without correcting the output of the neural network representations of the transition systems and the tabular controller.

Neural-aided controller synthesis. Constructing neural network representations of the dynamics of the control system and using them for synthesis is studied in specific application domains including the design of unmanned airborne collision avoidance systems [19]. The central idea of [19] is to start from a large look-up table representing the dynamics, train a neural network on the look-up table, and use it in the dynamic programming for issuing horizontal and vertical advisories. Several techniques are used to reduce the storage requirement since the obtained score table—that is the table mapping every discrete state-input pair into the associated score—becomes huge in size (hundreds of gigabytes of floating numbers). Since simple techniques such as down sampling and block compression [23], are unable to achieve the required storage reduction, Julian et al. have shown that deep neural networks can successfully approximate the score table [18]. However, as in the RL controller synthesis, there is no guarantee that the control input computed using the neural representation matches the one computed using the original score table. In contrast, our corrected neural representations are guaranteed to produce formally correct controllers.

Reactive synthesis. Binary decision diagrams (BDDs) are used extensively in the reactive synthesis literature to represent the underlying transition systems [12, 32]. While BDDs are compact enough for low-order dynamical systems, recent synthesis tools such as SCOTS v2.0 [35] have already migrated into the non-BDD setting in order to avoid the large runtime overheads. In fact, motivated by reducing the required memory foot print, the current trend is to synthesize controllers in a non-BDD on the fly to eliminate the need for storing the transition system [16, 21, 22, 25, 28, 34]. These memory-efficient methods exploit the analytic description of the system dynamics or growth bounds. In contrast, our technique is applicable also to the case where the finite abstractions are learned directly from the sampled system trajectories, i.e., when no compact analytical expression of the dynamics and growth bounds are available.

Verifying systems with neural controllers. An alternative approach developed for safety-critical systems is to use neural networks as a representation of the controller and learn the controller using techniques such as reinforcement learning and data-driven predictive control [9, 41]. In this approach, the controller synthesis stage does not provide any safety guarantee on the closed loop system, i.e., on the feedback connection of the neural controller and the physical system. Instead, the safety of the closed-loop system is verified a posteriori for the designed controller. Ivanov et al. have considered dynamical systems with sigmoid-based neural network controllers, used the fact that sigmoid is the solution to a quadratic differential equation to transform the composition of the system and the neural controller into an equivalent hybrid system, and studied reachability properties of the closed-loop system by utilizing existing reachability computation tools for hybrid systems [15]. Huang et al. have considered dynamical systems with Lipschitz continuous neural controllers and used Bernstein polynomials for approximating the input-output model of the neural network [14]. Development of formal verification ideas for closed-loop systems with neural controllers has led into emergence of dedicated tools such as NNV [42] and POLAR [13]. While these methods provide guarantees on closed-loop control system with neural controllers, they can only consider *finite horizon* specifications for a given set of initial states. In contrast, we consider controllers that are synthesized for *infinite horizon* specifications.

Minimizing the memory foot print for symbolic controllers. Girard et al. have proposed a method to reduce the memory needed to store safety controllers by determinizing them, i.e., choosing one control input per state such that an algebraic decision diagram (ADD) representing the control law is minimized [10, 16]. Zapreev et al. have provided two methods based on greedy algorithms
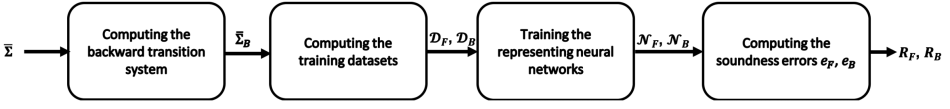
Fig. 1. Graphical description of the proposed scheme for compressing finite abstractions.

and symbolic regression to reduce the redundancy existing in the controllers computed by the abstraction-based methods [44]. Both of the ADD scheme in [10, 16] and the BDD-based scheme in [44] have the capability to determinize the symbolic controller and reduce its memory foot print. However, the computed controller still suffers from the additional runtime overhead of the ADD/BDD encoding. Further, as mentioned by the authors of [44], their regression-based method is not able to represent the original controller with high accuracy. In contrast, our tool produces real-valued representations for symbolic controllers and can (additionally) be computed on top of the simplified version found by either of the methods proposed in [16, 44].

Compressed representations for model predictive controllers (MPCs). Hertneck et al. have proposed a method to train an approximate neural controller representing the original robust (implicit) MPC satisfying the given specification [11]. While reducing the online computation time is the main motivation in implicit MPCs, minimizing the memory foot print is the main objective in explicit MPCs. Salamati et al. have proposed a method which is based on solving an optimization to compute a memory-optimized controller with mixed-precision coefficients used for specifying the required coefficients [36]. Our method considers a different class of controllers that can fulfill infinite horizon temporal specifications.

1.2 Overview of the Proposed Approach

In this subsection, we provide a high-level description of our approach for both synthesis and deployment.

Corrected neural representations. Figure 1 gives a pictorial description of the steps for computing a corrected neural network representation. Given a finite abstraction $\bar{\Sigma}$ that corresponds to the forward dynamics of the system and stored on the hard drive, we first compute the transition system $\bar{\Sigma}_B$ corresponding to the backward dynamics. Next, we extract the input-output training datasets $\mathcal{D}_F$ and $\mathcal{D}_B$ respectively from the forward and backward systems, and store them on the hard drive. Each data point contains one state-input pair and the characterization of ℓ_∞ ball for the corresponding reachable set. We train two neural networks $\mathcal{N}_F$ and $\mathcal{N}_B$ such that they represent compressed input-output surrogates for the datasets $\mathcal{D}_F$ and $\mathcal{D}_B$, respectively. Finally, we compute the *soundness errors* e_F and e_B which correspond to the difference between the output of $\mathcal{N}_F$ and $\mathcal{N}_B$ and the respective values in $\mathcal{D}_F$ and $\mathcal{D}_B$, calculated over all of the state-input pairs. We use the computed errors e_F and e_B in order to construct the corrected neural representations R_F and R_B. We will get memory savings by using R_F and R_B instead of $\bar{\Sigma}$ and $\bar{\Sigma}_B$, respectively.

Synthesis. Figure 2 gives a pictorial description of our proposed synthesis algorithm for a reach-avoid specification with the target set *Goal* and obstacle set *Avoid* as subsets of the state space. Let $W_0 \subseteq \bar{X}$ represents a discrete under-approximation of the target set *Goal*. We initialize the winning set as $L = W_0$, the controller as $C = \emptyset$, and the set of state-input pairs that must be added to the controller as $\Gamma_0 = \emptyset$. In each iteration, we compute the set of new states that belong to the winning set and update the controller accordingly, until no new state is added to L. To this end, we first use R_B and its corresponding soundness error e_B to compute a set of candidates S_i out of which some belong to L and it is guaranteed that there will be no winning state outside of S_i in the ith iteration. We use R_F and its corresponding soundness error e_F to compute the set of new
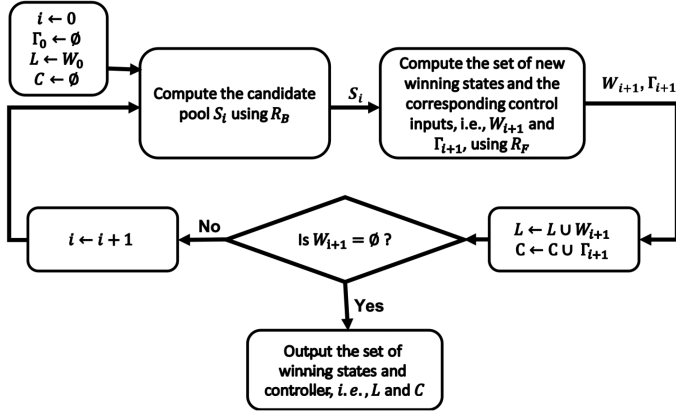
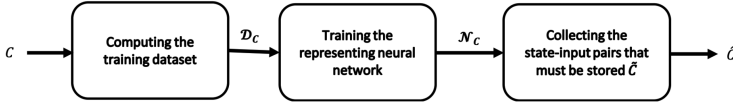Fig. 2. Graphical description of the proposed synthesis scheme.

Fig. 3. Graphical description of the proposed scheme for compressing controllers.

winning states $W_{i+1} \subseteq S_i$. We also compute the set of control inputs for every new winning state and compute the corresponding set of state-input pairs Γ_{i+1} that must be added to the controller. Finally, if $W_i = \emptyset$, we terminate the computations as we already have computed the winning set L and the controller C. Otherwise, we add the new winning set of states and state-input pairs, respectively, into the overall winning set ($L \leftarrow L \cup W_{i+1}$) and the controller ($C \leftarrow C \cup \Gamma_{i+1}$), and repeat the steps in the next iteration.

Deployment. Figure 3 shows our method for compressing controllers that are obtained from abstraction-based approaches. In the first step, we collect the training dataset $\mathcal{D}_C$ and reformat it to become appropriate for our specific formulation of a classification problem. Each data point contains one state and an encoding of the corresponding set of control inputs. We then train a neural network $\mathcal{N}_C$ on the data with the loss function designed for this specific classification problem. Finally, we find all the states at which the output label generated by $\mathcal{N}_C$ is invalid, and store the corresponding state-input pair in a look-up table, denoted by $\hat{C}$. We experimentally show that $\hat{C}$ only contains a very small portion of state-input pairs.

2 PRELIMINARIES

2.1 Notation

We denote the set of integer numbers and natural numbers including zero by $\mathbb{Z}$ and $\mathbb{N}$, respectively. We use the notation $\mathbb{R}$ and $\mathbb{R}_{>0}$ to denote respectively the set of real numbers and the set of positive real numbers. We use superscript $n > 0$ with $\mathbb{R}$ and $\mathbb{R}_{>0}$ to denote the Cartesian product of n copies of $\mathbb{R}$ and $\mathbb{R}_{>0}$ respectively. For a vector $\boldsymbol{a} \in \mathbb{R}^d$, we denote its ith component, element-wise absolute value and ℓ_2 norm by $\boldsymbol{a}(i)$, $|\boldsymbol{a}|$ and $\|\boldsymbol{a}\|$, respectively. For a pair of vectors $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^d$, $[\![\boldsymbol{a}, \boldsymbol{b}]\!]$ denotes the hyper-rectangular set $[\boldsymbol{a}(1), \boldsymbol{b}(1)] \times \cdots \times [\boldsymbol{a}(d), \boldsymbol{b}(d)]$. Further, given $\boldsymbol{c} \in \mathbb{R}^d$, $\boldsymbol{c} + [\![\boldsymbol{a}, \boldsymbol{b}]\!]$ is another hyper-rectangular set which is shifted compared to $[\![\boldsymbol{a}, \boldsymbol{b}]\!]$ to the extent determined by $\boldsymbol{c}$. Similarly, for a vector $\boldsymbol{\eta} \in \mathbb{R}^d$ and a pair of vectors $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^d$, for which $\boldsymbol{a} = \alpha \boldsymbol{\eta}$, $\alpha \in \mathbb{Z}$ and $\boldsymbol{b} = \beta \boldsymbol{\eta}$, $\beta \in \mathbb{Z}$,

we define $[\![a, b]\!]_\eta = \prod_{i=1}^d A_i$, where $A_i = \{\gamma \eta(i) \mid \gamma \in \mathbb{Z}, \ \alpha \leq \gamma \leq \beta\}$. Given $c \in \mathbb{R}^n$ and $\varepsilon \in \mathbb{R}^n_{>0}$, the ball with center c and radius ε in $\mathbb{R}^n$ is denoted by $\Omega_\varepsilon(c) := \{x \in \mathbb{R}^n \mid \lvert x - c \rvert \leq \varepsilon\}$. For two integers $a, b \in \mathbb{Z}$, we define $[a; b] = \{c \in \mathbb{Z} \mid a \leq c \leq b\}$.

Let A be a finite set of size $|A|$. The empty set is denoted by $\emptyset$. When A inherits a coordinate structure, i.e., when its members are vectors on the Euclidean space, $A(i)$ denotes the projection of set A onto its ith dimension. Further, we use the notation A^∞ to denote the set of all finite and infinite sequences formed using the members of A. Our control tasks are defined using a subset of Linear Temporal Logic (LTL). In particular, we use the *until* operator $\mathcal{U}$. Let p and q be subsets of $\mathbb{R}^n$ and $\rho = (x_0, x_1, \dots)$ be an infinite sequence of elements from $\mathbb{R}^n$. We write $\rho \models p\mathcal{U}q$ if there exists $i \in \mathbb{N}$ s.t. $x_i \in q$ and $x_j \in p$ for all $0 \leq j < i$. For the detailed syntax and semantics of LTL, we refer to [2] and references therein.

