2023 62nd IEEE Conference on Decision and Control (CDC)
December 13-15, 2023. Marina Bay Sands, Singapore

Exact and Cost-Effective Automated Transformation of Neural Network Controllers to Decision Tree Controllers

Kevin Chang[1], Nathan Dahlin[2], Rahul Jain[1] and Pierluigi Nuzzo[1]

***Abstract*— Over the past decade, neural network (NN)-based controllers have demonstrated remarkable efficacy in a variety of decision-making tasks. However, their black-box nature and the risk of unexpected behaviors and surprising results pose a challenge to their deployment in real-world systems with strong guarantees of correctness and safety. We address these limitations by investigating the transformation of NN-based controllers into equivalent soft decision tree (SDT)-based controllers and its impact on verifiability. Differently from previous approaches, we focus on discrete-output NN controllers including rectified linear unit (ReLU) activation functions as well as argmax operations. We then devise an exact but cost-effective transformation algorithm, in that it can automatically prune redundant branches. We evaluate our approach using two benchmarks from the OpenAI Gym environment. Our results indicate that the SDT transformation can benefit formal verification, showing runtime improvements of up to $21\times$ and $2\times$ for MountainCar-v0 and CartPole-v1, respectively.**

I. Introduction

Over the last decade, neural network (NN)-based techniques have exhibited outstanding efficacy in a variety of decision-making tasks, surpassing human-level performance in some of the most challenging control problems. They even lead the leaderboard in almost all benchmark problems in control and robotics [1]–[4]. However, their deployment in safety-critical applications, such as autonomous driving and flight control, raises concerns [5], [6] due to the black-box nature of NNs and the risk of unexpected behaviors.

To overcome the limitations of NN-based controllers, researchers have proposed distillation [7], which transfers the learned knowledge and behaviors of NN controllers to alternative models, such as decision trees, which are easier to interpret. In fact, the efficacy of NN controllers versus other models is not necessarily due to their richer representative capacity, but rather to the many regularization techniques available to facilitate training [8], [9]. By compressing NN controllers into simpler and more compact models, distillation can also facilitate formal verification. However, the distilled models typically fall short of the real-time performance of their full counterparts. In fact, the identification of effective metrics to characterize the approximation quality of a distilled model versus the original NN is itself an open problem.

In this paper, we focus instead on the *exact systematic transformation* of NN-based controllers into equivalent soft decision tree (SDT)-based controllers and the empirical evaluation of the impact of this transformation on the verifiability of the controllers. The equivalent SDT models can be used for verification, while the NN models are used at runtime. Moreover, unlike prior work, we consider discrete-output NN controllers with rectified linear unit (ReLU) activation functions and argmax operations. These discrete-action NNs are particularly challenging from a verification standpoint, in that they tend to amplify the approximation errors generated by reachability analysis of the closed-loop control system. Our contributions can be stated as follows:

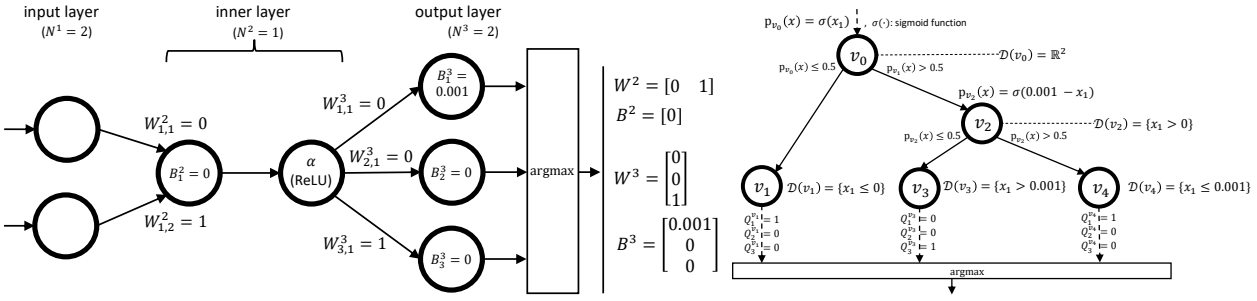
- We first prove that any discrete output argmax-based NN controller has an *equivalent* SDT. This is done by presenting a constructive procedure to transform any NN controller into an equivalent SDT controller that has the same properties.
- We show that our constructive procedure for creating an equivalent SDT controller can also be computationally practical, in that the number of nodes in the SDT scales polynomially with the maximum width of the hidden layers in the NN. To the best of our knowledge, this is the first such computationally practical transformation algorithm.
- We empirically validate the computational efficiency of formally verifying the SDT controller over the original NN controller in two benchmark OpenAI Gym environments [10], showing that verifying the SDT controller can be 21 times faster in the MountainCar-v0 environment and twice as fast in the CartPole-v1 environment.

Our results suggest that SDT transformation can be used to accelerate the verification of NN controllers in feedback control loops, with potential impact on applications where performance guarantees are critical but deep learning methods are the primary choice for control design.

Related Work. Distillation [7] has been used to transfer the knowledge and behavior of NN-based controllers to other models in an approximate, e.g., data-driven, or exact manner. *Approximate distillation* can be performed by training shallow NNs to mimic the behavior of state-of-the-art NNs using a teacher-student paradigm [8]. The distillation of SDTs from expert NNs was shown to lead to better performance than direct training of SDTs [7]. Furthermore, distillation has demonstrated success in reinforcement learning (RL) problems, where the DAGGER algorithm is used to transfer knowledge from Q-value NN models via simulation episodes [11]. Finally, distilling to SDTs has also been suggested to interpret the internal workings of black-box NNs [12].

