Interpretable Reinforcement Learning via Differentiable Decision Trees

Ivan Dario Jimenez Rodriguez
School of Interactive Computing
Georgia Institute of Technology
Atlanta, GA, USA
ijimenez3@gatech.edu

Taylor Killian and Sung-Hyun Son
AMMDT Division
MIT Lincoln Laboratory
Lexington, MA, USA
{Taylor.Killian,sson}@ll.mit.edu

Matthew Gombolay
School of Interactive Computing
Georgia Institute of Technology
Atlanta, GA, USA
Matthew.Gombolay@cc.gatech.edu

***Abstract*—Decision trees are ubiquitous in machine learning for their ease of use and interpretability; however, they are not typically implemented in reinforcement learning because they cannot be updated via stochastic gradient descent. Traditional applications of decision trees for reinforcement learning have focused instead on making commitments to decision boundaries as the tree is grown one layer at a time. We overcome this critical limitation by allowing for a gradient update over the entire tree structure that improves sample complexity when a tree is fuzzy and interpretability when sharp. We offer three key contributions towards this goal. First, we motivate the need for policy gradient-based learning by examining the theoretical properties of gradient descent over differentiable decision trees. Second, we introduce a regularization framework that yields interpretability via sparsity in the tree structure. Third, we demonstrate the ability to construct a decision tree via policy gradient in canonical reinforcement learning domains and supervised learning benchmarks.**

I. Introduction

Reinforcement learning (RL) with neural network function approximators, known as “Deep RL,” has achieved tremendous results in recent years [1]. Deep RL uses multi-layered neural networks to represent policies that are trained to maximize an agent’s expected future reward. However, these neural-network-based approaches are largely uninterpretable due to the millions of parameters involved.

In safety-critical domains, such as healthcare, aviation, and military operations, interpretability is of utmost importance. Human operators must be able to interpret and follow step-by-step procedures and checklists [10, 11, 5]. Of the class of machine learning methods that can generate such a set of procedures, decision tree algorithms are perhaps the most highly developed [26]. While interpretable machine learning methods offer the promise of revolutionizing safety-critical domains [14], they are generally unable to match the singular performance seen in Deep RL [21, 8]. Decision trees have often been viewed as the de facto technique for interpretable machine learning [18, 14] as they can learn compact representations of underlying relationships within data [3]. In prior work, decision trees have been applied to RL problems where they served as function approximators: compactly representing information about which action to take in which state [7, 8, 20, 17].

The challenge with applying decision trees as function approximators in RL lies in the online nature of the learning problem. As such, the decision tree model (or any model) must be able to adapt to handle the non-stationary distribution of the observed data. The two primary techniques for RL function approximation are Q-learning [25] and policy gradient(PG) [24]. Underlying these learning mechanisms are variants of stochastic gradient descent (SGD); at each time-step, the RL agent takes an action based on the prediction of the approximated policy, receives a reward from the environment, and computes how to update the setting of each of the policy’s parameters [2, 9]. Decision trees are not typically amenable to gradient descent due to their Boolean nature; they are a collection of nested if-then rules. Therefore, researchers have used heuristic, non-gradient-descent-based methods for training decision trees [7, 8, 17]. A common approach when applying decision trees to RL is to perform online state aggregation using heuristics to update or expand a decision tree’s terminal (i.e., leaf) nodes rather than seeking to update the entire model with respect to a global loss function [17]. Researchers have also attempted to use decision trees for RL by training them in batch mode, completely re-learning the tree from scratch to account for the non-stationarity introduced by an improving policy [7]. While effective, this approach is inefficient when seeking algorithms that scale to realistic situations. Despite these attempts, success comparable to that of modern deep learning approaches has been elusive [8].

Seeking to develop a decision tree formulation amenable to gradient descent, Suárez and Lutsko formulated a continuous and fully differentiable decision tree (DDT), in which they adopted a sigmoidal representation of each decision node’s splitting criterion [22]. Suárez and Lutsko applied their approach to offline, supervised learning but not to RL. Researchers have continued to explore continuous decision tree formulations [15, 12], while there has been limited success in applying such fuzzy trees to RL (e.g., [20]).