2.2 Control Systems

We consider the class of continuous-state continuous-time control systems characterized by the tuple $\Sigma = (X, U, W, f)$, where $X \subset \mathbb{R}^n$ is the compact state space, $U \subset \mathbb{R}^m$ is the compact input space, and $W \subset \mathbb{R}^n$ is the disturbance space being a compact hyper-rectangular set of disturbances which is symmetric with respect to the origin (i.e., for every $w \in W$ also it is the case that $-w \in W$). The vector field $f : X \times U \to X$ is such that $f(\cdot, u)$ is locally Lipschitz for all $u \in U$. The evolution of the state of Σ is characterized by the differential inclusion

$$\dot{x}(t) \in f(x(t), u(t)) + W. \tag{1}$$

Given a *sampling time* $\tau > 0$, an initial state $x_0 \in X$, and a constant input $u \in U$, define the *continuous-time trajectory* $\zeta_{x_0, u}$ of the system on the time interval $[0, \tau]$ as an absolutely continuous function $\zeta_{x_0, u} : [0, \tau] \to X$ such that $\zeta_{x_0, u}(0) = x_0$, and $\zeta_{x_0, u}$ satisfies the differential inclusion $\dot{\zeta}_{x_0, u}(t) \in f(\zeta_{x_0, u}(t), u) + W$ for almost all $t \in [0, \tau]$. Given τ, x_0, and u, we define $Sol(x_0, u, \tau)$ as the set of all $x \in X$ such that there is a continuous-time trajectory $\zeta_{x_0, u}$ with $\zeta(\tau) = x$. A sequence $x_0, x_1, x_2, \dots$ is a *time-sampled trajectory* for a continuous control system if $x_0 \in X$ and for each $i \geq 0$, we have $x_{i+1} \in Sol(x_i, u_i, \tau)$ for some $u_i \in U$.

2.3 Finite Abstractions

In order to satisfy a temporal specification on the trajectories of the system, it is generally needed to over-approximate the dynamics of the system with a finite discrete-time model. Let $\bar{X} \subset X$ and $\bar{U} \subset U$ be the finite sets of states and inputs, computed by (uniformly) quantizing the compact state and input spaces X and U using the rectangular discretization partitions of size $\eta_x \in \mathbb{R}^n_{>0}$ and $\eta_u \in \mathbb{R}^m_{>0}$, respectively. A *finite abstraction* associated with the dynamics in Equation (1) is characterized by the tuple $\bar{\Sigma} : (\bar{X}, \bar{U}, T_F)$, where $T_F \subseteq \bar{X} \times \bar{U} \times \bar{X}$ denotes the system's *forward-in-time transition system*. The transition system T_F is defined such that

$$(\bar{x}, \bar{u}, \bar{x}') \in T_F \Leftrightarrow \exists (x, u, x') \in \Omega_{\frac{\eta_x}{2}}(\bar{x}) \times \Omega_{\frac{\eta_u}{2}}(\bar{u}) \times \Omega_{\frac{\eta_x}{2}}(\bar{x}') \text{ s.t. } x' \in Sol(x, u, \tau).$$

When the dynamics in Equation (1) are known and satisfy the required Lipschitz continuity condition, the finite abstraction can be constructed using the method proposed in [31]. For systems with unknown dynamics, data-driven schemes for learning finite abstractions can be employed [8, 20, 30]. By abusing the notation, we denote the *reachable set* for a state-input pair $(\bar{x}, \bar{u}) \in \bar{X} \times \bar{U}$ by $T_F(\bar{x}, \bar{u}) = \{\bar{x}' \in \bar{X} \mid \bar{x}' \in Sol(\bar{x}, \bar{u}, \tau)\}$. We assume that the reachable sets take hyper-rectangular form, meaning that for every $\bar{x} \in \bar{X}$, $\bar{u} \in \bar{U}$ the corresponding reachable set $H = T_F(\bar{x}, \bar{u})$ can be rewritten as $H = \prod_{i=1}^n H(i)$, where $H(i)$ corresponds to the projection of the set H onto its ith coordinate. Otherwise, in case that H is not hyper-rectangular, it is

over-approximated by $\prod_{i=1}^{n} H(i)$. Note that $\bar{\Sigma}$ can in general correspond to a *non-deterministic* control system, i.e., $|T_F(\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}})| > 1$ for some $\bar{\boldsymbol{x}} \in \bar{X}, \bar{\boldsymbol{u}} \in \bar{U}$.

Given $\bar{\Sigma}$, one can easily compute the characterization of the *backward-in-time* dynamics as

$$\bar{\Sigma}_B = (\bar{X}, \bar{U}, T_B), \ T_B = \{ (\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}}, \bar{\boldsymbol{x}}') \in \bar{X} \times \bar{U} \times \bar{X} \mid (\bar{\boldsymbol{x}}', \bar{\boldsymbol{u}}, \bar{\boldsymbol{x}}) \in T_F \}. \tag{2}$$

A *trajectory* of $\bar{\Sigma}$ is a finite or infinite sequence $\boldsymbol{x}_0, \boldsymbol{x}_1, \boldsymbol{x}_2, \ldots \in \bar{X}^{\infty}$, such that for each $i \geq 0$, there is a control input $\bar{\boldsymbol{u}}_i \in \bar{U}$ such that $(\boldsymbol{x}_i, \bar{\boldsymbol{u}}_i, \boldsymbol{x}_{i+1}) \in T_F$. The operator $\mathrm{Pre}(\cdot)$ acting on sets $P \subseteq \bar{X}$ is defined as

$$\mathrm{Pre}(P) = \{ \bar{\boldsymbol{x}} \in \bar{X} \mid \exists \bar{\boldsymbol{u}} \in \bar{U} \ s.t. \ T_F(\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}}) \subseteq P \}.$$

Finally, to compute an over-approximating set of the discrete states that have overlap with a hyper rectangular set $[\![\boldsymbol{x}_{lb}, \boldsymbol{x}_{ub}]\!]$, we define the (over-approximating) quantization mapping as

$$\bar{K}(\boldsymbol{x}_{lb}, \boldsymbol{x}_{ub}) = \{ \bar{\boldsymbol{x}}' \in \bar{X} \mid [\![\bar{\boldsymbol{x}}' - \boldsymbol{\eta_x}/2, \bar{\boldsymbol{x}}' + \boldsymbol{\eta_x}/2]\!] \cap [\![\boldsymbol{x}_{lb}, \boldsymbol{x}_{ub}]\!] \neq \emptyset \}.$$

Similarly, the under-approximating quantization mapping is defined as

$$\underline{K}(\boldsymbol{x}_{lb}, \boldsymbol{x}_{ub}) = \{ \bar{\boldsymbol{x}}' \in \bar{X} \mid [\![\bar{\boldsymbol{x}}' - \boldsymbol{\eta_x}/2, \bar{\boldsymbol{x}}' + \boldsymbol{\eta_x}/2]\!] \subseteq [\![\boldsymbol{x}_{lb}, \boldsymbol{x}_{ub}]\!] \}.$$

2.4 Controllers

For a finite abstraction $\bar{\Sigma} = (\bar{X}, \bar{U}, T_F)$, a feedback controller is denoted by $C \subseteq \bar{X} \times \bar{U}$. The set of valid control inputs at every state $\bar{\boldsymbol{x}} \in \bar{X}$ is defined as $C(\bar{\boldsymbol{x}}) \coloneqq \{ \bar{\boldsymbol{u}} \in \bar{U} \mid (\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}}) \in C \}$. We denote the feedback composition of $\bar{\Sigma}$ with C as $C \parallel \bar{\Sigma}$. For an initial state $\bar{\boldsymbol{x}}^* \in \bar{X}$, the set of trajectories of $C \parallel \bar{\Sigma}$ having length $k \in \mathbb{N}$ is the set of sequences $\bar{\boldsymbol{x}}_0, \bar{\boldsymbol{x}}_1, \bar{\boldsymbol{x}}_2, \ldots, \bar{\boldsymbol{x}}_{k-1}$, s.t. $\bar{\boldsymbol{x}}_0 = \bar{\boldsymbol{x}}^*, \bar{\boldsymbol{x}}_{i+1} \in T_F(\bar{\boldsymbol{x}}_i, \bar{\boldsymbol{u}}_i)$ and $\bar{\boldsymbol{u}}_i \in C(\bar{\boldsymbol{x}}_i)$ for $i \in [0; k-2]$.

2.5 Neural Networks

A *neural network* $\mathcal{N}(\boldsymbol{\theta}, \cdot) \colon \mathbb{R}^d \to \mathbb{R}^q$ of depth $v \in \mathbb{N}$ is a parameterized function which transforms an input vector $\boldsymbol{a} \in \mathbb{R}^d$ into an output vector $\boldsymbol{b} \in \mathbb{R}^q$, and is constructed by the forward combination of v functions as follows:

$$\mathcal{N}(\boldsymbol{\theta}, \boldsymbol{a}) = G_v(\boldsymbol{\theta}_v, G_{v-1}(\boldsymbol{\theta}_{v-1}, \ldots, G_2(\boldsymbol{\theta}_2, G_1(\boldsymbol{\theta}_1, \boldsymbol{a})))),$$

where $\boldsymbol{\theta} = (\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_v)$ and $G_i(\boldsymbol{\theta}_i, \cdot) \colon \mathbb{R}^{p_{i-1}} \to \mathbb{R}^{p_i}$ denotes the ith *layer* of $\mathcal{N}$ parameterized by $\boldsymbol{\theta}_i$ with $p_0 = d, p_i \in \mathbb{N}$ for $i \in [1; v]$ and $p_v = q$. The ith layer of the network, $i \in [1; v]$, takes an input vector in $\mathbb{R}^{p_{i-1}}$ and transforms it into an output representation in $\mathbb{R}^{p_i}$ depending on the value of parameter vector $\boldsymbol{\theta}_i$ and type of the used *activation function* in G_i. During the *training phase* of the network, the set of parameters $\boldsymbol{\theta}$ is learned over the *training set* which consists of a number of input-output pairs $\{(\boldsymbol{a}_k, \boldsymbol{b}_k) \mid k = 1, 2, \ldots, N\}$, in order to achieve the highest performance with respect to an appropriate metric such as mean squared error. For a trained neural network, we drop its dependence on the parameters $\boldsymbol{\theta}$. In this paper, we characterize a neural network of depth v using its corresponding list of layer sizes, i.e., $(p_1, p_2, \ldots, p_v)$, and the type of the activation function used, e.g., hyperbolic tangent, Rectified Linear Unit (ReLU), etc.

Neural networks can be used for both *regression* and *classification* tasks. In a regression task, the goal is to predict a numerical value given an input, whereas, a classification task requires predicting the correct class label for a given input. In order to measure performance of the trained neural network, we consider *prediction error*. Note that prediction error is different from the metrics such as *mean squared error* (MSE) which are used during the training phase for defining the objective function for the training. The prediction error for regression and classification tasks is defined

differently. For *our regression tasks*, we define the prediction error for a trained neural network $\mathcal{N}$ over a training set $\{(a_k, b_k) \mid k = 1, 2, \ldots, N\}$ as

$$e = \max_{k \in [1;N]} |\mathcal{N}(a_k) - b_k|.$$

In this paper, we consider the classification tasks wherein there may exist more than one valid class label for each input. Therefore, the training set would be of the form $\{(a_k, b_k) \mid k = 1, 2, \ldots, N\}$, where $b_k \in \{0, 1\}^q$ and $b_k(i) = 1$ iff $i \in [1;q]$ corresponds to a valid label at a_k. Since the number of valid labels for each input can be different, we define the prediction error of a trained classifier $\mathcal{N}$ in the following way:

$$err = \frac{|\{k \in [1;N] \mid b_k(i) = 0 \text{ with } i = argmax(\mathcal{N}(a_k))\}|}{N}.$$

For a given neural network $\mathcal{N}$ with the training set $\{(a_k, b_k) \mid k = 1, 2, \ldots, N\}$, we define the *continuity index* as

$$\alpha_{\mathcal{N}} = \max_{1 \leq i,j \leq N, \ i \neq j} \frac{\|\mathcal{N}(a_i) - \mathcal{N}(a_j)\|}{\|a_i - a_j\|}. \tag{3}$$

2.6 Problem Statement

We now consider the controller synthesis problem for finite abstractions w.r.t. a reach-avoid specification. Let *Goal*, *Avoid* $\subseteq X$, *Goal* $\cap$ *Avoid* $= \emptyset$ be the set of states representing the target and unsafe spaces, respectively. The *winning domain* for the finite abstraction $\bar{\Sigma} = (\bar{X}, \bar{U}, T_F)$ is the set of states $\bar{x}^* \in \bar{X}$ such that there exists a feedback controller C such that all trajectories of $C \parallel \bar{\Sigma}$, which are started at $\bar{x}^*$, satisfy the given specification Φ. $\bar{x}_0 = \bar{x}^*, \bar{x}_1, \bar{x}_2, \ldots \models \Phi$. The aim is to find the set of the winning states L together with a feedback controller C such that $C \parallel \bar{\Sigma}$ satisfies the reach-avoid specification Φ. To compute the winning domain and the controller, one can use the methods from reactive synthesis. For many of interesting control systems, size of T_F in the finite abstraction becomes huge. This restricts the application of reactive-synthesis-based methods for computing the controller. Therefore, we are looking for a method which uses compressed surrogates of T_F to save memory. In particular, we want to train two *corrected neural* surrogates, i.e., neural network representations whose output is corrected to maintain the soundness property: R_F for the forward-in-time dynamics and R_B for the backward-in-time dynamics.