In contrast, *exact distillation* has been proposed to transform feedforward NNs into simpler models while preserving equivalence [13]–[15]. Locally constant networks [13] and

[1]K. Chang, R. Jain, and P. Nuzzo are with the Ming Hsieh Department of Electrical and Computer Engineering, University of Southern California, Los Angeles, USA, {kcchang, rahul.jain, nuzzo}@usc.edu.
[2]N. Dahlin was with the Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Urbana, USA. He is now with the Department of Electrical Engineering, State University of New York at Albany, Albany, USA, ndahlin@albany.edu.
This research was supported in part by the NSF under Awards 1846524 and 2139982, the ONR under Award N00014-20-1-2258, DARPA under Award HR00112010003, the Okawa Research Grant, and Siemens.



(a) Reference NN.

(b) Equivalent SDT for the NN in Fig. 1a.

Fig. 1: Example NN-to-SDT transformation.

linear networks [14] have been introduced as intermediate representations to establish the equivalence between NNs and SDTs whose worst-case size scales exponentially in the maximum width of the NN hidden layers. Nguyen et al. [15] proposed transforming NNs with ReLUs into decision trees and then compressing the trees via a learning-based approach. As in previous approaches, our algorithm preserves equivalence with the SDTs. However, it also guarantees, without the need for compression, that the size of the tree scales polynomially in the width of the maximum hidden layer of the NN. Finally, we also provide quantitative evidence about the impact of the proposed transformations on the verifiability of the controllers.

II. Preliminaries

We review key definitions for both neural networks and soft decision trees using Fig. 1 as an illustrative example. Throughout the paper, we use $A_{i,j}$ to denote the element in the i-th row and j-th column of matrix A, and a_i or $(a)_i$ to denote the i-th element of vector a.

1) Neural Networks: Let L denote the total number of layers in a NN, N^l denote the *width*, or number of neurons in the l-th layer, and define $N := (N^1, \ldots, N^L)$. In Fig. 1a, we have $L = 3$, with a two-neuron input layer ($N^1 = 2$), a single-neuron hidden layer ($N^2 = 1$), and a three-neuron output layer ($N^3 = 3$), so that $N = (2, 1, 3)$. The nodes in layer l, with $l > 1$, are fully connected to the previous layer. Edges and nodes are associated with weights and biases, denoted by W^l and B^l, respectively, for $l \geq 2$. The output at each neuron is determined by its inputs using a *feedforward function*, defined below, and then passed through a rectified linear unit (ReLU) *activation function*.

Definition 1: (NN Layer Feedforward Function). Given input $x \in \mathbb{R}^{N^{l-1}}$, the feedforward function $f^l_F : \mathbb{R}^{N^{l-1}} \to \mathbb{R}^{N^l}$ maps x to the output of layer $l \geq 2$, where the output of the i-th neuron in layer l is defined as

$$(f^l_F(x))_i := \sum_{i=1}^{N^{l-1}} W^l_{i,j} x_j + B^l_i.$$

We also recall the definition of *characteristic function*. Let α be the element-wise ReLU activation function, i.e., $\alpha(x)_i = \max\{x_i, 0\}$.

Definition 2: (NN Layer Characteristic Function). The characteristic function for layer l of NN $\mathcal{N}$, $f^l_{\mathcal{N}} : \mathbb{R}^{N^1} \to \mathbb{R}^{N^l}$, is defined as

$$f^l_{\mathcal{N}}(x) := f^l_F \circ \alpha \circ f^{l-1}_F \circ \cdots \circ f^3_F \circ \alpha \circ f^2_F(x).$$

Definition 3: (NN Characteristic Function). The characteristic function of NN $\mathcal{N}$, $f_{\mathcal{N}} : \mathbb{R}^{N^1} \to \mathbb{N}$, is defined as

$$f_{\mathcal{N}}(x) := \underset{i \in \{1,\ldots,N^L\}}{\arg\max} \{(f^L_{\mathcal{N}}(x))_i : i \in \{1, \ldots, N^L\}\}.$$

In Definition 3, we assume that when $\arg\max(.)$ is not a singleton, a deterministic tie-breaking procedure is used to select a single index.

2) Soft Decision Trees: We consider a binary SDT [12], [16]. Let $\mathcal{I}_{\mathcal{S}}$ and $\mathcal{L}_{\mathcal{S}}$ denote the sets of inner and leaf nodes for an SDT $\mathcal{S}$ with input dimension n. Each inner node $v \in \mathcal{I}_{\mathcal{S}}$ is associated with weights $w^v \in \mathbb{R}^n$ and a bias term $b^v \in \mathbb{R}$. At node v, w^v and b^v are applied to input x. The resulting value is passed through an activation function σ, producing a scalar, $p_v(x) = \sigma(x^\top w^v + b^v)$, which is compared to a threshold to determine whether to proceed to the left or right branch. In this paper, we use the sigmoid logistic activation function.

Each leaf node $v \in \mathcal{L}_{\mathcal{S}}$ is associated with a vector Q^v. In general, Q^v is a distribution over possible output selections. In our SDTs, the elements of Q^v will take values in $\{0, 1\}$. Rather than learning the parameters (w^v, b^v) or Q^v for each node v from data as in the literature [12], we derive these parameters via transformation of a reference NN. We denote the left and right child of an inner node v as $l(v)$ and $r(v)$, respectively, and the parent of v as $\rho(v)$. The root node of the SDT is v_0. We now define the SDT characteristic functions.

Definition 4: (SDT Node Characteristic Function). Let $\mathcal{S}$ be an SDT with input dimension n. Given input vectors $x \in \mathbb{R}^n$, the characteristic function of node v, $f^v_{\mathcal{S}} : \mathbb{R}^n \to \mathbb{N}$, is defined recursively as

$$f^v_{\mathcal{S}}(x) = \begin{cases} \arg\max_k \{Q^v_k\}, & \text{if } v \in \mathcal{L}_{\mathcal{S}}, \\ f^{l(v)}_{\mathcal{S}}(x), & \text{if } v \in \mathcal{I}_{\mathcal{S}} \wedge p_v(x) \leq 0.5, \\ f^{r(v)}_{\mathcal{S}}(x), & \text{if } v \in \mathcal{I}_{\mathcal{S}} \wedge p_v(x) > 0.5. \end{cases}$$

The output of each leaf node is then directly determined by a single path within the tree [12], rather than a weighted aggregation of all the potential tree paths [16].