In this paper, we develop and demonstrate an end-to-end framework for RL with function approximation via DDTs. We provide three key contributions: first, we examine the theoretical properties for gradient descent over DDTs, motivating the need for PG based learning. To the best of our knowledge, this is the first investigation of the optimization surfaces of Q-learning and PGs for DDTs. Second, we introduce a regularization formulation to ensure interpretability via sparsity in the tree structure. Third, we demonstrate the novel ability to seamlessly update an entire decision tree via PG in canonical RL domains to produce an interpretable, sharp policy.

II. Preliminaries

In this section, we highlight the traditional decision tree and describe how Suárez and Lutsko augmented this model to be fully differentiable for gradient descent. We also review RL, Q-Learning, and PG.

A. Decision Trees

A decision tree is a directed, acyclic graph, with nodes and edges, that takes as input an example, x, performs a forward recursion, and returns a label $\hat{y}$ as shown in Equations 1-3. There are two types of nodes: *decision* nodes and *leaf* nodes. Decision and leaf nodes have an outdegree of two and zero, respectively. All nodes, η, have an indegree of one except for the root node, η_o, which has an indegree of zero. Each decision node η is represented as a Boolean expression, μ_η (Equation 3), where x_{j_η} and ϕ_η are the selected feature and splitting threshold, respectively, for decision node η. For each decision node, the left outgoing edge is labeled "true" and the right outgoing edge is labeled "false." If μ_η is evaluated true (or false) for an example, then the *left child* node, $\eta_\swarrow$, (or *right child* node, $\eta_\searrow$) is considered next. If the child node is a decision node, the process is repeated until a leaf node is reached. Once a leaf node is reached, the tree returns the label represented by that leaf node. The problem of finding the optimal decision tree is to determine, j_η^* ϕ_η^*, the best feature and splitting criterion, for each decision node the label, y_η, for each leaf node; and the structure of the tree (i.e., whether, for each node η, there exists a corresponding child node).

$$\hat{y}(x) := T_{\eta_o}(x) \tag{1}$$

$$T_\eta(x) := \begin{cases} y_\eta, & \text{if } \nexists \eta_\swarrow \\ \mu_\eta(x) T_{\eta_\searrow}(x) + (1 - \mu_\eta(x)) T_{\eta_\swarrow}(x), & \text{o/w} \end{cases} \tag{2}$$

$$\mu_\eta(x) := \begin{cases} 1, & \text{if } x_{j_\eta} < \phi_\eta \\ 0, & \text{o/w} \end{cases} \tag{3}$$

There are many heuristic techniques for learning decision trees with a batch data set that reason about the entropy, r-squared error, or other loss function [3]. A limitation of these Boolean decision trees is that one cannot readily apply standard gradient descent update rules since a tree is fixed after generation. Some researchers have tried heuristic approaches to iteratively grow trees in an RL context [17]; however, these approaches do not allow for a natural update of the entire structure of the tree in online learning environments.

B. DDTs

Suárez and Lutsko provide one of the first DDT models. In their approach, they employ a sigmoid formulation for Equation 3, with a linear combination of m features, x_j, weighted by β_η, and a steepness parameter a_η. Suárez and Lutsko demonstrate that one can then compute the gradient of the tree with respect to the tree's parameters, ϕ_η, β_η, and a_η, for all nodes, η to approximately solve classification and regression problems [22].

$$\mu_\eta(x) := \frac{1}{1 + e^{-(a_\eta(\beta_\eta^\intercal x + \phi_\eta))}} \tag{4}$$

While there are limitations to this approach (i.e., whether a weighted, linear combination of many features is interpretable), we believe this model is at least a strong building block towards developing interpretable, machine learning models amenable to gradient descent.