> PROBLEM 1. **Inputs:** *Finite abstraction* $\bar{\Sigma} = (\bar{X}, \bar{U}, T_F)$ *and the specification* $\Phi = \neg Avoid \, \mathcal{U} \, Goal$.
> **Outputs:** *Corrected neural representations* R_F *and* R_B, *winning domain* L *and a feedback controller* C *for* $\bar{\Sigma}$ *such that* $C \parallel \bar{\Sigma}$ *realizes* Φ.

It is important to notice that any solution for this problem is required to provide a *formal guarantee* on the satisfaction of Φ, i.e., the reach-avoid specification Φ *must* be satisfied under any disturbance affecting the control systems.

Let $C \in \bar{X} \times \bar{U}$ be the computed controller for the abstraction $\bar{\Sigma}$ such that $C \parallel \bar{\Sigma}$ realizes a given specification Φ. The size of this controller can be large due to the large number of discrete state and inputs. For deployment purposes, we would like to compute a corrected neural controller $\hat{C} := \bar{X} \rightarrow \bar{U}$ s.t. $\hat{C} \parallel \bar{\Sigma}$ realizes Φ.

> PROBLEM 2. **Inputs:** *Controller* C *computed for the discrete control system* $\bar{\Sigma}$ *and the specification* Φ *s.t.* $C \parallel \bar{\Sigma}$ *realizes* Φ.
> **Outputs:** *A corrected neural controller* $\hat{C}$ *such that* $\hat{C} \parallel \bar{\Sigma}$ *realizes* Φ.

ALGORITHM 1: Regression-based compression algorithm for finite abstractions

---

Data: Forward dynamics $\bar{\Sigma}$ and learning rate λ

1 Compute backward dynamics $\bar{\Sigma}_B$ and the datasets $\mathcal{D}_F$ and $\mathcal{D}_B$ using Equations (2), (4), and (5)
2 Train neural networks $\mathcal{N}_F$ on the dataset $\mathcal{D}_F$ and train $\mathcal{N}_B$ on $\mathcal{D}_B$ using the learning rate λ
3 Compute the soundness errors $\boldsymbol{e}_F$ and $\boldsymbol{e}_B$ using Equation (6)
4 Compute the final corrected representations R_F and R_B using Equations (7) and (8)
    Result: corrected neural representations R_F and R_B

---

3 SYNTHESIS

One approach to formally synthesize controllers for a given specification is to store the transition system corresponding to quantization of the state and input spaces, and to use the methods from reactive synthesis to design a controller. However, the memory required to store these transition systems increases exponentially with the number of state variables, which causes a memory blow-up for many real-world systems. In this section, we propose our memory-efficient algorithm for synthesizing controllers to satisfy reach-avoid specifications for finite abstractions and reach-avoid specifications. Our method requires computation of corrected neural representations for the finite abstraction. Computation of these representations is discussed in Section 3.1. Later, in Section 3.3, we show how our synthesis method makes use of the computed representations.

3.1 Corrected Neural Representations for Finite Abstractions

Let $\bar{\Sigma} = (\bar{X}, \bar{U}, T_F)$ be a finite abstraction. In this section, we show that T_F can be approximated by some *generator functions*. In particular, we show how to compute generator functions $R_F : \bar{X} \times \bar{U} \to \mathbb{R}^n \times \mathbb{R}^n_{\geq 0}$ and $R_B : \bar{X} \times \bar{U} \to \mathbb{R}^n \times \mathbb{R}^n_{\geq 0}$ which can produce characterization of an ℓ_∞ ball corresponding to the over-approximation of forward- and backward-in-time reachable sets, respectively, for every state-input pair picked from $\bar{X} \times \bar{U}$. Our aim is to use the expressive power of neural networks to represent the behavior of $\bar{\Sigma}$ such that the memory requirements significantly decrease.

Our compression scheme is summarized in Algorithm 1. We first compute the backward-in-time system $\bar{\Sigma}_B$ using Equation (2). We then calculate the over-approximating ℓ_∞ ball for every state-input pair. Let $c_F(\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}}) \in X$ and $r_F(\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}}) \in \mathbb{R}^n_{\geq 0}$ characterize the tightest ℓ_∞ ball such that

$$(\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}}, \bar{\boldsymbol{x}}') \in T_F \Leftrightarrow \|\bar{\boldsymbol{x}}' - c_F(\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}})\|_\infty \leq r_F(\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}}) - \boldsymbol{\eta}_x/2.$$

This is illustrated in Figure 4 in two-dimensional space for a given state-input pair $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}})$. The dotted red rectangle corresponds to the hyper-rectangular reachable set. The center $c_F(\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}})$ and radius $r_F(\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}})$ are computed using the lower-left and upper-right corners of the reachable set denoted, respectively, by $g_{FL}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}})$ and $g_{FU}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}})$. Then, we have $c_F(\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}}) = (g_{FU}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}}) + g_{FL}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}}))/2$ and $r_F(\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}}) = (g_{FU}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}}) - g_{FL}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}}))/2 + \boldsymbol{\eta}_x/2$. At the end of the first step we have computed and stored the dataset

$$\mathcal{D}_F = \{ ((\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}}), (c_F(\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}}), r_F(\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}}))) \mid \bar{\boldsymbol{x}} \in \bar{X}, \bar{\boldsymbol{u}} \in \bar{U} \}. \tag{4}$$

Note that every data-point in $\mathcal{D}_F$ consists of two pairs: one specifies a state-input pair $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}})$ and the other one characterizes the center and radius corresponding to the over-approximating ℓ_∞ disc $(c_F(\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}}), r_F(\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}}))$. Similarly, we need to store another dataset corresponding to the backward dynamics. First, we define $c_B(\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}}) \in X$ and $r_B(\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}}) \in \mathbb{R}^n_{\geq 0}$ characterizing the tightest ℓ_∞ ball such that

$$(\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}}, \bar{\boldsymbol{x}}') \in T_B \Leftrightarrow \|\bar{\boldsymbol{x}}' - c_B(\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}})\|_\infty \leq r_B(\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}}) - \boldsymbol{\eta}_x/2.$$

The dataset corresponding to the backward dynamics is of the following form

$$\mathcal{D}_B = \{ ((\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}}), (c_B(\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}}), r_B(\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}}))) \mid \bar{\boldsymbol{x}} \in \bar{X}, \bar{\boldsymbol{u}} \in \bar{U} \}. \tag{5}$$
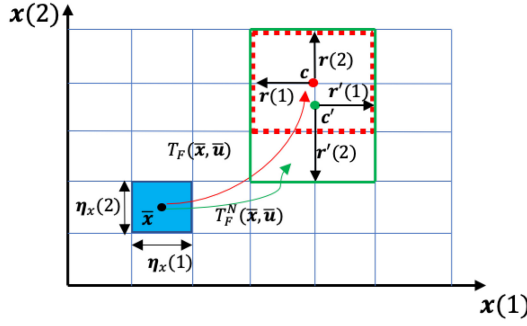
Fig. 4. Comparing the set of successor states in the transition system T_F and its representation T_F^N. We have $c = c_F(\bar{x}, \bar{u})$, $r_F(\bar{x}, \bar{u}) = [r(1), r(2)]^\top$, $c' = \mathcal{N}_F^c(\bar{x}, \bar{u})$ and $\mathcal{N}_F^r(\bar{x}, \bar{u}) = [r'(1), r'(2)]^\top$.
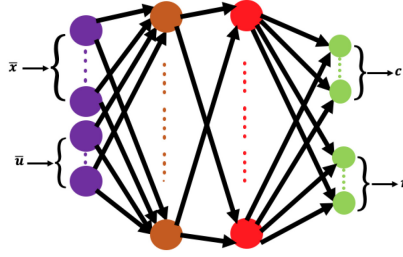

Fig. 5. The regression-based configuration used in compression of abstractions. The input to the neural network includes state-input pair $(\bar{x}, \bar{u})$, and the output includes the pair (c, r) corresponding to the center and radius of the rectangular reachable set, respectively. Right: The classification-based representation of finite abstractions. The representation receives a state-input pair $(\bar{x}, \bar{u})$. In the output, y_{lb} and y_{ub} correspond to the lower-left and upper-right corners for the rectangular reachable set.

The size of $\mathcal{D}_F$ and $\mathcal{D}_B$ grows exponentially with the dimension of state space. Hence, we store both the datasets $\mathcal{D}_F$ and $\mathcal{D}_B$ (potentially) into the hard drive. Next, we take the datasets $\mathcal{D}_F$ and $\mathcal{D}_B$, for which we train neural networks $\mathcal{N}_F$ and $\mathcal{N}_B$, taking the state-input pairs $(\bar{x}, \bar{u})$ as input and $(c_F(\bar{x}, \bar{u}), r_F(\bar{x}, \bar{u}))$ as output, and try to find an input-output mapping minimizing mean squared error (MSE). For systems with state and input spaces of dimensions n and m, the input and output layers of both neural networks are of sizes $n + m$ and $2n$, respectively. The configuration of the neural networks which we used is illustrated in Figure 5. During training, we load batches of data from $\mathcal{D}_F$ and $\mathcal{D}_B$, which are stored on the the hard drive, into the RAM. We use the stochastic gradient descent (SGD) method to minimize MSE.

As mentioned earlier, in contrast to the usual applications wherein neural networks are used to represent an unknown distribution, we have the full dataset and require computing representations which are *sound* with respect to the input dataset. A sound representation for the given finite abstractions produces reachable sets that include $T_F(\bar{x}, \bar{u})$ for every state-input pair $(\bar{x}, \bar{u})$. For instance, the solid green rectangle in Figure 4 contains the set of reachable states corresponding to $\mathcal{N}_F(\bar{x}, \bar{u})$ and contains the set of states included in the dotted red rectangle, i.e., $T_F(\bar{x}, \bar{u})$. Therefore, we can say that the representation $\mathcal{N}_F$ is sound for the pair $(\bar{x}, \bar{u})$. In order to guarantee *soundness*, we need to compute the maximum error induced during the training process among all the training data points. To that end, we go over all the state-input pairs (which are stored on the hard drive) and compute the maximum error in approximating the centers of the ℓ_∞ balls, denoted by e_F^c, e_B^c

and radius e_F^r, e_B^r corresponding to the forward and backward representations:

$$e_F^c = \max_{\bar{x} \in \bar{X}, \bar{u} \in \bar{U}} |c_F(\bar{x}, \bar{u}) - \mathcal{N}_F^c(\bar{x}, \bar{u})|, \quad e_F^r = \max_{\bar{x} \in \bar{X}, \bar{u} \in \bar{U}} |r_F(\bar{x}, \bar{u}) - \mathcal{N}_F^r(\bar{x}, \bar{u})|.$$

Similarly, for the backward dynamics,

$$e_B^c = \max_{\bar{x} \in \bar{X}, \bar{u} \in \bar{U}} |c_B(\bar{x}, \bar{u}) - \mathcal{N}_B^c(\bar{x}, \bar{u})|, \quad e_B^r = \max_{\bar{x} \in \bar{X}, \bar{u} \in \bar{U}} |r_B(\bar{x}, \bar{u}) - \mathcal{N}_B^r(\bar{x}, \bar{u})|.$$

We define

$$e_F = e_F^c + e_F^r, \quad e_B = e_B^c + e_B^r, \tag{6}$$

and use the errors e_F and e_B to compute the *corrected representations* R_F and R_B, corresponding to $\mathcal{N}_F$ and $\mathcal{N}_B$, as described next. Let R_F^c and R_F^r correspond to the center and radius components of R_F. Similarly, R_B^c and R_B^r correspond to the center and radius components of R_B. For state-input pair $(\bar{x}, \bar{u}) \in \bar{X} \times \bar{U}$, we define

$$R_F^c(\bar{x}, \bar{u}) = \mathcal{N}_F^c(\bar{x}, \bar{u}), \quad R_F^r(\bar{x}, \bar{u}) = \mathcal{N}_F^r(\bar{x}, \bar{u}) + e_F, \tag{7}$$

for the forward dynamics, and

$$R_B^c(\bar{x}, \bar{u}) = \mathcal{N}_B^c(\bar{x}, \bar{u}), \quad R_B^r(\bar{x}, \bar{u}) = \mathcal{N}_B^r(\bar{x}, \bar{u}) + e_B, \tag{8}$$

for the backward dynamics.