Definition 5: (SDT Characteristic Function). The characteristic function of SDT $\mathcal{S}$ with input dimension n, $f_{\mathcal{S}} : \mathbb{R}^n \to \mathbb{N}$, is defined as $f_{\mathcal{S}}(x) := f^{v_0}_{\mathcal{S}}(x)$.

Throughout this work, we assume that NNs and SDTs use the same deterministic argmax-based tie-breaking procedure. We further introduce the notation $\mathcal{D}(v)$ to represent the effective *domain* of an SDT node v, i.e., the set of all possible inputs arriving at v. The set $\mathcal{D}(v)$ can be computed recursively in a top-down manner from the root node as

Rules	Conditions	Computation of $p_v(x)$ or Q^v
1. Split based on ReLU	$\exists\, i, l < L$ s.t. $\mathcal{D}(v) \cap \{\mathcal{F}_i^l[\![v]\!](x) \leq 0\} \neq \emptyset \wedge$ $\mathcal{D}(v) \cap \{\mathcal{F}_i^l[\![v]\!](x) > 0\} \neq \emptyset$	$p_v(x) = \begin{cases} \sigma(\mathcal{F}_i^l[\![v]\!](x)) \\ \text{for } \min i, l \text{ such that Rule 1 conditions hold} \end{cases}$
2. Split based on output layer	Rule 1 not satisfied and $\exists\, i_1, i_2$ s.t. $\mathcal{D}(v) \cap \bigcap_{i' \neq i_1} \{\mathcal{F}_{i_1}^L[\![v]\!](x) \geq \mathcal{F}_{i'}^L[\![v]\!](x)\} \neq \emptyset \wedge$ $\mathcal{D}(v) \cap \bigcap_{i' \neq i_2} \{\mathcal{F}_{i_2}^L[\![v]\!](x) \geq \mathcal{F}_{i'}^L[\![v]\!](x)\} \neq \emptyset$	$p_v(x) = \sigma(\mathcal{F}_{i_1}^L[\![v]\!](x) - \mathcal{F}_{i_2}^L[\![v]\!](x))$
3. Form leaf node	Neither Rule 1 nor Rule 2 satisfied	$Q_i^v = \begin{cases} 1 & \text{if } i = \arg\max_k \{\mathcal{F}_k^L[\![v]\!](x)\}\ \forall x \in \mathcal{D}(v) \\ 0 & \text{otherwise} \end{cases}$

TABLE I: Rules for constructing an SDT node v.

follows

$$\mathcal{D}(v) = \begin{cases} \mathbb{R}^n, & \text{if } v = v_0, \\ \mathcal{D}(\rho(v)) \cap \{x | p_{\rho(v)}(x) \leq 0.5\}, & \text{if } v \neq v_0 \wedge l(\rho(v)) = v, \\ \mathcal{D}(\rho(v)) \cap \{x | p_{\rho(v)}(x) > 0.5\}, & \text{if } v \neq v_0 \wedge r(\rho(v)) = v, \end{cases} \quad (1)$$

where n is the input dimension of the SDT.

For example, for the SDT in Fig. 1b, the inputs $x = (x_0, x_1)^\top \in \mathbb{R}^2$ are processed starting from the root node v_0, where $\mathcal{D}(v_0) = \mathbb{R}^2$. The input range related to node v_1 is $\mathcal{D}(v_1) = \{x \in \mathbb{R}^2 | x_1 \leq 0\}$ and v_1 is reached only if $x_1 \leq 0$. Similarly, node v_2 is reached when $x_1 > 0$, i.e., $\mathcal{D}(v_2) = \{x \in \mathbb{R}^2 | x_1 > 0\}$. In the remainder of the paper, we omit the reference to the underlying state-space in the expressions for the node input domains, when it is clear from the context, and simply write, e.g., $\mathcal{D}(v_1) = \{x_1 \leq 0\}$ and $\mathcal{D}(v_2) = \{x_1 > 0\}$.

III. Transformation from Neural Network to Decision Tree

We present an algorithm for constructing an SDT $\mathcal{S}_\mathcal{N}$ which is equivalent to a reference NN $\mathcal{N}$ in the sense that $f_\mathcal{N}(x) = f_{\mathcal{S}_\mathcal{N}}(x)$ for all inputs $x \in \mathbb{R}^n$. We outline the relation between fully connected NNs with ReLU activation and argmax output and SDTs based on our transformation in Fig. 1.

In the NN of Fig. 1a, the output of the single neuron in layer 2 following the ReLU activation is

$$\alpha(f_\mathcal{N}^2(x)) = \alpha\left(W^2 x + B^2\right) = \alpha(x_1) = \begin{cases} 0, & \text{if } x_1 \leq 0, \\ x_1, & \text{if } x_1 > 0. \end{cases} \quad (2)$$

Carrying these two cases forward through the output layer $L = 3$, we have

$$f_\mathcal{N}^L(x) = \begin{cases} B^L = (0.001,\, 0,\, 0)^\top, & \text{if } x_1 \leq 0, \\ W^L x_1 + B^L = (0.001,\, 0,\, x_1)^\top, & \text{if } x_1 > 0. \end{cases} \quad (3)$$

Thus, when $x_1 \leq 0$, we have

$$f_\mathcal{N}(x) = \underset{i \in \{1,\ldots,N^L\}}{\arg\max} \{(f_\mathcal{N}^L(x))_i\} = 1,$$

while when $x_1 > 0$, we have

$$f_\mathcal{N}(x) = \begin{cases} 1, & \text{if } x_1 \leq 0.001, \\ 3, & \text{if } x_1 > 0.001, \end{cases} \quad (4)$$

assuming we take the lowest index to break ties.