C. Reinforcement Learning

RL is an approach within machine learning where an agent is tasked with learning the optimal sequence of actions that will lead to the maximal expected future reward [23]. The actions and observations of the agent are traditionally abstracted as a Markov Decision Process (MDP). Formally, an MDP is a five-tuple $\langle S, A, P, \gamma, R \rangle$ defined as follows: S is the set of states; A is the set of actions; $P : S \times A \times S \rightarrow [0, 1]$ is the transition matrix describing the probability that taking action $a \in A$ in state $s \in S$ will result in state $s' \in S$; $\gamma \in [0, 1]$ is the discount factor which states how important it is to receive a reward in the current state versus a future state; and $R : S \times A \rightarrow \mathbb{R}$ is the function dictating the reward an agent receives by taking action $a \in A$ in state $s \in S$. In RL, the goal is to learn a policy, $\pi : S \rightarrow A$, that prescribes which action to take in each state in order to maximize the agent's future expected reward, as defined in Equation 5:

$$\pi^*(s) := \arg\max_\pi V^\pi(s) := \arg\max_\pi E\left[\sum_{t=0}^{\infty} \gamma^t r_t | s_o = s\right] \tag{5}$$

Here, π^* is the optimal policy and $V^\pi(s)$ is the value of policy π when starting in state $s = s_o$. There are two widely practiced approaches to learn such a policy: Q-learning and PG.

1) Q-learning: In Q-learning, one seeks to learn a Q-function, $Q^\pi : S \times A \rightarrow \mathbb{R}$, which returns the expected future reward when taking a given action a in a given state s when following policy $\pi(s) = \arg\max_{a \in A} Q^\pi(s, a)$. Since enumerating the state space is intractable for problems of a realistic nature, the Q-function is typically approximated by some parameterization θ (e.g., a linear combination of features describing the states weighted by θ), as denoted by Q_θ^π. To learn these parameters and in turn, an approximation for the Q-function one seeks to minimize the Bellman residual by applying the update rule in Equation 6, where ${}^{Q}\Delta\theta$ indicates the change in θ with time-step under Q-learning, s' is the state the agent arrives in after applying action a' in state s, and α is the step size.

$${}^{Q}\Delta\theta := \alpha \left(R(s, a) + \gamma \max_{a' \in A} Q_\theta^\pi(s', a') - Q_\theta^\pi(s, a) \right) \nabla_\theta Q_\theta^\pi(s, a) \tag{6}$$

2) Policy Gradient: In PG, the objective is to directly learn a policy, $\pi_\theta(s)$, parameterized by θ as opposed to Q-learning. The update rule seeks to maximize the expected reward of a policy, as shown in Equation 7, where ${}^{PG}\Delta\theta$ indicates the change in θ with time-step under PG, s_t and a_t are the state

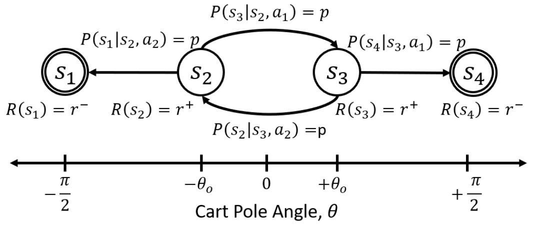


Fig. 1: Diagram of the MDP used to investigate DDTs for RL. For visual clarity, not all possible transitions are shown.

and action chosen at time t, $r_{t'}$ is the reward received at time t', and $A_{it} = \sum_{t'=t} \gamma^{t'-t} r_{t'}$ is the PG coefficient. This update considers an entire episode to compute an estimate of the expected value of the policy for each time step.

$$ {}^{PG}\Delta\theta := \alpha \sum_{t} A_{it} \nabla_\theta \log\left(\pi_\theta\left(s_t, a_t\right)\right) \tag{7} $$

III. Assessing Online Methods for DDTs: Q-Learning and Policy Gradient

A. Problem Set Up

For our investigation into using a DDT for RL, we consider an MDP with four states $S = \{s_1, s_2, s_3, s_4\}$ and two actions $A = \{a_1, a_2\}$ as depicted in Figure 1. The rewards are r^-, r^+, r^+, r^- for each state, respectively. The transition matrix, defined in Equation 8, indicates that the agent moves to a state with a higher index (i.e., $s = s+1$) when taking action a_1 and a state with a lower index (i.e., $s' = s-1$) when taking action a_2. Actions are taken successfully with probability p. We note that s_1 and s_4 are terminal states.