Let us define the forward transition system computed using the trained neural network as follows

$$T_F^N = \left\{ (\bar{x}, \bar{u}, \bar{x}') \in \bar{X} \times \bar{U} \times \bar{X} \,|\, \bar{x}' \in \bar{K}\left(\mathcal{N}_F^c(\bar{x}, \bar{u}) - \mathcal{N}_F^r(x, \bar{u}) - e_F, \mathcal{N}_F^c(\bar{x}, \bar{u}) + \mathcal{N}_F^r(\bar{x}, \bar{u}) + e_F \right) \right\}, \tag{9}$$

where $\mathcal{N}_F^c(\cdot, \cdot)$, $\mathcal{N}_F^r(\cdot, \cdot)$ denote the components of the output of $\mathcal{N}_F(\cdot, \cdot)$ corresponding to the center and radius of disc, respectively. Similarly, we can define the transition system T_B^N corresponding to the backward dynamics as follows

$$T_B^N = \left\{ (\bar{x}, \bar{u}, \bar{x}') \in \bar{X} \times \bar{U} \times \bar{X} \,|\, \bar{x}' \in \bar{K}\left(\mathcal{N}_B^c(\bar{x}, \bar{u}) - \mathcal{N}_B^r(\bar{x}, \bar{u}) - e_B, \mathcal{N}_B^c(\bar{x}, \bar{u}) + \mathcal{N}_B^r(\bar{x}, \bar{u}) + e_B \right) \right\}. \tag{10}$$

The following lemma states that we can use the trained neural networks to compute sound transition systems for both forward and backward dynamics. However, our synthesis approach does not require the computation of T_F^N and T_B^N and only uses the compressed representations $\mathcal{N}_F$ and $\mathcal{N}_B$.

LEMMA 3.1. *Transition systems T_F^N and T_B^N computed by* (9) *and* (10) *are sound for T_F and T_B, i.e., we have $T_F \subseteq T_F^N$ and $T_B \subseteq T_B^N$.*

To reduce the level of conservativeness, we require that T_F^N and T_B^N do not contain too many additional edges compared to T_F and T_B. The *mismatch rate* of the forward and backward dynamics are defined as

$$d_F := \frac{|T_F^N \setminus T_F|}{|T_F|}, \quad d_B := \frac{|T_B^N \setminus T_B|}{|T_B|}.$$

If the trained representations are accurate, the mismatch rate is low, which results in a *less restrictive* representation.

Remark 1. The method proposed in this section formulates the computation of the representations as a regression problem, wherein the representative neural networks are supposed to predict the center and radius corresponding to ℓ_∞ reachable sets. In Section 3.2, we describe a classification-based formulation for compressing finite abstractions, wherein the representative neural networks are supposed to predict the vectorized indices corresponding to the lower-left and upper-right corners of the reachable set. We experimentally show that this second formulation, while being more memory demanding, provides a less conservative representation compared to the formulation discussed in this section.

ALGORITHM 2: Computing classification-based representations of finite abstractions

---

Data: Forward dynamics $\bar{\Sigma}$ and learning rate λ

1 Compute backward dynamics $\bar{\Sigma}_B$ and the datasets $\mathcal{D}_F$ and $\mathcal{D}_B$ using Equations (2), (11), and (12)
2 Train neural networks $\mathcal{N}_F$ and $\mathcal{N}_B$ on the datasets $\mathcal{D}_F$ and $\mathcal{D}_B$ using the learning rate λ
3 Compute the set of misclassified state-input pairs E_F and E_B as in Equation (13)
4 Compute the set of transitions $\tilde{\mathcal{N}}_F$ and $\tilde{\mathcal{N}}_B$ associated with E_F and E_B as in Equation (15)
5 Compute the corrected neural representations R_F, R_B using Equations (16) and (17)

Result: R_F, R_B

3.2 Classification-Based Computation of Representations for Finite Abstractions

We proposed in Section 3.1 a formulation for training neural networks that can *guess* at any given state-input pair the center and radius of a hyper-rectangular over-approximation of the reachable states. This guess is then *corrected* using the computed soundness errors. A nice aspect of this formulation is that we only need to store the trained representations and their corresponding soundness errors. However, the result of using the soundness errors to correct the output values produced by the neural networks may give a very conservative over-approximation of the reachable sets, even when the trained representations have a very good performance on a large subset of the state-input pairs, since the soundness errors must be computed over *all* state-input pairs.

In this section, we provide an alternative formulation for computing a compressed representation of a given abstraction. Intuitively, our idea is to train neural network representations which can guess for any given state-input pair the *vectorized indices* corresponding to the lower-left and upper-right corner points of the hyper-rectangular reachable set. The architecture of the representation is shown in Figure 6. As illustrated, for every state-input pair $(\bar{x}, \bar{u}) \in \bar{X} \times \bar{U}$, the output of the representation gives the lower-left and upper-right corners of the rectangular set that is reachable by taking the control input $\bar{u}$ at the state $\bar{x}$. Algorithm 2 describes our classifier-based compression scheme for finite abstractions. We first compute the backward system $\bar{\Sigma}_B$ using Equation (2). We then compute the training datasets for both the forward and backward systems $\bar{\Sigma}$ and $\bar{\Sigma}_B$. For $\bar{\Sigma}$, let $g_{FU}: \bar{X} \times \bar{U} \to \bar{X}$ and $g_{FL}: \bar{X} \times \bar{U} \to \bar{X}$ denote the mappings from the state-input pair $(\bar{x}, \bar{u}) \in \bar{X} \times \bar{U}$ into the corresponding upper-right and lower-left corners of the rectangular reachable set from $(\bar{x}, \bar{u})$. We define $z_F: \bar{X} \times \bar{U} \to \{0, 1\}^{2 \sum_{i=1}^n |\bar{X}(i)|}$ with $|\bar{X}(i)|$ being the cardinality of the projection of $\bar{X}$ along the ith axis and $z_F(\bar{x}, \bar{u})(l) = 1$ if and only if

$$l = 2 \sum_{k=1}^{i-1} |\bar{X}(k)| + \mathcal{I}_{x, i}(g_{FL}(\bar{x}, \bar{u})(i)) \text{ or } l = 2 \sum_{k=1}^{i-1} |\bar{X}(k)| + |\bar{X}(i)| + \mathcal{I}_{x, i}(g_{FU}(\bar{x}, \bar{u})(i)),$$

for some $i \in \{1, 2, \ldots, n\}$. The indexing function $\mathcal{I}_{x, i}: \bar{X}(i) \to [1; |\bar{X}(i)|]$ maps every element of $\bar{X}(i)$ into a *unique* integer index in the interval $[1; |\bar{X}(i)|]$. The training dataset for $\bar{\Sigma}$ is defined as

$$\mathcal{D}_F := \{(\bar{x}, \bar{u}, z_F(\bar{x}, \bar{u})) \mid \bar{x} \in \bar{X} \text{ and } \bar{u} \in \bar{U}\}. \tag{11}$$

Intuitively, each element of the dataset $\mathcal{D}_F$ contains a state-input pair $(\bar{x}, \bar{u})$ and a vector $h \in \{0, 1\}^{2 \sum_{i=1}^n |\bar{X}(i)|}$ that has 1 only at the entries corresponding to $\mathcal{I}_{x, i}(g_{FL}(\bar{x}, \bar{u})(i))$ and $\mathcal{I}_{x, i}(g_{FU}(\bar{x}, \bar{u})(i))$ for $i \in \{1, 2, \ldots, n\}$. Similarly, we define $z_B: \bar{X} \times \bar{U} \to \{0, 1\}^{2 \sum_{i=1}^n |\bar{X}(i)|}$ for $\bar{\Sigma}_B$ such that $z_B(\bar{x}, \bar{u})(l) = 1$ if and only if

$$l = 2 \sum_{k=1}^{i-1} |\bar{X}(k)| + \mathcal{I}_{x, i}(g_{BL}(\bar{x}, \bar{u})(i)) \text{ or } l = 2 \sum_{k=1}^{i-1} |\bar{X}(k)| + |\bar{X}(i)| + \mathcal{I}_{x, i}(g_{BU}(\bar{x}, \bar{u})(i)),$$
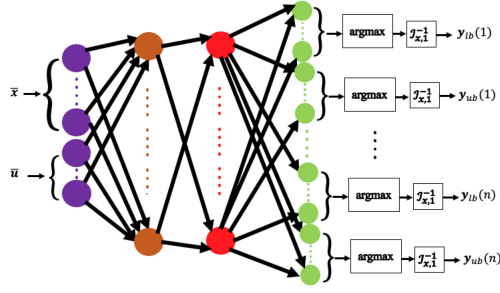
Fig. 6. The classification-based representation of finite abstractions. The representation receives a state-input pair $(\bar{x}, \bar{u})$. In the output, y_{lb} and y_{ub} correspond to the lower-left and upper-right corners for the rectangular reachable set.

for some $i \in \{1, 2, \ldots, n\}$. The training dataset for the backward dynamics is also defined similarly as

$$\mathcal{D}_B \coloneqq \{(\bar{x}, \bar{u}, z_B(\bar{x}, \bar{u})) \mid \bar{x} \in \bar{X} \text{ and } \bar{u} \in \bar{U}\}. \tag{12}$$

Once the training datasets are ready, we train the neural networks $\mathcal{N}_F$ and $\mathcal{N}_B$ respectively on the datasets $\mathcal{D}_F$ and $\mathcal{D}_B$. Note that the output layer of $\mathcal{N}_F$ and $\mathcal{N}_B$ will be a vector of size $2 \sum_{i=1}^{n} |\bar{X}(i)|$, while the final output of the representations are of size $2n$ (cf. Figure 6). These final outputs give an approximation of the coordinates of the lower-left and upper-right corners of the reachable set corresponding to the pair $(\bar{x}, \bar{u})$. Note that, because $\bar{X}$ was computed by equally partitioning over X, both the indexing function $\mathcal{I}_{x,i}$ and its inverse can be implemented in a memory-efficient way using floor and ceil operators. We then evaluate the performance of the trained neural networks $\mathcal{N}_F$ and $\mathcal{N}_B$. Let $\rho_{FL}(\bar{x}, \bar{u})$ and $\rho_{FU}(\bar{x}, \bar{u})$ denote respectively the estimated lower-left and upper-right corners of the reachable set estimated by $\mathcal{N}_F$. Define $\rho_{BL}(\bar{x}, \bar{u})$ and $\rho_{BU}(\bar{x}, \bar{u})$ similarly for $\mathcal{N}_B$, and let the set of misclassified state-input pairs be

$$E_F \coloneqq \{(\bar{x}, \bar{u}) \in \bar{X} \times \bar{U} \mid T_F(\bar{x}, \bar{u}) \setminus [\![\rho_{FL}(\bar{x}, \bar{u}), \rho_{FU}(\bar{x}, \bar{u})]\!]_{\eta_x} \neq \emptyset\}$$

$$E_B \coloneqq \{(\bar{x}, \bar{u}) \in \bar{X} \times \bar{U} \mid T_B(\bar{x}, \bar{u}) \setminus [\![\rho_{BL}(\bar{x}, \bar{u}), \rho_{BU}(\bar{x}, \bar{u})]\!]_{\eta_x} \neq \emptyset\}. \tag{13}$$

The *soundness error* of $\mathcal{N}_F$ and $\mathcal{N}_B$ can be considered as their misclassification rate:

$$err_F \coloneqq \frac{|E_F|}{|\bar{X} \times \bar{U}|} \quad \text{and} \quad err_B \coloneqq \frac{|E_B|}{|\bar{X} \times \bar{U}|}. \tag{14}$$

For the misclassified pairs in E_F and E_B, we extract the related transitions in the abstraction:

$$\tilde{\mathcal{N}}_F \coloneqq \{(\bar{x}, \bar{u}, \bar{x}') \mid (\bar{x}, \bar{u}) \in E_F, \bar{x}' \in T_F(\bar{x}, \bar{u})\}, \ \tilde{\mathcal{N}}_B \coloneqq \{(\bar{x}, \bar{u}, \bar{x}') \mid (\bar{x}, \bar{u}) \in E_B, \bar{x}' \in T_B(\bar{x}, \bar{u})\}. \tag{15}$$

Finally, we *correct* the output of neural network representations to maintain soundness

$$R_F(\bar{x}, \bar{u}) \coloneqq \begin{cases} [\![\rho_{FL}(\bar{x}, \bar{u}), \rho_{FU}(\bar{x}, \bar{u})]\!]_{\eta_x} & \text{if } (\bar{x}, \bar{u}) \notin E_F \\ \tilde{\mathcal{N}}_F(\bar{x}, \bar{u}) & \text{if } (\bar{x}, \bar{u}) \in E_F, \end{cases} \tag{16}$$

$$R_B(\bar{x}, \bar{u}) \coloneqq \begin{cases} [\![\rho_{BL}(\bar{x}, \bar{u}), \rho_{BU}(\bar{x}, \bar{u})]\!]_{\eta_x} & \text{if } (\bar{x}, \bar{u}) \notin E_B \\ \tilde{\mathcal{N}}_B(\bar{x}, \bar{u}) & \text{if } (\bar{x}, \bar{u}) \in E_B. \end{cases} \tag{17}$$

Note that these corrected neural representations are memory efficient only if the misclassification rates are small, i.e., the size of E_F and E_B are small compared with $\bar{X} \times \bar{U}$.