Turning to the SDT in Fig. 1b, we observe that (2) partitions the input space into two subsets based on the inner node's ReLU activation. This is precisely the split that occurs at the root node v_0 of the SDT toward nodes v_1 and v_2. Then, (4) further splits one of these subsets based on the output layer values. This split occurs at v_2 to provide nodes v_3 and v_4. Finally, we have three regions where $f_\mathcal{N}(x)$ is constant, corresponding to leaf nodes v_1, v_3, and v_4 in Fig. 1b.

As shown in (2)-(4) for subsets of the NN input space, based on the sequence of ReLU activations, $f_\mathcal{N}^L(x)$ is an affine function. Such subsets may be further partitioned based on the argmax operation. Therefore, the operation of fully connected NNs with ReLU activations and argmax output can be understood in terms of successive assignment of inputs from the state space to increasingly refined subspaces, which can be shown to consist of convex polyhedra [13]. Identification of SDT splits and leaf node assignments based on NN neuron activations and output layer outputs forms the core of our transformation technique, which we explain further in this section. See the extended version of this paper [17] for an application of the transformation to the example in Fig. 1.

A. Pre and Post-Activation Formulas

We establish the relationship between $\mathcal{N}$ and $\mathcal{S}_\mathcal{N}$ by first introducing, for each node v, a *pre-activation function* $\mathcal{F}_i^l[\![v]\!] : \mathcal{D}(v) \to \mathbb{R}$ and a *post-activation function* $\overline{\mathcal{F}_i^l[\![v]\!]} : \mathcal{D}(v) \to \mathbb{R}$. The pre-activation function provides the output of the i-th neuron in the l-th layer of $\mathcal{N}$ as a function of the input in $\mathcal{D}(v)$ prior to the ReLU. The *post-activation function* gives the neuron output after the ReLU. The pre-activation functions for node v are defined for $l = 2, \ldots, L$ and $i = 1, \ldots, N^l$ as

$$\mathcal{F}_i^l[\![v]\!](x) = \begin{cases} \sum_{j=1}^{N^1} W_{i,j}^l x_j + B_i^l, & \text{if } l = 2, \\ \sum_{j=1}^{N^{l-1}} W_{i,j}^l \overline{\mathcal{F}_j^{l-1}[\![v]\!]}(x) + B_i^l, & \text{if } l > 2 \wedge \overline{\mathcal{F}_j^{l-1}[\![v]\!]}(x) \neq \bot \\ & \quad \text{for } j = 1, 2, \ldots, N^{l-1}, \\ \bot, & \text{otherwise.} \end{cases} \quad (5)$$

Here, $\bot$ denotes an undefined value. The post-activation functions for node v are defined for $l = 2, \ldots, L-1$ and $i = 1, \ldots, N^l$ as

$$\overline{\mathcal{F}_i^l[\![v]\!]}(x) = \begin{cases} \mathcal{F}_i^l[\![v]\!](x), & \text{if } \mathcal{F}_i^l[\![v]\!] \neq \bot \wedge \mathcal{D}(v) \cap \{\mathcal{F}_i^l[\![v]\!](x) \leq 0\} = \emptyset, \\ 0, & \text{if } \mathcal{F}_i^l[\![v]\!] \neq \bot \wedge \mathcal{D}(v) \cap \{\mathcal{F}_i^l[\![v]\!](x) > 0\} = \emptyset, \\ \bot, & \text{otherwise.} \end{cases} \quad (6)$$

We define $\mathcal{F}^l[\![v]\!] := (\mathcal{F}_1^l[\![v]\!], \ldots, \mathcal{F}_{N^l}^l[\![v]\!])^\top$ and $\overline{\mathcal{F}^l[\![v]\!]}$ similarly for all $l = 2, \ldots, L$. The pre-activation formulas play a crucial role in determining the pre-threshold branching functions p_v at each inner node v of $\mathcal{S}$.

B. SDT Split and Leaf Formation Rules

We construct the branches of $\mathcal{S}_\mathcal{N}$ based on the rules in Table I. By starting with the root node v_0, for each node v in $\mathcal{S}_\mathcal{N}$, we use Table I to obtain $p_v(x)$ or Q^v in a top-down fashion from root to leaves, as further explained below.

Rule 1. If there exists a neuron i in layer l of $\mathcal{N}$ such that $D_0 := \mathcal{D}(v) \cap \{\alpha(\mathcal{F}_i^l[\![v]\!](x)) = 0\} \neq \emptyset$ and $D_1 := \mathcal{D}(v) \cap \{\alpha(\mathcal{F}_i^l[\![v]\!](x)) > 0\} \neq \emptyset$ hold, we form a split at node v in $\mathcal{S}_\mathcal{N}$. $\mathcal{D}(v)$ is then partitioned along the hyperplane $\mathcal{F}_i^l[\![v]\!](x) = 0$, with $\mathcal{D}(l(v)) = D_0$ and $\mathcal{D}(r(v)) = D_1$. For example, for node v_0 in Fig. 1b, we create a split at v_0 with $p_{v_0}(x) = \sigma(\mathcal{F}_1^2[\![v_0]\!](x))$.