We optimistically assume $p = 1$; despite this hopeful assumption, we show unfavorable results for Q-learning and PG based agents using DDTs as function approximators. Further, we note that the Q-learning and PG updates do not explicitly consider transition probabilities (i.e., Equations 6 and 7). We assume the agent operating on this MDP learns episodically, meaning that the agent takes a sequence of actions until either time expires or a terminal node is reached. The agent begins in state s_3 when the time, t, is $t = 0$ (i.e., $s_o = s_3$) and takes at most four actions (i.e., $T = 3$, where T is the final time step).

Given these assumptions, one can see by inspection[1] that the optimal policy, π^*, is to apply action a_1 in state s_2 and action a_2 in state s_3.

$$ P(s'|s,a) = \begin{cases} 0, & \text{if } s' = 1,4 \\ p, & \text{if } s' = s+1, a = a_1 \\ p, & \text{if } s' = s-1, a = a_2 \\ \frac{1-p}{|S|-1}, & \text{otherwise} \end{cases} \tag{8} $$

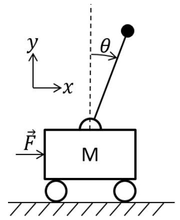


Fig. 2: Schematic of the Cart Pole domain

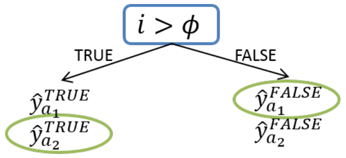


Fig. 3: A decision tree representation of the optimal policy for the MDP in Figure 1. Optimal actions are circled in green.

Remark 1 (Analogy to Cart Pole)**.** *We note that this MDP determines that when the agent is in one portion of the state space, one action should be applied; when the agent is in a different portion of the state space, a different action should be applied. This behavior is analogous to many canonical RL problems, such as Cart Pole (Figure 2). In the Cart Pole problem, there is a point mass located at the end of a pole, connected to a cart through an un-actuated joint. Gravity causes the pole to fall to the left when the pole is leaning left and right when leaning right. The RL agent must provide a counteracting force to balance the pole.*

B. Decision Trees as Function Approximators

We aim to learn a decision tree that can serve as a function approximator for Q-learning or PG. For simplicity, we consider a decision tree with one decision node and two leaf nodes, as shown in Figure 3 and defined in Equation 9. This decision tree bifurcates the state space into two: states with an index less than or equal to ϕ and those greater than ϕ.

$$ \pi_T(s,a) = \mu(s)\hat{y}_a^{\text{TRUE}} + (1-\mu(s))\,\hat{y}_a^{\text{FALSE}} \tag{9} $$

Under Q-learning, the leaf nodes return an estimate of the expected future reward (i.e., the Q-value) for applying each action when in the portion of the state space dictated by the decision node's criteria. For example, if $\phi = 2$ and the agent is in s_3 (i.e, parameter s in $T(s,a)$ is 3), the Q-values for taking actions a_1 and a_2 are $\hat{y}_{a_1}^{TRUE}$ and $\hat{y}_{a_2}^{TRUE}$, respectively. Under PG, the leaves represent an estimate of the optimal probability distribution over actions the RL agent should take to maximize its future expected reward. Therefore, the values at these leaves represent the probability of selecting the corresponding action. We note here that one would impose the constraint $\hat{y}_{a_1}^{TRUE} + \hat{y}_{a_2}^{TRUE} = 1$.