ALGORITHM 3: Controller synthesis algorithm

Data: Set $W_0 \subseteq \bar{X}$ and the corrected neural representations R_F and R_B

1 Initialize $C \leftarrow \emptyset$, $P_0 \leftarrow W_0$, $\Gamma_0 \leftarrow \emptyset$ and $i \leftarrow 0$

2 **while** $W_i \neq \emptyset$ **do**

3   Compute the candidate pool S_i using Equation (18)

4   Compute the set of new winning states W_{i+1} using Equation (19) and add them to the winning set ($P_{i+1} \leftarrow P_i \cup W_{i+1}$)

5   Compute the set of new state-input pairs Γ_{i+1} using Equation (20) and add them to the controller ($C \leftarrow C \cup \Gamma_{i+1}$)

6     $i \leftarrow i + 1$

7 **end**

8 $L \leftarrow P_i$

    Result: Controller C and its winning set L

3.3 On-the-Fly Synthesis

In the previous subsection, we described the computation of the compressed representations corresponding to the forward and backward dynamics for finite abstractions. In this subsection, we use these representations in order to synthesize formally correct controllers.

Our synthesis procedure is provided in Algorithm 3. It takes the representations R_F and R_B to synthesize a controller which fulfills the given reach-avoid specification. Let

$$W_0 = \{\, \bar{x} \in \bar{X} \mid [\![\bar{x} - \eta_x/2, \bar{x} + \eta_x/2]\!] \subseteq Goal \,\}$$

be a discrete under-approximation of the target set *Goal*. We take W_0 as the input and perform a fixed-point computation to solve the given reach-avoid game. We initialize the winning set and controller with $P_0 = W_0$ and $C = \emptyset$, and in each iteration, we add the new winning set of states and state-input pairs, respectively, into the overall winning set and the controller, until no new state is found ($W_{i+1} = \emptyset$).

Let W_i be the set of *new* winning states in the beginning of the ith iteration. Further, we denote the set of winning states in the beginning of the ith iteration by $P_i = \bigcup_{k=0}^{i} W_k$. In every iteration, for every $\bar{x} \in W_i$ and $\bar{u} \in \bar{U}$, we compute the backward over-approximating ℓ_∞ ball and discretize it to get the *candidate pool* S_i defined as

$$S_i := \bigcup_{\bar{u} \in \bar{U}} Y_i(\bar{u}), \tag{18}$$

with

$$Y_i(\bar{u}) := \bigcup_{\bar{x} \in W_i} \left(\bar{X} \cap \bar{K} \left(R_B^c(\bar{x}, \bar{u}) - R_B^r(\bar{x}, \bar{u}), R_B^c(\bar{x}, \bar{u}) + R_B^r(\bar{x}, \bar{u}) \right) \right),$$

where $R_B^c(\cdot, \cdot)$, $R_B^r(\cdot, \cdot)$ denote the components of the output of $R_B(\cdot, \cdot)$ corresponding to the center and radius of the ℓ_∞ ball, respectively. Note that we compute the candidate pool by running R_B over W_i *instead of* P_i. This is computationally beneficial, because $|W_i| \leq |P_i|$. Next lemma shows that S_i includes the *whole* set of new winning states W_{i+1}.

LEMMA 3.2. *Let the set of candidates S_i be as defined in Equation (18). Then, we have $W_{i+1} \subseteq S_i$ for all $i \geq 0$.*

PROOF. We prove this lemma by contradiction. Suppose that $W_{i+1} \nsubseteq S_i$. Then there exists at least one $\bar{x}^* \in W_{i+1} \setminus S_i$. Since $\bar{x}^* \in W_{i+1}$, we know that there exists at least one $\bar{u}^* \in \bar{U}$ such that $T_f(\bar{x}^*, \bar{u}^*) \subseteq P_i$ and $\bar{x}^* \notin P_i$. Moreover, since $\bar{x}^* \notin S_i$, by Equation (18) we get $T_f(\bar{x}^*, \bar{u}^*) \cap W_i = \emptyset$.

So, $T_f(\bar{x}^*, \bar{u}^*) \subseteq P_i \setminus W_i = P_{i-1}$. This gives $\bar{x}^* \in P_i$, which is a contradiction. This completes the proof. □

Now, we can use R_F, which represents the forward transition system, in order to choose the *legitimate* candidates out of S_i and add the new ones to W_{i+1}. Let

$$A = \{ \bar{x} \in \bar{X} \mid [\![\bar{x} - \eta_x/2, \bar{x} + \eta_x/2]\!] \cap Avoid \neq \emptyset \}$$

be a discrete over-approximation over the set of obstacles. The next lemma states that we can use the representation R_F to compute W_{i+1}.

LEMMA 3.3. *The set of states added to the winning set in the ith step can be computed as*

$$W_{i+1} = \{ \bar{x} \in S_i \mid \exists \bar{u} \in \bar{U} \ s.t. \bar{K}(R_F^c(\bar{x}, \bar{u}) - R_F^r(\bar{x}, \bar{u}), R_F^c(\bar{x}, \bar{u}) + R_F^r(\bar{x}, \bar{u})) \subseteq P_i \} \setminus (P_i \cup A). \quad (19)$$

PROOF. To prove this lemma, we denote $G = \{ \bar{x} \in S_i \mid \exists \bar{u} \in \bar{U} \ s.t. \ \bar{K}(R_F^c(\bar{x}, \bar{u}) - R_F^r(\bar{x}, \bar{u}), R_F^c(\bar{x}, \bar{u}) + R_F^r(\bar{x}, \bar{u})) \subseteq P_i \} \setminus (P_i \cup A)$, and show $W_{i+1} \subseteq G$ and $G \subseteq W_{i+1}$. The second direction ($G \subseteq W_{i+1}$) holds by definition. To prove the first direction ($W_{i+1} \subseteq G$), we note that $G \subseteq S_i$ and further, by the result of Lemma 3.2, we have $W_{i+1} \subseteq S_i$. Assume $W_{i+1} \nsubseteq G$. Then there should exist at least one $\bar{x}^* \in W_{i+1} \setminus G$. Note that $\bar{x}^* \in S_i \setminus G$. Since $\bar{x}^* \in S_i$, we get that there exists at least one $\bar{u}^* \in \bar{U}$ for which $T_F(\bar{x}^*, \bar{u}^*) \subseteq W_i$. Also, because $\bar{x}^* \notin G$, we have $T_F(\bar{x}^*, \bar{u}^*) \nsubseteq W_i$, which is a contradiction. Therefore, $W_{i+1} \subseteq G$. Hence the proof ends. □

In each iteration, we calculate Γ_i, which is the set of new state-input pairs that must be added into the controller, and is defined as

$$\Gamma_{i+1} = \left\{ (\bar{x}, \bar{u}) \mid \bar{x} \in W_{i+1}, \ \bar{K} \left(R_F^c(\bar{x}, \bar{u}) - R_F^r(\bar{x}, \bar{u}), R_F^c(\bar{x}, \bar{u}) + R_F^r(\bar{x}, \bar{u}) \right) \subseteq P_i \right\}. \quad (20)$$

Finally, If $W_{i+1} = \emptyset$, we can terminate the computations as we already have computed the winning set and the controller. Otherwise, we add W_i and Γ_i into the overall winning set ($P_{i+1} \leftarrow P_i \cup W_{i+1}$) and controller ($C \leftarrow C \cup \Gamma_{i+1}$) and restart the depicted process.

4 DEPLOYMENT

Once the controller C is computed such that $C \parallel \bar{\Sigma}$ realizes the given specification Φ, we need to deploy C onto an embedded controller platform, e.g., a microcontroller. Since such embedded controller platforms generally have a small on-board memory, we would like to minimize the size of the stored controller.

We define the *set of valid control inputs* corresponding to $\bar{x}$ as $C(\bar{x}) = \{ \bar{u} \mid (\bar{x}, \bar{u}) \in C \}$. The approach we proposed for finding representations for the finite abstractions may not work, since we are not allowed to over-approximate $C(\bar{x})$, and thus the set of valid control inputs is not representable as a compact ℓ_∞ ball described by its center and radius. The following example illustrates a disconnected $C(\bar{x})$, which cannot be represented by an ℓ_∞ ball.

Example 1. Consider a system with one-dimensional state and input spaces ($n = m = 1$). Figure 7 illustrates the set of transitions starting from the white middle box ($\bar{x} = 0$). Let the boxes with green check mark and red cross mark correspond to the target and obstacle states and C be the controller for the corresponding reach-avoid specification. Then, we have $\{ (0, 2), (0, 3), (0, -2), (0, -3) \} \subseteq C$ and $C(0) = \{ -2, -3, 2, 3 \}$. It is clear that $C(0)$ is a disconnected set, which is not characterizable by an ℓ_∞ ball.

In contrast to the symbolic regression method proposed in [44], we formulate the controller compression problem as a classification task, that is, we train a neural network which assigns every state to a list of scores over the set of control inputs, and picks the control input with the highest score. The configuration of the neural network is illustrated in Figure 8. The justification for our
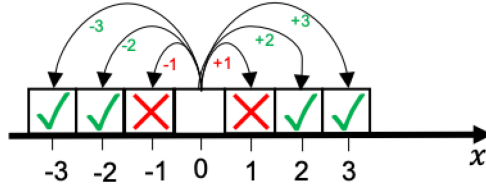
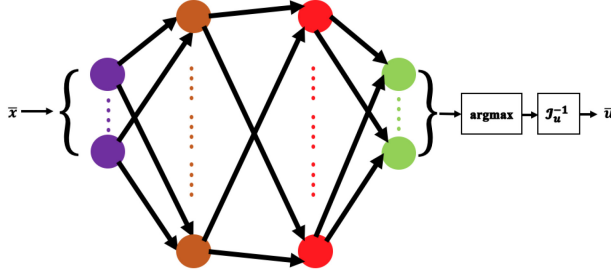Fig. 7. Illustration of a disconnected set of valid control inputs.

Fig. 8. The configuration used in compression of controllers. Given a state $\bar{x}$, the representation produces a corresponding control input $\bar{u}$.

formulation is that any representation for the controller can only perform well if it is trained over a dataset which respects the *continuity property*, i.e., neighboring states are not mapped into control input values which are very different from each other. A representation that respects the continuity property corresponds to a low continuity index (see Equation (3)). During the training phase, we keep all the valid control inputs and let the training process to choose which value respects the continuity property more, by minimization of the cost function. Therefore, our formulation automatically takes care of the *redundancy* problem by mapping a neighborhood in the state space into close-in-value control inputs to respect the continuity requirement of the trained representation. The reason that our formulation does not correspond to a *standard* classification setting is that during the training phase a *non-uniform* number of labels (corresponding to the control input values in the output stage of the neural network) per input (corresponding to the state values at the input layer of the neural network) are considered as valid, while we only will consider *one* label—corresponding to the highest score—as the trained representation's choice during the runtime.

Remark 2. In order to formulate the problem of finding a neural-network-based representation for the controller as a regression problem, first the training data must be *pre-processed* such that the continuity property is respected, i.e., the set of valid control-inputs per each state is pruned so that neighboring states are mapped to close-in-value control inputs. However, this pre-processing is time consuming and does not work efficiently in practice (see, e.g., [10, 44]).

Algorithm 4 summarizes the proposed procedure for computing a compressed representation for the original controller. In the first step, we need to store the training set

$$\mathcal{D}_C = \{(\bar{x}, h(\bar{x})) \mid (\bar{x}, \bar{u}) \in C \Leftrightarrow h(\bar{x})(\mathcal{I}_u(\bar{u})) = 1, \ (\bar{x}, \bar{u}) \notin C \Leftrightarrow h(\bar{x})(\mathcal{I}_u(\bar{u})) = 0\}, \quad (21)$$

where $\mathcal{I}_u : U \to [1; |\bar{U}|]$ is an *indexing function* for the control set $\bar{U}$, which assigns every value in $\bar{U}$ into a *unique* integer in the interval $[1; |\bar{U}|]$. Intuitively, each point in the dataset $\mathcal{D}_C$ contains a state $\bar{x} \in L$ and a vector $h(\bar{x})$ which is of length $|\bar{U}|$ and has ones at the entries corresponding to the valid control inputs and zeros elsewhere.

ALGORITHM 4: Compression algorithm for the controller

   Data: Controller C, learning rate λ
1 Compute the dataset $\mathcal{D}_C$ using Equation (21)
2 Train the neural network $\mathcal{N}_C$ on the dataset $\mathcal{D}_C$ using the learning rate λ
3 Compute the set of state-input pairs $\tilde{C}$ using Equation (23)
4 Compute $\hat{C}$ using Equation (24)
   Result: Corrected neural representation $\hat{C}$

Once the training dataset is ready, we can train a neural network $\mathcal{N}_C$ which takes $\bar{x} \in \bar{X}$ as input and approximates $\mathcal{I}_u^{-1}(argmax(h(\bar{x})))$ in the output, where $\mathcal{I}_u^{-1}(\cdot)$ denotes the inverse of the indexing function used in Equation (21).