Rule 2. Suppose no partitions can be found for $\mathcal{D}(v)$ based on the ReLUs according to Rule 1. Then, $\mathcal{D}(v)$ may be further partitioned based on the output layer values. If there exist i_1 and i_2 and sets $D_{i_1} := \{x \in \mathcal{D}(v) | \arg\max_i \{(\mathcal{F}^L[\![v]\!](x))_i\} = i_1\} \neq \emptyset$ and $D_{i_2} := \{x \in \mathcal{D}(v) | \arg\max_i \{(\mathcal{F}^L[\![v]\!](x))_i\} = i_2\} \neq \emptyset$, we form a split at node v. $\mathcal{D}(v)$ is then partitioned along the hyperplane $\mathcal{F}_{i_2}^L[\![v]\!](x) - \mathcal{F}_{i_1}^L[\![v]\!](x) = 0$, giving $D_{i_1} \subseteq \mathcal{D}(l(v))$ and $D_{i_2} \subseteq \mathcal{D}(r(v))$. For example, for node v_2 in Fig. 1b, we create a split at v_2 and set $p_{v_2}(x)$ as $\sigma(\mathcal{F}_1^3[\![v_2]\!](x) - \mathcal{F}_3^3[\![v_2]\!](x))$.

Rule 3. If neither Rule 1 nor Rule 2 can be applied to further partition $\mathcal{D}(v)$, then the index of the neuron with the maximum output at layer L, i.e., the outcome of the $\arg\max$ operator remains constant over the entire set $\mathcal{D}(v)$. We then declare v as a leaf node, setting $Q_i^v = 1$ for the neuron index corresponding to the largest output, and 0 otherwise. For example, for node v_1 in Fig. 1b we set $Q_1^{v_1}(x) = 1$ and $Q_2^{v_1}(x) = Q_3^{v_1}(x) = 0$.

The conditions above, such as non-emptiness of $\mathcal{D}(v) \cap \{\alpha(\mathcal{F}_i^l[\![v]\!](x)) = 0\}$ in Rule 1, can be formulated and efficiently solved in terms of feasibility problems for linear programs.

C. Transformation Procedure

We construct $\mathcal{S}_\mathcal{N}$ by starting with the root node v_0 and building the binary tree downwards. We first construct the left-hand branches until a leaf node is discovered, from which we backtrack the tree to define the unexplored right branches. A recursive method implementing this procedure is presented in Algorithm 1. The transformation procedure effectively identifies a partition of the input space according to the domains associated with the leaf nodes of the SDT. Within these sets, the neural network characteristic function $f_\mathcal{N}$ is constant, and due to our choice of Q^v at each leaf node v, we have that $f_\mathcal{N}(x) = f_{\mathcal{S}_\mathcal{N}}(x)$ over $\mathcal{D}(v)$. Taking a union over the SDT partitions, we establish the equivalence of $\mathcal{N}$ and $\mathcal{S}_\mathcal{N}$ in Theorem 1. As detailed in the extended version of this paper [17], Theorem 1 can be proved by induction on layer l to show that $\mathcal{F}^L[\![v]\!](x) = f_\mathcal{N}^L(x)$ holds for leaf nodes $v \in \mathcal{L}_{\mathcal{S}_\mathcal{N}}$.

Theorem 1: For a given NN $\mathcal{N}$ and its SDT transformation $\mathcal{S}_\mathcal{N}$ using Algorithm 1 with input space $\mathbb{R}^n$, the corresponding characteristic functions are pointwise equal, i.e., $f_\mathcal{N}(x) = f_{\mathcal{S}_\mathcal{N}}(x)$ for all $x \in \mathbb{R}^n$.

Differently from previous algorithms in the literature [13]–[15], our algorithm only generates essential branches during the creation of the SDT. Given the NN input and output layer sizes and the number of hidden layers, the SDT size scales polynomially in the maximum hidden layer width. The size complexity of $\mathcal{S}_\mathcal{N}$ in terms of the number of nodes is stated in Theorem 2, whose proof, provided in the extended version [17], is based on the upper bound on the number of piecewise affine regions achievable with ReLU NNs [18].

Algorithm 1 SDT Transformation $\mathcal{T}(.)$

Global Variables: NN parameters L, $\{N^l\}_{l=1}^L$, $\{W^l, b^l\}_{l=2}^L$
Input: v
Compute $\mathcal{D}(v)$ via (1)
for $l = 2, \ldots, L$ **do**
  for $i = 1, \ldots, N^l$ **do**
    Compute $\mathcal{F}_i^l[\![v]\!]$, $\overline{\mathcal{F}_i^l[\![v]\!]}$ via (5), (6)
if $\exists\ i, l$ s.t. $(\mathcal{D}(v), \mathcal{F}_i^l[\![v]\!])$ satisfy Rule 1 **then**
  $p_v(x) = \sigma(\mathcal{F}_i^l[\![v]\!](x))$, create $l(v)$, $r(v)$
  $\mathcal{T}(l(v))$ ▷ Recursion
  $\mathcal{T}(r(v))$
else if $\exists\ i_1, i_2$ s.t. $(\mathcal{D}(v), \mathcal{F}_{i_1}^L[\![v]\!], \mathcal{F}_{i_2}^L[\![v]\!])$ satisfy Rule 2 **then**
  $p_v(x) = \sigma(\mathcal{F}_{i_1}^L[\![v]\!](x) - \mathcal{F}_{i_2}^L[\![v]\!](x))$, create $l(v)$, $r(v)$.
  $\mathcal{T}(l(v))$ ▷ Recursion
  $\mathcal{T}(r(v))$
else
  $Q_i^v = \begin{cases} 1 & \text{if } i = \arg\max_k \{\mathcal{F}_k^L[\![v]\!]\}, \forall x \in \mathcal{D}(v) \\ 0 & \text{otherwise} \end{cases}$

Theorem 2: Let $\mathcal{N}$ be a NN and $\mathcal{S}_\mathcal{N}$ the SDT resulting from the application of Algorithm 1 to $\mathcal{N}$. Denote the number of nodes in $\mathcal{S}_\mathcal{N}$ by $|\mathcal{S}_\mathcal{N}|$. Then, we obtain

$$|\mathcal{S}_\mathcal{N}| = O\left(\prod_{l=2}^{L-1} \frac{(N^l)^{(N^1)}}{(N^1)!} 2^{N^L}\right) = O\left(\frac{2^{N^L}}{(N^1!)^L} \bar{l}^{N^1 L}\right),$$

where $\bar{l} := \max_{l \in \{2,\ldots,L-1\}} N^l$.