For our investigation, we assume that the decision tree's parameters are initialized to the optimal setting. As noted previously and regardless of using Q-learning or PG the optimal policy, π^*, is to apply action a_1 in state 2 and action a_2 in state s_3, assuming $p = 1$. As such, for Q-learning, we set $\hat{y}_{a_2}^{TRUE} = \hat{y}_{a_2}^{FALSE} = r^+(1+\gamma+\gamma^2+\gamma^3)$ and $\hat{y}_{a_1}^{TRUE} = \hat{y}_{a_1}^{FALSE} = r^+ + \gamma r^-$,

[1] Derivation withheld due to space constraints.

which correspond to the Q-values of taking action a_1 and a_2 in states s_2 and s_3 when otherwise following the optimal policy starting in a non-terminal node. When generating results in Section V for PG, we set $\hat{y}_{a_2}^{TRUE} = \hat{y}_{a_2}^{FALSE} = 0.99$ and $\hat{y}_{a_1}^{TRUE} = \hat{y}_{a_1}^{FALSE} = 0.01$. These settings correspond to a decision tree that focuses on exploiting the current (optimal if $\phi = 2$) policy. While varying the setting of leaf node values using PG is outside of the scope of this paper due to space considerations, we note that the results generalize to other settings of these parameters

C. Decision Tree Function Approximator Policies

There can be five qualitatively unique policies using a Boolean tree (i.e., Equation 3). These policies correspond to $\phi = 0, 1, 2, 3, 4$. For each policy, we can generate the sequence of states and the associated rewards the agent would receive, shown in Table I, assuming the agent starts in s_3. Based on this information, we compute in Table II the value function $V^{\pi_\phi}(s = 2)$ (Equation 5) for each setting ϕ. For simplicity, we assume $p = 1$.

ϕ	$t = 0$	$t = 1$	$t = 2$	$t = 3$
0	$(3, r^+, 2)$	$(2, r^+, 2)$	$(1, r^-, 2)$	
1	$(3, r^+, 2)$	$(2, r^+, 2)$	$(1, r^-, 1)$	
2	$(3, r^+, 2)$	$(2, r^+, 1)$	$(3, r^+, 2)$	$(2, r^+, 1)$
3	$(3, r^+, 1)$	$(4, r^-, 2)$		
4	$(3, r^+, 1)$	$(4, r^-, 1)$		

TABLE I: The set of execution traces for a Boolean decision tree with varying ϕ, assuming $s_o = 3$. Columns indicate increasing time, rows indicate settings for ϕ, and entries indicate $(s_t, R(s_t, a_t), a_t)$.

ϕ	$\gamma^t r^t$				$V^{\pi_\phi}(s_3)$
	$t = 0$	$t = 1$	$t = 2$	$t = 3$	
0	r^+	$r^+\gamma$	$r^-\gamma^2$		$r^+(1+\gamma) + r^-$
1	r^+	$r^+\gamma$	$r^-\gamma^2$		$r^+(1+\gamma) + r^-$
2	r^+	$r^+\gamma$	$r^+\gamma^2$	$r^+\gamma^3$	$r^+(1+\gamma+\gamma^2+\gamma^3)$
3	r^+	$r^-\gamma$			$r^+ + r^-\gamma$
4	r^+	$r^-\gamma$			$r^+ + r^-\gamma$

TABLE II: Derived from Table I, the values V^{π_ϕ} of Boolean decision tree policies π_ϕ with varying ϕ and assuming $s_o = 3$.

Remark 2 (Boolean vs. Continuous Decision Trees). *A key difference between a Boolean and differentiable decision tree is that the output of the differentiable tree is a weighted, nonlinear combination of the leaves (Equation 9). Using PG, one samples actions probabilistically from $\pi_T(s, a)$. The probability of applying the "wrong" action (i.e., one resulting in a negative reward) is $\pi_T(s_3, a_1)$ in state s_3 and $\pi_T(s_2, a_2)$ in state s_2. Assuming it equally likely to be in states s_3 and s_2, the overall probability is $\frac{1}{2}(\pi_T(s_2, a_2) + \pi_T(s_3, a_1))$. These probabilities are depicted in Figure 4, which shows how the optimal setting, ϕ^*, for ϕ should be $\phi^* = 2.5$ using PG.*

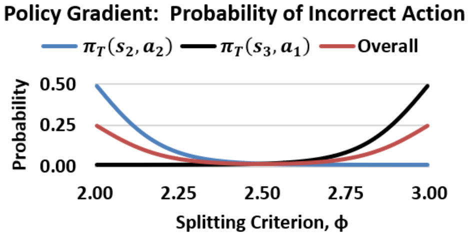


Fig. 4: This figure demonstrates the probability of taking a "wrong" action for PG with $\gamma = 0.95$, $a = 10$, $r^+ = 1$, and $r^- = -1$.