Remark 3. Note that the output layer of $\mathcal{N}_C$ has to be of size $|\bar{U}|$ and for every $\bar{x} \in L$, we consider the value $\mathcal{I}_u^{-1}(argmax(\mathcal{N}_C(x)))$ as the final control input assigned by $\mathcal{N}_C$ to the state $\bar{x}$. Moreover, because $\bar{U}$ was computed by equally partitioning over U, both the indexing function $\mathcal{I}_u$ and its inverse can be implemented in a memory-efficient way using *floor* and *ceil* functions.

Once the neural network $\mathcal{N}_C$ is trained, we evaluate its performance by finding all the states $\bar{x}$ at which using $\mathcal{N}_C$ produces an *invalid* control input, i.e.,

$$E = \{\, \bar{x} \in L \mid \mathcal{I}_u^{-1}(argmax(\mathcal{N}_C(\bar{x}))) \notin C(\bar{x}) \,\}.$$

The misclassification rate of the trained classifier $\mathcal{N}_C$ is defined as:

$$err_C = \frac{|E|}{|L|}. \tag{22}$$

In order to maintain the guarantee provided by the original controller C, it is very important to *correct* the output of the trained representation, so that it outputs a valid control input at *every* state. In case the misclassification rate is small, we can store $\mathcal{N}_C$ together with $\tilde{C}$, where

$$\tilde{C} = \{\, (\bar{x}, \bar{u}) \mid \bar{x} \in E,\ \bar{u} \in C(\bar{x}) \,\}. \tag{23}$$

The final deployable controller $\hat{C}$ consists of both $\mathcal{N}_C$ and $\tilde{C}$, and is defined as

$$\hat{C}(\bar{x}) := \begin{cases} \mathcal{I}_u^{-1}(argmax(\mathcal{N}_C(\bar{x}))) & \text{if } \bar{x} \notin E \\ \tilde{C}(\bar{x}) & \text{if } \bar{x} \in E. \end{cases} \tag{24}$$

LEMMA 4.1. *Let $\hat{C}$ be as defined in Equation (24). The winning domain of both $\hat{C} \parallel \bar{\Sigma}$ and $C \parallel \bar{\Sigma}$ for satisfying a specification Φ is the same.*

Remark 4. Our deployment method preserves soundness. The input to our deployment approach is a formally guaranteed controller computed by any abstraction-based method. We train a neural representation that maps the states to a control input. This control input is valid for majority of the states. For the states that the control input is not valid, we keep the set of valid control inputs from the original controller and store them as a small look-up table. Therefore, the final corrected neural controller in Equation (24) is sound with respect to the original controller.

5 EXPERIMENTAL EVALUATION

We evaluate the performance of our proposed algorithms on several control systems, namely multi-dimensional cars [7, 35, 38], inverted pendulum [27] and TORA [17]. Dynamics of our control systems are listed in Table 1. We used configurations (1) and (2) in Table 1, respectively, for evaluating our methods for synthesis and deployment. We construct the transition system in all the case studies using the sampling approach in [20]. This approach generates T_F using sampled trajectories

Table 1. Catalog of Models used to Generate the Finite Abstractions in Section 5

Case study	Dynamics of the model	Configuration (1)				Configuration (2)			
		X	U	η_x	η_u	X	U	η_x	η_u
2D car $(\boldsymbol{x}(1), \boldsymbol{x}(2))$- position $(\boldsymbol{u}(1), \boldsymbol{u}(2))$- speed	$\begin{bmatrix} \dot{\boldsymbol{x}}(1) \\ \dot{\boldsymbol{x}}(2) \end{bmatrix} \in \begin{bmatrix} \boldsymbol{u}(1) \\ \boldsymbol{u}(2) \end{bmatrix} + W$ $\tau = 0.4, W = [-0.025, 0.025]^2$	$[0, 5]^2$	$[-1, 1]^2$	$\begin{bmatrix} 0.05 \\ 0.05 \end{bmatrix}$	$\begin{bmatrix} 0.23 \\ 0.23 \end{bmatrix}$	$[0, 10]^2$	$[-2, 2]^2$	$\begin{bmatrix} 0.025 \\ 0.025 \end{bmatrix}$	$\begin{bmatrix} 0.3 \\ 0.3 \end{bmatrix}$
3D car $(\boldsymbol{x}(1), \boldsymbol{x}(2))$- position $\boldsymbol{x}(3)$- angle $\boldsymbol{u}(1)$- speed $\boldsymbol{u}(2)$- turn rate	$\begin{bmatrix} \dot{\boldsymbol{x}}(1) \\ \dot{\boldsymbol{x}}(2) \\ \dot{\boldsymbol{x}}(3) \end{bmatrix} \in \begin{bmatrix} \boldsymbol{u}(1)\cos(\boldsymbol{x}(3)) \\ \boldsymbol{u}(1)\sin(\boldsymbol{x}(3)) \\ \boldsymbol{u}(2) \end{bmatrix} + W$ $\tau = 0.3, W = \{\boldsymbol{0}\}$	$[0, 5]^2 \times$ $[-1.6, 1.6]$	$[-1, 1]^2$	$\begin{bmatrix} 0.2 \\ 0.2 \\ 0.2 \end{bmatrix}$	$\begin{bmatrix} 0.3 \\ 0.3 \end{bmatrix}$	$[0, 10]^2 \times$ $[-\pi, \pi]$	$[-1.5, 1.5] \times$ $[-1, 1]$	$\begin{bmatrix} 0.1 \\ 0.1 \\ 0.1 \end{bmatrix}$	$\begin{bmatrix} 0.23 \\ 0.23 \end{bmatrix}$
4D car $(\boldsymbol{x}(1), \boldsymbol{x}(2))$- position $\boldsymbol{x}(3)$- angle $\boldsymbol{x}(4)$- speed $\boldsymbol{u}(1)$- turn rate $\boldsymbol{u}(2)$- acceleration control	$\begin{bmatrix} \dot{\boldsymbol{x}}(1) \\ \dot{\boldsymbol{x}}(2) \\ \dot{\boldsymbol{x}}(3) \\ \dot{\boldsymbol{x}}(4) \end{bmatrix} \in \begin{bmatrix} \boldsymbol{x}(4)\cos(\boldsymbol{x}(3)) \\ \boldsymbol{x}(4)\sin(\boldsymbol{x}(3)) \\ \boldsymbol{u}(1) \\ \boldsymbol{u}(2) \end{bmatrix} + W$ $\tau = 0.5, W = \{\boldsymbol{0}\}$	$[0, 5]^2 \times$ $[-1.6, 1.6] \times$ $[-1, 1]$	$[-1, 1]^2$	$\begin{bmatrix} 0.2 \\ 0.2 \\ 0.2 \\ 0.2 \end{bmatrix}$	$\begin{bmatrix} 0.3 \\ 0.3 \end{bmatrix}$	$[0, 10]^2 \times$ $[-\pi, \pi] \times$ $[-1, 1]$	$[-2, 2]^2$	$\begin{bmatrix} 0.2 \\ 0.2 \\ 0.2 \\ 0.2 \end{bmatrix}$	$\begin{bmatrix} 0.2 \\ 0.2 \end{bmatrix}$
5D car $(\boldsymbol{x}(1), \boldsymbol{x}(2))$- position $\boldsymbol{x}(3)$- angle $\boldsymbol{x}(4)$- speed $\boldsymbol{x}(5)$- turn rate $\boldsymbol{u}(1)$- acceleration $\boldsymbol{u}(2)$- angular acceleration	$\begin{bmatrix} \dot{\boldsymbol{x}}(1) \\ \dot{\boldsymbol{x}}(2) \\ \dot{\boldsymbol{x}}(3) \\ \dot{\boldsymbol{x}}(4) \\ \dot{\boldsymbol{x}}(5) \end{bmatrix} \in \begin{bmatrix} \boldsymbol{x}(4)\cos(\boldsymbol{x}(3)) \\ \boldsymbol{x}(4)\sin(\boldsymbol{x}(3)) \\ \boldsymbol{x}(5) \\ \boldsymbol{u}(1) \\ \boldsymbol{u}(2) \end{bmatrix} + W$ $\tau = 0.5, W = \{\boldsymbol{0}\}$	$[0, 5]^2 \times$ $[-1.6, 1.6] \times$ $[-1, 1] \times$ $[-1, 1]$	$[-1, 1]^2$	$\begin{bmatrix} 0.2 \\ 0.2 \\ 0.2 \\ 0.2 \\ 0.2 \end{bmatrix}$	$\begin{bmatrix} 0.3 \\ 0.3 \end{bmatrix}$	$[0, 10]^2 \times$ $[-\pi, \pi] \times$ $[-1, 1] \times$ $[-1, 1]$	$[-2, 2]^2$	$\begin{bmatrix} 0.1 \\ 0.1 \\ 0.1 \\ 0.1 \\ 0.1 \end{bmatrix}$	$\begin{bmatrix} 0.2 \\ 0.2 \end{bmatrix}$
Inverted pendulum $\boldsymbol{x}(1)$- angle $\boldsymbol{x}(2)$- angular velocity $\boldsymbol{u}(1)$- torque	$\begin{bmatrix} \dot{\boldsymbol{x}}(1) \\ \dot{\boldsymbol{x}}(2) \end{bmatrix} \in \begin{bmatrix} \boldsymbol{x}(2) \\ \frac{g}{L}\sin(\boldsymbol{x}(1)) + \frac{1}{mL^2}\boldsymbol{u}(1) \end{bmatrix} + W$ $L = g, m = \frac{8}{g^2}$ $\tau = 0.05, W = \{\boldsymbol{0}\}$	$[-\pi, \pi] \times$ $[-2, 2]$	$[-1, 1]$	$\begin{bmatrix} 0.2 \\ 0.2 \end{bmatrix}$	$\begin{bmatrix} 0.3 \end{bmatrix}$	$[-\pi, \pi] \times$ $[-2, 2]$	$[-1, 1]$	$\begin{bmatrix} 0.1 \\ 0.1 \end{bmatrix}$	$\begin{bmatrix} 0.3 \end{bmatrix}$
TORA $\boldsymbol{x}(1)$ $\boldsymbol{x}(2)$ $\boldsymbol{x}(3)$- angle $\boldsymbol{x}(4)$- angular velocity $\boldsymbol{u}(1)$- torque	$\begin{bmatrix} \dot{\boldsymbol{x}}(1) \\ \dot{\boldsymbol{x}}(2) \\ \dot{\boldsymbol{x}}(3) \\ \dot{\boldsymbol{x}}(4) \end{bmatrix} \in \begin{bmatrix} \boldsymbol{x}(2) \\ -\boldsymbol{x}(1) + 0.1\sin(\boldsymbol{x}(3)) \\ \boldsymbol{x}(4) \\ \boldsymbol{u}(1) \end{bmatrix} + W$ $\tau = 0.5, W = \{\boldsymbol{0}\}$	$[-2, 2]^4$	$[-1, 1]$	$\begin{bmatrix} 0.2 \\ 0.2 \\ 0.2 \\ 0.2 \end{bmatrix}$	$\begin{bmatrix} 0.3 \end{bmatrix}$	$[-2, 2]^4$	$[-1, 1]$	$\begin{bmatrix} 0.1 \\ 0.1 \\ 0.1 \\ 0.1 \end{bmatrix}$	$\begin{bmatrix} 0.3 \end{bmatrix}$

while providing confidence on the correctness of T_F. Our experiments were performed on a cluster with Intel Xeon E7-8857 v2 CPUs (32 cores in total) at 3GHz, with 100GB of RAM. For training neural networks, we did not use a distributed implementation as we found that distributing the process across GPUs actually decelerates the process. However, for the rest of our compression and synthesis algorithms, we used a distributed implementation.