IV. Verification Formulation

We describe the verification problems that we solve to evaluate the impact of the proposed transformation. We consider closed-loop controlled dynamical systems with state and action spaces $\mathcal{X}$ and $\mathcal{U}$, respectively, and dynamics $h : \mathcal{X} \times \mathcal{U} \to \mathcal{X}$. Let $\pi : \mathcal{X} \to \mathcal{U}$ denote a time-invariant Markovian policy (controller) mapping states to actions. Given a system dynamics h, an initial set $\mathcal{X}_i$, and goal set $\mathcal{X}_g$, we wish to determine whether $\mathcal{X}_g$ is reachable in finite horizon T for all initial states $x_0 \in \mathcal{X}_i$, under h and policy π. If so, we say that the specification $(\mathcal{X}_i, \mathcal{X}_g, T)$ is verified for π. In this context, we consider two verification approaches.

Problem 1: (One-Shot Verification). We "unroll" the system dynamics at time instant $t \in \{0, \ldots, T\}$ and encode the verification problem to a satisfiability modulo theory (SMT) problem [19] using the bounded model checking approach [20]. Verification of $(\mathcal{X}_i, \mathcal{X}_g, T)$ for a policy π is then equivalent to showing that the following formula is *not* satisfiable:

$$\phi_1(\pi) := (\mathcal{X}_i) \wedge (\overline{\mathcal{X}_g}) \bigwedge_{t=0}^{T-1} (x_{t+1} = h(x_t, u_t)) \wedge (u_t = \pi(x_t)). \quad (7)$$

If $\phi_1(\pi)$ is false, then π is guaranteed to drive the system from any $x_0 \in \mathcal{X}_i$ to $\mathcal{X}_g$ within the finite horizon T.

We observe that (7) requires encoding T replicas of the NN (policy π) in the control loop, which can make the SMT problem intractable due to its computational complexity. Therefore, we also consider an alternative verification approach via reachability analysis. In particular, we adopt

Environment	$\mathfrak{i}_l$	$\mathfrak{i}_u$	$\mathfrak{g}$
MountainCar	-0.11	-0.1	0.5
CartPole	-0.1	N/A	$\pi/15$

TABLE II: Hyperparameters in OpenAI Gym.

a *recursive reachability analysis* [21], where we use the s-step unrolled dynamics, with $s \ll T$, to compute an over-approximation of the s-step reachable set in terms of a rectangle.

Problem 2: (Recursive Reachability Analysis (RRA)). We fix a step parameter s and recursively compute reachable sets $\mathcal{R}_t$, where $t = ms$ for $m \in \mathbb{N}$. Given a system dynamics h, policy π and step parameter s, we encode the reachable set at time t as the SMT formula

$$\mathcal{R}_t(\pi) = \left\{ x \middle| x = x_t \wedge (\mathcal{R}_{t-s}) \wedge \bigwedge_{k=t-s}^{t-1} (x_{k+1} = h(x_k, u_k)) \wedge (u_k = \pi(x_k)) \right\}. \tag{8}$$

By setting $\mathcal{R}_0 = \mathcal{X}_i$, we can verify the specification $(\mathcal{X}_i, \mathcal{X}_g, T)$ by checking satisfaction of the following SMT formula

$$\phi_{\text{RRA}}(\pi) := (\mathcal{X}_i) \wedge (\overline{\mathcal{X}_g}) \wedge \bigcup_{m=1}^{\lfloor T/s \rfloor} \mathcal{R}_{ms}(\pi).$$

While RRA may yield overly conservative reachable sets due to error propagation, it is often more tractable than one-shot verification, since it decomposes the overall reachability problem into a set of smaller sub-problems, each having a finite horizon of s and encoding the NN policy only s times, with $s \ll T$.

V. Case Studies

A. The Environments

1) MountainCar-v0: In the MountainCar control task, an underpowered car needs to reach the top of a hill starting from a valley within a fixed time horizon T [22]. The state vector $x_t \in \mathbb{R}^2$ comprises the car's position $x_{t,0}$ and horizontal velocity $x_{t,1}$ at time step t, while the input $u_t \in \{-1, 0, 1\}$ represent the car's acceleration action, either left (L), idle (I), or right (R), respectively. The system dynamics are given in the extended version of this paper [17].

While the reference NNs are trained to reach the top of the hill ($x \geq \mathfrak{g}$) in the minimum number of steps, we seek to verify whether a given controller will reach the goal state within T steps, starting from any point in an initial interval. Our specification $(\mathcal{X}_i, \mathcal{X}_g, T)$ is given by

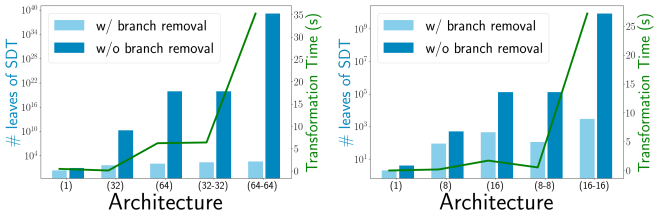
$$\mathcal{X}_i = \{x_0 \in [\mathfrak{i}_l, \mathfrak{i}_u], x_1 = 0\}, \ \mathcal{X}_g = \bigcup_{t=0}^{T} \{x_{t,0} \geq \mathfrak{g}\},$$

where $\mathfrak{i}_l$, $\mathfrak{i}_u$, and $\mathfrak{g}$ are parameters.