D. Computing Critical Points

To apply Q-learning and PG updates, we must compute the gradient of the DDT formulation from Equations 2 and 4. As we focus primarily on the splitting criterion, ϕ, we need only consider ${}^{\partial T}/_{\partial \phi}$ as shown in Equation 10.

$$\frac{\partial \pi_T(s, a)}{\partial \phi} = -a\mu(s)\left(1 - \mu(s)\right)\left(\hat{y}^a_{\text{TRUE}} - \hat{y}^a_{\text{FALSE}}\right) \tag{10}$$

The full Q-learning and PG updates, ${}^{Q}\Delta\phi^{(t)}$ and ${}^{PG}\Delta\phi^{(t)}$, respectively, are then given by Equations 11 and 12, where the (t) indicates the updates are dependent on a specific time step with an associated state-action pair.

$$\begin{aligned} {}^{Q}\Delta\phi^{(t)} &= \alpha \left(R(s, a) + \gamma \max_{a' \in A} Q^\pi_\phi(s', a') - Q^\pi_\phi(s, a) \right) \nabla_\phi Q^\pi_\phi(s, a) \\ &= \alpha \Bigg(R(s, a) + \gamma \max_{a' \in A} \left(\mu(s')\hat{y}_{a'}^{TRUE} - \left(1 - \mu(s')\right) \hat{y}_{a'}^{FALSE} \right) \\ &\quad - \left(\mu(s)\hat{y}_a^{TRUE} - \left(1 - \mu(s)\right) \hat{y}_a^{FALSE} \right) \Bigg) \\ &\quad \left(-a\mu(s)\left(1 - \mu(s)\right) \left(\hat{y}_a^{\text{TRUE}} - \hat{y}_a^{\text{FALSE}} \right) \right) \end{aligned} \tag{11}$$

$$\begin{aligned} {}^{PG}\Delta\phi^{(t)} &= \alpha A_{it} \nabla_\phi \log\left(\pi_T\left(s, a\right)\right) \\ &= \alpha A_{it} \frac{-a\mu(s)\left(1 - \mu(s)\right)\left(\hat{y}_a^{\text{TRUE}} - \hat{y}_a^{\text{FALSE}}\right)}{\mu(s)\hat{y}_a^{\text{TRUE}} + \left(1 - \mu(s)\right)\hat{y}_a^{\text{FALSE}}} \end{aligned} \tag{12}$$

Recall the agent experiences episodes with four time steps $(t = 0, 1, 2, 3)$. Each step generates its own update, which are combined to give the overall update $\Delta\phi$ in Equation 13.

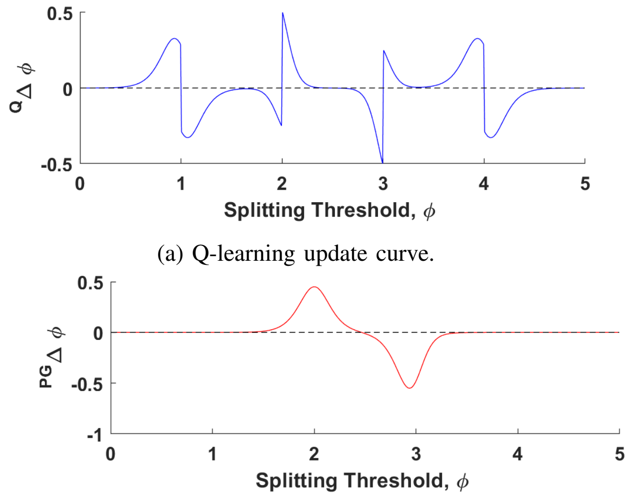
$$\Delta\phi = \sum_{t=0}^{3} \Delta\phi^{(t)} \tag{13}$$

Pseudo-critical points exist, then, whenever $\Delta\phi$ = 0. A gradient descent algorithm would treat these as extrema because the gradient update would push ϕ towards these points. As such, we consider them critical points in our analyses.