Synthesis. We considered the ℓ_∞ ball centered at $(4, 4)$ with the radius 0.8 over the Euclidean plane as the target set for the multi-dimensional car examples, $[-0.5, 0.5] \times [-1, 1]$ for the inverted pendulum example, and $[-1, 1]^4$ for the TORA example. To evaluate our corrected neural method described in Section 3.1, we set the list of neuron numbers in different layers as $(n + m, 20, 40, 30, 2n)$, select the activation functions to be hyperbolic tangent, and set the learning rate to be $\lambda = 0.001$. As discussed in Section 3.2, the corrected neural representations for finite abstractions can also be constructed by solving a classification problem. To evaluate this method, we set the list of neuron numbers in different layers for both $\mathcal{N}_F$ and $\mathcal{N}_B$ as $(n + m, 40, 160, 160, 160, 160, 160, 160, 160, 160, 500, 800, 2\sum_{i=1}^{n} |\bar{X}(i)|)$, select the activation functions to be ReLU, and set the learning rate to be $\lambda = 0.0001$. We used stochastic gradient descent method with the corresponding learning rate for training the neural networks [33]. Tables 2 and 3 illustrate the synthesis results related to our experiments for finite abstractions, using the regression-based and classification-based methods, respectively. Although we used the same neural network structure for all the examples, soundness errors take small values that are bounded by 3.44×10^{-2} as the maximum of $\boldsymbol{e}_F$ and $\boldsymbol{e}_B$ in the regression-based method, and by 1.27×10^{-1} as the maximum of err_F and err_B in the classification-based method. Moreover, memory requirement of our proposed regression-based and classification-based methods at higher dimensions remains almost constant while the size of the transition system increases exponentially (see the illustration shown in Figure 9 (Left) for the multi-dimensional car case studies). Further, we notice that the

Table 2. The Results of Regression-based Controller Synthesis for Finite Abstractions

Case study	$|\bar{X}| \times |\bar{U}|$	e_F	e_B	d_F	d_B	$\mathcal{M}_T$ (kB)	$\mathcal{M}_F + \mathcal{M}_B$ (kB)	$\mathcal{T}_c$ (min)	$\mathcal{T}_s$ (min)
2D car	810000	$\begin{bmatrix}1.02\times10^{-2}\\1.58\times10^{-2}\end{bmatrix}$	$\begin{bmatrix}2.81\times10^{-2}\\1.17\times10^{-2}\end{bmatrix}$	6.81×10^{-1}	9.64×10^{-1}	7.76×10^{4}	488	68.58	8.55
3D car	451584	$\begin{bmatrix}2.05\times10^{-2}\\2.19\times10^{-2}\\2.26\times10^{-2}\end{bmatrix}$	$\begin{bmatrix}2.48\times10^{-2}\\1.76\times10^{-2}\\2.32\times10^{-2}\end{bmatrix}$	7.11×10^{-1}	7.85×10^{-1}	1.35×10^{5}	488	65.46	14.50
4D car	4967424	$\begin{bmatrix}1.71\times10^{-2}\\2.40\times10^{-2}\\1.62\times10^{-2}\\1.96\times10^{-2}\end{bmatrix}$	$\begin{bmatrix}2.05\times10^{-2}\\1.54\times10^{-2}\\1.35\times10^{-2}\\1.25\times10^{-2}\end{bmatrix}$	4.24×10^{-1}	2.87×10^{-1}	5.58×10^{6}	488	446.23	20.55
5D car	30735936	$\begin{bmatrix}1.41\times10^{-2}\\1.18\times10^{-2}\\1.97\times10^{-2}\\2.22\times10^{-2}\\1.93\times10^{-2}\end{bmatrix}$	$\begin{bmatrix}2.11\times10^{-2}\\1.79\times10^{-2}\\1.13\times10^{-2}\\1.65\times10^{-2}\\2.45\times10^{-2}\end{bmatrix}$	5.34×10^{-1}	4.25×10^{-1}	$3.64\times10^{8}(OOM)$	488	3025.14	312.15
Inverted pendulum	17360	$\begin{bmatrix}2.53\times10^{-2}\\3.44\times10^{-2}\end{bmatrix}$	$\begin{bmatrix}2.31\times10^{-2}\\2.97\times10^{-2}\end{bmatrix}$	6.50×10^{-1}	5.61×10^{-1}	2.27×10^{4}	488	68.58	4.18
TORA	1433531	$\begin{bmatrix}2.53\times10^{-2}\\2.67\times10^{-2}\\2.39\times10^{-2}\\2.24\times10^{-2}\end{bmatrix}$	$\begin{bmatrix}2.21\times10^{-2}\\2.57\times10^{-2}\\1.88\times10^{-2}\\3.03\times10^{-2}\end{bmatrix}$	4.34×10^{-1}	4.15×10^{-1}	1.57×10^{7}	488	241.48	166.16

$\bar{X} \times \bar{U}$ indicates the number of discrete state-input pairs, e_F, e_B denote the soundness errors, respectively, for the forward and backward representations, computed using Equation (6), d_F and d_B give the graph mismatch rates for the forward and backward dynamics using using Equation (4), $\mathcal{M}_T$ gives the memory needed to store the original transition system in kB, $\mathcal{M}_F + \mathcal{M}_B$ denotes the memory taken by the representing neural networks for the forward and backward dynamics in kB, $\mathcal{T}_c$ denotes the total execution time for computing the compressed representations in minutes. and $\mathcal{T}_s$ denotes the total execution time for synthesizing the controller in minutes.

Table 3. The Results of Classifier-based Controller Synthesis for Finite Abstractions

Case study	$|\bar{X}| \times |\bar{U}|$	err_F	err_B	d_F	d_B	$\mathcal{M}_T$ (kB)	$\mathcal{M}_F + \mathcal{M}_B$ (kB)	$\mathcal{T}_c$ (min)	$\mathcal{T}_s$ (min)
2D car	810000	2.75×10^{-2}	3.27×10^{-2}	2.65×10^{-2}	2.93×10^{-2}	7.76×10^{4}	1.33×10^{4}	68.58	10.71
3D car	451584	2.71×10^{-4}	2.21×10^{-6}	3.71×10^{-5}	9.47×10^{-7}	1.35×10^{5}	1.91×10^{4}	50.74	12.11
4D car	4967424	6.24×10^{-4}	0	2.84×10^{-4}	0	5.58×10^{6}	2.37×10^{4}	565.13	24.58
5D car	30735936	3.41×10^{-5}	5.33×10^{-8}	3.21×10^{-5}	2.19×10^{-8}	$3.64\times10^{8}(OOM)$	3.27×10^{4}	3421.21	215.88
Inverted pendulum	17360	6.03×10^{-2}	5.85×10^{-2}	0	0	2.27×10^{4}	2.08×10^{4}	8.21	8.33
TORA	1433531	1.27×10^{-1}	1.26×10^{-1}	1.55×10^{-1}	1.48×10^{-1}	1.57×10^{7}	2.38×10^{4}	234.87	159.75

$\bar{X} \times \bar{U}$ indicates the number of discrete state-input pairs, err_F, err_B denote the soundness errors, respectively, for the forward and backward representations, computed using Equation (14), d_F and d_B give the graph mismatch rates for the forward and backward dynamics, $\mathcal{M}_T$ gives the memory needed to store the original transition system in kB, $\mathcal{M}_F + \mathcal{M}_B$ denotes the memory taken by the representing neural networks for the forward and backward dynamics in kB, $\mathcal{T}_c$ denotes the total execution time for computing the compressed representations in minutes. and $\mathcal{T}_s$ denotes the total execution time for synthesizing the controller in minutes.

regression-based method results in higher mismatch rates d_F and d_B compared to the classification-based method: on average, 5.87×10^{-1} versus 3.03×10^{-2} for d_F, and 6.15×10^{-1} versus 2.96×10^{-2} for d_B (see the illustration shown in Figure 9 (Right) for the multi-dimensional car case studies). Therefore, using the classification-based method, while being sound, produces a smaller graph, which is less restrictive for the synthesis purpose. Most importantly, memory requirement using both our approaches is way less than the memory needed to store the original (forward) transition system ($\mathcal{M}_F + \mathcal{M}_B << \mathcal{M}_T$). Regression-based method reduces the memory requirements by a factor of 1.31×10^{5} and up to 7.54×10^{5}. However, the classification-based method reduces the memory requirements by a factor of 2.01×10^{3} and up to 1.12×10^{4}. This shows that the regression-based method requires less memory compared to the classification-based method.

Deployment. Table 4 lists our experimental results for compressing the symbolic controllers. For $\mathcal{N}_C$, we set the list of neuron numbers in different layers for both $\mathcal{N}_F$ and $\mathcal{N}_B$ as
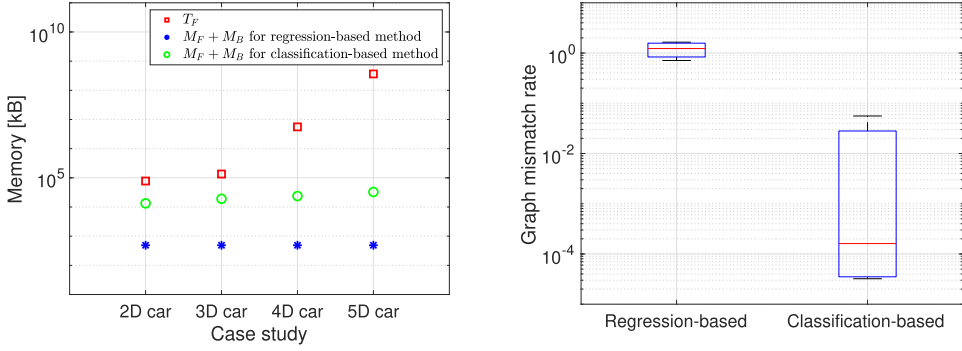
Fig. 9. Left: Memory requirement of different methods for storing transition systems of multi-dimensional cars (cf. Table 1) in logarithmic scale. Right: Distribution of total graph mismatch rate ($d_F + d_B$) for our proposed methods in logarithmic scale.

Table 4. The Results of Controller Compression

Case study	$|C|$	err_C	$\mathcal{M}_C$ (kB)	$\mathcal{M}_{\hat{C}}(kB)$	$\mathcal{T}$ (min)
2D car	2.15×10^6	1.85×10^{-5}	2.75×10^5	1.21×10^3	6.31
3D car	2.87×10^6	2.16×10^{-3}	4.65×10^5	1.05×10^3	19.14
4D car	9.35×10^7	3.63×10^{-2}	2.24×10^6	1.35×10^3	39.48
5D car	1.69×10^9	4.51×10^{-3}	4.71×10^7	1.48×10^3	201.86
Inverted pendulum	8.16×10^5	1.08×10^{-3}	7.83×10^4	8.92×10^2	7.51
TORA	4.78×10^7	3.78×10^{-4}	7.65×10^6	8.92×10^2	113.97

$|C|$ gives the number of state-input pairs in the original controller, err_C denotes the portion of the states at which the representing neural network produces non-valid control inputs computed using Equation (22), $\mathcal{M}_C$ gives the memory needed to store the original controller in kB, $\mathcal{M}_{\hat{C}}$ denotes the memory taken by the representing neural network in kB, and $\mathcal{T}$ denotes the total execution time for our implementation in minutes.

(n, 20, 80, 80, 80, 80, 80, 160, $|\bar{U}|$), select the activation functions to be rectified linear unit (ReLU), and set the learning rate to be $\lambda = 0.0001$. It can be noticed that err_C is very small for all the examples. Therefore, we only need to store a very small portion of C in addition to $\mathcal{N}_C$. As it can be observed in Table 4, our method has been successful in computing representations which are very accurate and compact-in-size ($\mathcal{M}_{\hat{C}} << \mathcal{M}_C$).

Parametrization. Our approach requires selecting the hyperparameters of the training process and choosing the structure of the neural networks. We have performed several experiments to select the hyperparameters of the training (e.g., the learning rate, epoch number, and batch size). Regarding the structure of the neural networks, we have explored different choices such as the type of the activation functions (hyperbolic tangent, ReLU, etc.), number of neurons per layer, and the depth. Increasing the complexity of the neural network, by increasing the number of neurons per layer or depth, leads to a better performance. Note that the neural networks employed in our setting are not supposed to make any generalization over unseen data. Therefore, our approach does not suffer from over-parametrization of the neural networks. We have demonstrated this in Figure 10 by providing the error as a function of the depth of the neural representation for the 3D car example. The error always decreases by increasing the depth of the neural representation. Therefore, the structure of the neural representations can be selected for having an acceptable accuracy within a given time bound for the training process.
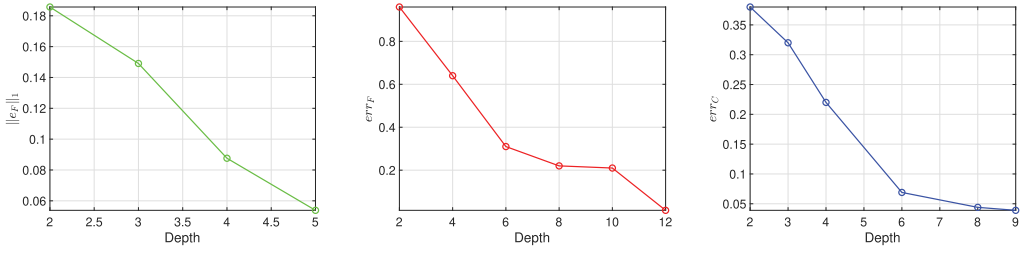
Fig. 10. Demonstrating the effect of increasing the depth of the neural representation on the norm of the soundness error e_F (cf. Equation (6)) for regression-based controller synthesis (Left), the soundness error err_F (cf. Equation (14)) for classification-based controller synthesis (Middle), and the misclassification rate (cf. Equation (22)) for deployment (Right). The experiments are performed on the 3D car example.

6 DISCUSSION AND CONCLUSIONS

In this paper, we considered abstraction-based methods for controller synthesis to satisfy high-level temporal requirements. We addressed the (exponentially large) memory requirements of these methods in both synthesis and deployment. Using the expressive power of neural networks, we proposed memory-efficient methods to compute compressed representations for both forward and backward dynamics of the system. With focus on reach-avoid specifications, we showed how to perform synthesis using corrected neural representations of the system. We also proposed a novel formulation for finding compact corrected neural representations of the controller to reduce the memory requirements in deploying the controller. Finally, we evaluated our approach on multiple case studies, showing reduction in memory requirements by orders of magnitude, while providing formal correctness guarantees.