2) CartPole-v1: In the CartPole control task [23], a pole is attached to a cart moving along a frictionless track, to be balanced upright for as long as possible within a fixed horizon T. The state vector $x_t \in \mathbb{R}^4$ consists of the cart position $x_{t,0}$, the cart velocity $x_{t,1}$, the pole angle $x_{t,2} = \theta_t$, and the pole angular velocity $x_{t,3}$ at time step t. The input $u_t \in \{0, 1\}$ denotes the acceleration of the cart to the left or right. The system dynamics are given in the extended version [17].

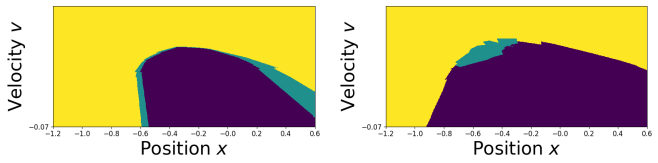
We seek to verify that the pole is balanced upright within tolerance $\mathfrak{g}$ at the end of horizon T, starting from a range of initial cart positions and pole angles. With $\mathfrak{i}_l$, $\mathfrak{g}$ as parameters, our specification $(\mathcal{X}_i, \mathcal{X}_g, T)$ is given by

$$\mathcal{X}_i = \{x_{0,0} \in [\mathfrak{i}_l, 0], x_{0,2} \in [\mathfrak{i}_l, 0], x_{0,1} = x_{0,3} = 0\}$$
$$\mathcal{X}_g = \{|x_{T,2}| \geq \mathfrak{g}\}.$$



(a) MountainCar (b) CartPole

Fig. 2: Transformation time and size of the transformed SDT.



(a) $N = (2, 64, 3)$ (b) $N = (2, 64, 64, 3)$

Fig. 3: MountainCar NN policies, L (▬), I (▬), R (▬).

B. Evaluation Results

We adopt the problem setups available in the open-source package OpenAI Gym [10] and the parameters in Table II. We train NN controllers for 1×10^7 steps. Fig. 2 illustrates the transformation time and the number of leaves of the SDT for each NN controller. We also display the estimated number of leaves under a naïve transformation method that generates a split at every node for every neuron in every layer, denoted by "w/o branch removal". This value signifies the percentage reduction in size achieved by our transformation. As shown in Fig. 2, the percentage of size reduction for the SDT increases with the number of NN layers. Furthermore, the size of the transformed SDT for the CartPole problem increases faster than for the MountainCar problem, which can be attributed to the difference in $N^1 = |x_t|$. Nevertheless, we were able to transform all the controllers in less than 60 s.

Fig. 3 plots the policies and SDT leaf node partitions for example MountainCar controllers. Note that constant action state-space regions become more complex as the depth of the NN increases.

All SMT problems in our experiments are solved using an in-house implementation of a satisfiability modulo convex programming (SMC) solver [24], which integrates the Z3 satisfiability (SAT) solver [25] with Gurobi [26]. For the one-shot verification approach, Fig. 4 shows a comparison of the verification times for $\mathcal{N}$ with $N = (2, 1, 3)$ and $\mathcal{S}_{\mathcal{N}}$ with three leaf nodes in the MountainCar problem. The NN verification time increases significantly compared to the equivalent SDT. However, verification takes more than 60000 seconds for $\mathcal{N}$ when $T > 38$ and for $\mathcal{S}_{\mathcal{N}}$ when $T > 64$.

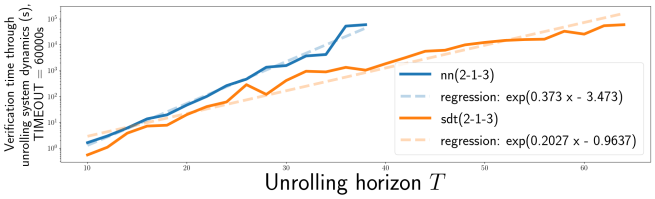
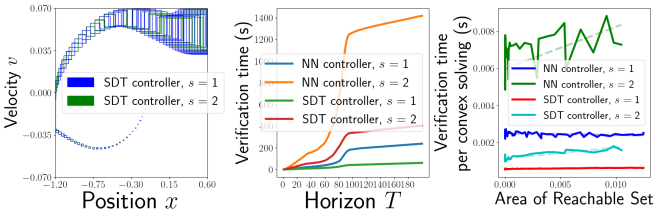
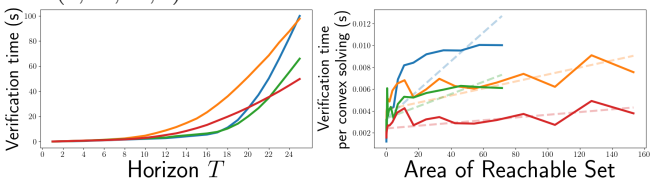


Fig. 4: Comparison of MountainCar NN and SDT controller one-shot verification times.



(a) Reachable sets and verification time for MountainCar NN with $N = (2, 32, 32, 3)$.



(b) Verification time for CartPole NN (▬) and SDT (▬) with $N = (4, 8, 2)$, and NN (▬) and SDT (▬) with $N = (4, 8, 8, 2)$.

Fig. 5: Recursive Reachability Analysis results.

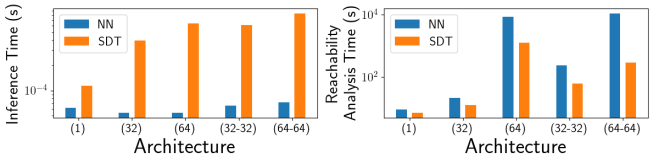
For the CartPole problem, all controllers fail to solve the verification problem (7) within 60000 seconds for $T > 10$.