The critical points given by $\Delta\phi = 0$ are shown in Figures 5a and 5b for Q-learning and PG, respectively. For each curve, there are five critical points. We note that the curve is piece-wise curvilinear, with breaks at $\phi = 1, 2, 3$, and 4. The only true critical point exists at $\phi = 2.5$ for Q-learning and, suboptimally, at $\phi \approx 2.19$ for PG.

E. Inferring the Optimality Curve

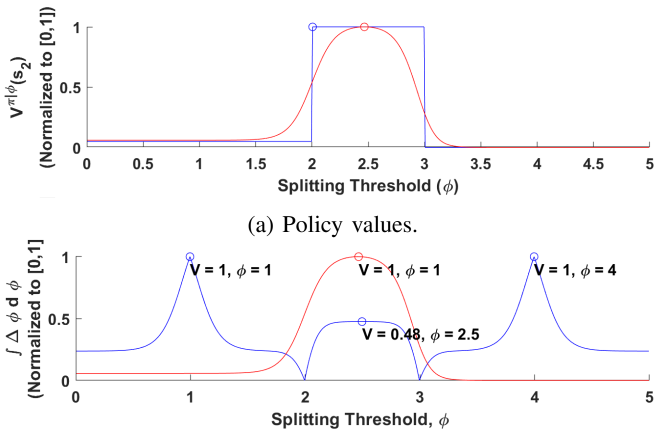
By integrating $\Delta\phi$ (Equation 14) with respect to ϕ from 0 to ϕ, we infer the "optimality curve," which should equal the value of the policy, V^{π_ϕ}, implied by Q-learning and PG. We



(a) Q-learning update curve.

(b) PG update curve.

Fig. 5: Figures 5a and 5b depict the Q-learning and PG update curves and critical points.



(a) Policy values.

(b) Optimality curves.

Fig. 6: Figures 6a and 6b depicts the policy value, V^{π} (Figure 6a), and the integrated gradient updates (Figure 6b) for Q-learning (blue) and PG (red) for the MDP depicted in Figure 1.

numerically integrate using Riemann's method as shown in Figure 6, normalized to be in $[0, 1]$.

$$\text{Optimality}(\phi) = \int_{\phi'=0}^{\phi} \Delta\phi' d\phi' \tag{14}$$

F. Evaluation of Gradient-Based Methods

Figures 5a and 5b depict the Q-learning and PG updates, respectively. We can see that the Q-learning update introduces multiple critical points as a function of the splitting criterion, ϕ, whereas the PG update includes only a single critical point for finite values of the splitting criterion, ϕ.

Figure 6a depicts the value of the DDT policy when trained using Q-learning and PG. This Figure shows the expected behavior with a maximum at $\phi = 2.5$ for both training methods. However, Figure 6b, derived using Equation 13 from the curves in Figure 5, stands in contradiction.

One would expect that the respective curves for the policy value and integrated gradient updates (i.e., Equation 13) would be identical; however, this does not hold for Q-learning. As we saw in Figure 5a, Q-learning with DDTs introduces undesired extrema (critical points in 5a), depicted by the blue curve in Figure 6b. PG, on the other hand, maintains a single maximum, coinciding with the expected $\phi = 2.5$.

The conclusion of this finding is that PG is not always superior to Q-learning as a training method for DDTs. However, this analysis does provide evidence that, even for toy problems, Q-learning exhibits weaknesses. As such, we conclude that PG serves as a more promising approach for training DDTs. Based on this evidence, the results reported below are for DDTs trained with PG.

IV. Interpretability in Full-gradient Learning

Given a training algorithm (i.e., PG as per Section III's conclusion), we now seek to address the two key drawbacks of the original DDT formulation in [22] in making the tree interpretable. First, the operation $\beta_\eta^\top x$ at each node produces a linear combination of the features (rather than a single feature) and there is a smooth transition on the feature space between the 1 and 0 states of a node (rather than a step function transition occurring at the splitting criterion). To overcome these limitations, we propose a regularizing β_η to generate decisions on a single feature and tune a_η to encourage steepness (i.e., "crispness") of the tree.