Extension to more general specifications. Our approach is based on computing an under-approximation of Pre and over-approximation of Post operators. Therefore, it can be applied to any synthesis problem whose solution is characterized based on these operators. This means our approach can be applied to control synthesis for other linear temporal logic specifications including safety, Büchi, and Rabin objectives.

Reusability of the computed representations. Our approach computes the corrected neural representations that is sound on the whole state space. These representations can be used for any other problem defined over the same finite abstraction.

Application to systems with known analytical model. Our approach is efficient in providing compact representations for a given finite abstraction at the cost of increasing the off-line computational time. This is regardless of constructing the finite abstraction using model-based methods or (correct) data-driven methods. Model-based on-the-fly synthesis methods will utilize numerical solutions of differential equations when the analytical model of the system is known with available bounds on the continuity properties of the system. These methods may perform better in case solving the corresponding differential equations is faster than making a forward pass through the neural representation.

Comparison with a baseline method. We have demonstrated the effectiveness of our method on a number of case studies in compressing finite transition systems and controllers which are stored in the form of look-up tables. In the introduction and related work sections of our paper, we have discussed why other methods cannot be used to solve our problem. In below, we have listed our main arguments.

- While transition systems and controllers can be encoded using BDDs instead of look-up tables, the memory blow-up problem still exists for systems of higher dimensions. However, using our technique, we empirically show that the size of the computed representations is not necessarily affected by size of the original mapping. See for example Figure 9 (Left), wherein the memory required by the trained compressed representation stays at 488 kB, despite the fact that the required memory by the original transition system has increased by a factor of 5000.

- Also, our synthesis setting is different from the one considered in references [16, 28, 34], wherein memory-efficient synthesis methods are proposed based on a (compact) analytical description of the nominal dynamics of the system and its growth bound. We consider the case wherein the input is a huge finite transition system which can also be learned from simulations.

- Finally, while the control determinization and compression schemes proposed in [16, 44] are based on the BDD and ADD encodings of the controller, the only methodologically that is in a similar spirit as our deployment approach is the symbolic regression of [44]. As mentioned by the authors of [44], their regression-based method is not able to represent the original controller with an acceptable accuracy. Our superior performance is mainly because of our classification-based formulation, as opposed to a regression-based formulation.

Utilizing invertible neural networks. Our method requires training two different neural networks associated with the forward and backward dynamics. A possible future research direction would be to use invertible neural networks instead of training two separate neural networks. However, given the specific application and inherent differences between our approach and the successful experiences with invertible neural networks, it is currently not obvious to us that the same performance would be accessible.

REFERENCES

[1] Rajeev Alur. 2015. *Principles of Cyber-physical Systems*. MIT press.

[2] C. Baier and J.-P. Katoen. 2008. *Principles of Model Checking*. MIT press.

[3] Calin Belta, Boyan Yordanov, and Ebru Aydin Gol. 2017. *Formal Methods for Discrete-time Dynamical Systems*. Vol. 15. Springer.

[4] D. P. Bertsekas. 1999. *Nonlinear Programming*. Athena Scientific.

[5] Justin Boyan and Andrew Moore. 1994. Generalization in reinforcement learning: Safely approximating the value function. In *Advances in Neural Information Processing Systems*, Vol. 7. MIT Press.

[6] Bryant. 1986. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Comput.* C-35, 8 (1986), 677–691.

[7] Ji Chen, Salar Moarref, and Hadas Kress-Gazit. 2018. Verifiable control of robotic swarm from high-level specifications. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2018, Stockholm, Sweden, July 10–15, 2018*. International Foundation for Autonomous Agents and Multiagent Systems Richland, SC, USA / ACM, 568–576.

[8] Alex Devonport, Adnane Saoud, and Murat Arcak. 2021. Symbolic abstractions from data: A PAC learning approach. In *2021 60th IEEE Conference on Decision and Control (CDC'21)*. IEEE, Austin, TX, USA, 599–604.

[9] Souradeep Dutta, Xin Chen, and Sriram Sankaranarayanan. 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*. ACM, Montreal Quebec Canada.

[10] Antoine Girard. 2012. Low-complexity switching controllers for safety using symbolic models. *IFAC Proceedings Volumes* 45, 9 (2012), 82–87. 4th IFAC Conference on Analysis and Design of Hybrid Systems.

[11] Michael Hertneck, Johannes Köhler, Sebastian Trimpe, and Frank Allgöwer. 2018. Learning an approximate model predictive controller with guarantees. *IEEE Control. Syst. Lett.* 2, 3 (2018), 543–548.

[12] Kyle Hsu, Rupak Majumdar, Kaushik Mallik, and Anne-Kathrin Schmuck. 2018. Multi-layered abstraction-based controller synthesis for continuous-time systems. In *Proceedings of the 21st International Conference on Hybrid Systems: Computation and Control (part of CPS Week)*. ACM, Porto Portugal.

[13] Chao Huang, Jiameng Fan, Xin Chen, Wenchao Li, and Qi Zhu. 2022. POLAR: A polynomial arithmetic framework for verifying neural-network controlled systems. In *Automated Technology for Verification and Analysis*. Springer International Publishing, 414–430.

[14] Chao Huang, Jiameng Fan, Wenchao Li, Xin Chen, and Qi Zhu. 2019. ReachNN. *ACM Transactions on Embedded Computing Systems* 18, 5s (Oct. 2019), 1–22.

[15] Radoslav Ivanov, James Weimer, Rajeev Alur, George J. Pappas, and Insup Lee. 2019. Verisig. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*. ACM, Montreal, QC, Canada.

[16] Elena Ivanova, Adnane Saoud, and Antoine Girard. 2022. Lazy controller synthesis for monotone transition systems and directed safety specifications. *Autom.* 135 (2022), 109993.

[17] M. Jankovic, D. Fontaine, and P. V. Kokotovic. 1996. TORA example: Cascade- and passivity-based control designs. *IEEE Transactions on Control Systems Technology* 4, 3 (1996), 292–297.

[18] Kyle D. Julian, Mykel J. Kochenderfer, and Michael P. Owen. 2019. Deep neural network compression for aircraft collision avoidance systems. *Journal of Guidance, Control, and Dynamics* 42, 3 (March 2019), 598–608.

[19] 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'16)*. Sacramento, CA, USA, 1–10.

[20] Milad Kazemi, Rupak Majumdar, Mahmoud Salamati, Sadegh Soudjani, and Ben Wooding. 2022. Data-Driven Abstraction-Based Control Synthesis. (2022).

[21] Mahmoud Khaled, Eric S. Kim, Murat Arcak, and Majid Zamani. 2019. Synthesis of symbolic controllers: A parallelized and sparsity-aware approach. In *Tools and Algorithms for the Construction and Analysis of Systems - 25th International Conference, TACAS 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6–11, 2019, Proceedings, Part II (Lecture Notes in Computer Science)*, Vol. 11428. Springer, 265–281.

[22] Mahmoud Khaled and Majid Zamani. 2019. pFaces: An acceleration ecosystem for symbolic control. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control, HSCC 2019, Montreal, QC, Canada, April 16–18, 2019*. ACM, 252–257.

[23] Mykel J. Kochenderfer and Nicholas Monath. 2013. Compression of optimal value functions for Markov decision processes. In *2013 Data Compression Conference*. IEEE, Snowbird, UT, USA, 501–501.

[24] Marta Kwiatkowska, Gethin Norman, and David Parker. 2005. Probabilistic model checking in practice: Case studies with PRISM. *ACM SIGMETRICS Performance Evaluation Review* 32, 4 (2005), 16–21.

[25] Abolfazl Lavaei, Mahmoud Khaled, Sadegh Soudjani, and Majid Zamani. 2020. AMYTISS: Parallelized automated controller synthesis for large-scale stochastic systems. In *Computer Aided Verification: 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21–24, 2020, Proceedings, Part II 32*. Springer, 461–474.

[26] Edward Ashford Lee and Sanjit Arunkumar Seshia. 2016. *Introduction to Embedded Systems: A Cyber-physical Systems Approach*. MIT Press.

[27] Diego Manzanas Lopez, Matthias Althoff, Luis Benet, Xin Chen, Jiameng Fan, Marcelo Forets, Chao Huang, Taylor T. Johnson, Tobias Ladner, Wenchao Li, Christian Schilling, and Qi Zhu. 2022. ARCH-COMP22 category report: Artificial intelligence and neural network control systems (AINNCS) for continuous and hybrid systems plants. In *Proceedings of 9th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH22'22) (EPiC Series in Computing)*, Vol. 90. EasyChair, 142–184.

[28] E. Macoveiciuc and G. Reissig. 2022. On-the-fly symbolic synthesis with memory reduction guarantees. *IEEE Trans. Automat. Control* 68, 4 (2022), 1–8.

[29] Rupak Majumdar, Necmiye Ozay, and Anne-Kathrin Schmuck. 2020. On abstraction-based controller design with output feedback. In *HSCC'20: 23rd ACM International Conference on Hybrid Systems: Computation and Control, April 21–24, 2020*. ACM, 15:1–15:11.

[30] Anas Makdesi, Antoine Girard, and Laurent Fribourg. 2021. Efficient data-driven abstraction of monotone systems with disturbances. *IFAC-PapersOnLine* 54, 5 (2021), 49–54. 7th IFAC Conference on Analysis and Design of Hybrid Systems ADHS 2021.

[31] G. Reissig, A. Weber, and M. Rungger. 2016. Feedback refinement relations for the synthesis of symbolic controllers. *IEEE TAC* 62, 4 (2016), 1781–1796.

[32] Pritam Roy, Paulo Tabuada, and Rupak Majumdar. 2011. Pessoa 2.0. In *Proceedings of the 14th International Conference on Hybrid Systems: Computation and Control*. ACM, Chicago IL USA.

[33] Sebastian Ruder. 2016. An overview of gradient descent optimization algorithms. *ArXiv* abs/1609.04747 (2016).

[34] Matthias Rungger and Olaf Stursberg. 2012. On-the-fly model abstraction for controller synthesis. In *2012 American Control Conference (ACC'12)*. IEEE, Montreal, QC, Canada, 2645–2650.

[35] M. Rungger and M. Zamani. 2016. SCOTS: A tool for the synthesis of symbolic controllers. In *HSCC*. ACM, Vienna, Austria, 99–104.

[36] Mahmoud Salamati, Rocco Salvia, Eva Darulova, Sadegh Soudjani, and Rupak Majumdar. 2019. Memory-efficient mixed-precision implementations for robust explicit model predictive control. *ACM Trans. Embed. Comput. Syst.* 18, 5s (2019), 100:1–100:19.

[37] Stanly Samuel, Kaushik Mallik, Anne-Kathrin Schmuck, and Daniel Neider. 2020. Resilient abstraction-based controller design. In *HSCC'20: 23rd ACM International Conference on Hybrid Systems: Computation and Control, Sydney, New South Wales, Australia, April 21–24, 2020.* ACM, Jeju, South Korea, 33:1–33:2.

[38] Sumeet Singh, Mo Chen, Sylvia L. Herbert, Claire J. Tomlin, and Marco Pavone. 2020. Robust tracking with model mismatch for fast and safe planning: An SOS optimization approach. In *Algorithmic Foundations of Robotics XIII.* Springer International Publishing, Cham, 545–564.

[39] Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction* (2nd ed.).

[40] Paulo Tabuada. 2009. *Verification and Control of Hybrid Systems: A Symbolic Approach* (1st ed.). Springer Publishing Company, Incorporated.

[41] Hoang-Dung Tran, Feiyang Cai, Manzanas Lopez Diego, Patrick Musau, Taylor T. Johnson, and Xenofon Koutsoukos. 2019. Safety verification of cyber-physical systems with reinforcement learning control. *ACM Transactions on Embedded Computing Systems* 18, 5s (Oct. 2019), 1–22.

[42] Hoang-Dung Tran, Xiaodong Yang, Diego Manzanas Lopez, Patrick Musau, Luan Viet Nguyen, Weiming Xiang, Stanley Bak, and Taylor T. Johnson. 2020. NNV: The neural network verification tool for deep neural networks and learning-enabled cyber-physical systems. In *Computer Aided Verification.* Springer International Publishing, Cham, 3–17.

[43] J. N. Tsitsiklis and B. Van Roy. 1997. An analysis of temporal-difference learning with function approximation. *IEEE Trans. Automat. Control* 42, 5 (1997), 674–690.

[44] Ivan S. Zapreev, Cees Verdier, and Manuel Mazo. 2018. Optimal symbolic controllers determinization for BDD storage. *IFAC-PapersOnLine* 51, 16 (2018), 1–6. 6th IFAC Conference on Analysis and Design of Hybrid Systems ADHS 2018.