To assess the efficiency of RRA for NN and SDT controllers, we measured runtimes both for individual reachability set determination as well as overall verification time. Fig. 5a plots the reachable sets, along with overall verification and reachable set generation times for an example MountainCar controller with $s \in \{1, 2\}$ in (8). Fig. 5b plots verification and reachable set generation times for a collection of CartPole controllers with $s = 1$ in (8). As shown, RRA verification becomes more efficient using the equivalent SDT controllers, rather than the reference NNs.

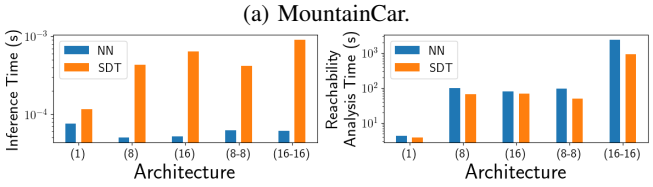
Fig. 6 compares the average inference and overall RRA verification times for a collection of NN and equivalent SDT controllers. We set $T = 200$ for the MountainCar problem and $T = 25$ for the CartPole problem. While the inference time for the SDT increases with the number of layers in NN, our results show that the transformed SDT controller accelerates verification for the MountainCar problem by up to $21\times$ and for the CartPole problem by up to $2\times$. Moreover, the larger the number of layers in the NN, the larger the difference between runtime for NN verification and runtime for SDT verification.

VI. Conclusion

We proposed a cost-effective algorithm to construct equivalent SDT controllers from any discrete-output argmax-based NN controller. Empirical evaluation of formal verification tasks performed on two benchmark OpenAI Gym environments shows a significant reduction in verification times for



(a) MountainCar.



(b) CartPole.

Fig. 6: Inference and RRA runtimes for NN and SDT controllers. For RRA, $T = 200$ for MountainCar and $T = 25$ for CartPole.

SDT controllers. Future work includes experimenting with more sophisticated RL environments and performing more extensive comparisons with state-of-the-art verification tools.

References

[1] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, 2016.
[2] Y. Taigman, M. Yang *et al.*, "DeepFace: Closing the Gap to Human-Level Performance in Face Verification," in *CVPR*, 2014.
[3] D. Silver, A. Huang, C. J. Maddison *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, 2016.
[4] N. Dahlin *et al.*, "Practical Control Design for the Deep Learning Age: Distillation of Deep RL-Based Controllers," in *Allerton*, 2022.
[5] N. Naik and P. Nuzzo, "Robustness Contracts for Scalable Verification of Neural Network-Enabled Cyber-Physical Systems," in *ACM-IEEE Int. Conf. on Formal Methods and Models for System Design*, 2020.
[6] P. Nuzzo, "From Electronic Design Automation to Cyber-Physical System Design Automation: A Tale of Platforms and Contracts," in *Proc. of the 2019 International Symposium on Physical Design*, 2019.
[7] G. Hinton, O. Vinyals, and J. Dean, "Distilling the Knowledge in a Neural Network," *arXiv preprint arXiv:1503.02531*, 2015.
[8] J. Ba and R. Caruana, "Do Deep Nets Really Need to be Deep?" *Advances in neural information processing systems*, 2014.
[9] O. Bastani, Y. Pu, and A. Solar-Lezama, "Verifiable Reinforcement Learning via Policy Extraction," *Adv. Neural Inf. Process. Syst.*, 2018.
[10] G. Brockman *et al.*, "OpenAI Gym," *arXiv:1606.01540*, 2016.
[11] S. Ross *et al.*, "A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning," in *AISTATS*, 2011.
[12] N. Frosst and G. Hinton, "Distilling a Neural Network Into a Soft Decision Tree," *arXiv preprint arXiv:1711.09784*, 2017.
[13] G.-H. Lee and T. S. Jaakkola, "Oblique Decision Trees from Derivatives of ReLU Networks," *arXiv preprint arXiv:1909.13488*, 2019.
[14] A. Sudjianto, W. Knauth, R. Singh, Z. Yang *et al.*, "Unwrapping The Black Box of Deep ReLU Networks: Interpretability, Diagnostics, and Simplification," *arXiv preprint arXiv:2011.04041*, 2020.
[15] D. T. Nguyen, K. E. Kasmarik, and H. A. Abbass, "Towards Interpretable ANNs: An Exact Transformation to Multi-Class Multivariate Decision Trees," *arXiv preprint arXiv:2003.04675*, 2020.
[16] O. Irsoy *et al.*, "Soft Decision Trees," in *ICPR*, 2012.
[17] K. Chang, N. Dahlin, R. Jain, and P. Nuzzo, "Exact and Cost-Effective Automated Transformation of Neural Network Controllers to Decision Tree Controllers," *arXiv preprint arXiv:2304.06049*, 2023.
[18] T. Serra, C. Tjandraatmadja *et al.*, "Bounding and Counting Linear Regions of Deep Neural Networks," in *ICML*, 2018.
[19] C. Barrett and C. Tinelli, *Satisfiability Modulo Theories*, 2018.
[20] A. Biere, "Bounded Model Checking." *Handbook of SAT.*, 2009.
[21] S. Chen, V. M. Preciado, and M. Fazlyab, "One-Shot Reachability Analysis of Neural Network Dynamical Systems," in *ICRA*, 2023.
[22] A. W. Moore, "Efficient memory-based learning for robot control," University of Cambridge, Computer Laboratory, Tech. Rep., 1990.
[23] D. Michie and R. A. Chambers, "BOXES: An experiment in adaptive control," *Machine intelligence*, 1968.
[24] Y. Shoukry, P. Nuzzo *et al.*, "SMC: Satisfiability Modulo Convex Programming," *Proceedings of the IEEE*, 2018.
[25] L. De Moura *et al.*, "Z3: An Efficient SMT Solver," in *TACAS*, 2008.
[26] T. Achterberg, "What's new in gurobi 9.0," *https://www.gurobi.com*, 2019.