A. Modification and Regularization for Interpretability

To achieve our goal of interpretability while maintaining competitive performance, we applied three modifications to the original [22]: 1) sparsity-inducing regularization, 2) unbiased (i.e., uniform) tree initialization, and 3) discretization of the tree (i.e., to obtain the interpretable, classical version of a decision tree) at each time step to assess model performance.

1) Sparsity-inducing Regularization: During online RL via stochastic gradient descent, we added a regularization term to the loss that encouraged sparse feature representation in $\beta_\eta, \forall\eta$. Although regularization is often applied at the beginning of a training episode and increased according to some schedule, we found that allowing the model to converge in performance and then beginning the regularization procedure improved results in our tests.

$$\mathcal{L}_{reg}(\beta) = ||\text{nonmax}(\beta_\eta)||_1 - |\text{max}(\beta_\eta)| \tag{15}$$

Although equation 15 improves sparsity, if used alone, it likely multiplies the chosen feature by some amount other than the unit (i.e., $\beta_\eta^j = 1$ for a single feature j and $\beta_\eta^j = 0$ otherwise). To mitigate this problem, we applied a Softmax operator, $\zeta(\beta_n)$, to the learned beta parameter, β_n, such that the resulting decision node was governed by Equation 16 and 17. This modeling choice encouraged emphasis on a single feature.

$$\mu_\eta(x) := \frac{1}{1 + e^{-(a_\eta(\zeta(\beta_\eta)^\intercal x + \phi_\eta))}} \tag{16}$$

$$\zeta(\beta)_j = \frac{e_j^\beta}{\sum_{i=1}^{|\beta} |\beta_i} \tag{17}$$

2) Tree Initialization: By employing a sparsity-inducing regularization term, we sought to improve the tree's interpretability; however, this regularization can result in undesired under-fitting of the model by overpowering the reward signal. Through experimentation, we found that a random tree initialization introduced bias into the training process from which the tree could not recover even when allowed to converge before regularizing. To avoid this, we uniformly initialized β_η at each node such that $\beta_\eta^j = \frac{1}{|\beta_\eta|}, \forall \eta, j$.

3) Decision Sharpening: Due to the nature of the sigmoid function, even a sparse β_η was not sufficient to guarantee a discrete decision at each node. Thus, to obtain a truly discrete tree, we converted the fuzzy tree into a discrete tree at each iteration by employing an $\arg\max_j(\beta_\eta^j)$ to obtain the index of the feature of j that the node will use as in Equation 18.

$$\mu_\eta(x) := \begin{cases} 1 & \text{if} \quad x_{\arg\max(\beta_\eta)} \geq \phi_\eta \\ 0 & \text{otherwise} \end{cases} \tag{18}$$

V. Results

A. Supervised Learning

Next, as a proof of concept, we evaluated our algorithm using the Tic-Tac-Toe Endgame, the Breast Cancer Wisconsin (Diagnostic) and the Caesarian Section Classification data sets from the UCI repository [6]. For comparison we used a decision tree trained with C4.5 using the gini coefficient as the splitting criterion. For evaluation, we computed the Precision Recall Curve's AUC over three-fold cross-validation using 30% of the data for validation on each fold. The goal of this validation was not to achieve on-par performance with traditional decision-tree-learning mechanisms, which have the advantage of offline learning; rather, the goal of this intermediate investigation was to confirm that the model could learn competent mappings from input to outputs. Given such confirmation, we confidently move to demonstrate on our target task: RL.

For this intermediate supervised-learning task, we initialized the DDT to be a full binary tree of depth equal to the maximum depth of the tree generated by C4.5. Namely, for Tic-Tac-Toe a depth of 13, for Cancer a depth of 8, and for Cesarean a depth of 11.

Dataset	Cesarean	Tick-Tack-Toe	Cancer
Tree AUC	0.5032	0.8513	0.9263
Sharp DDT AUC	0.6141	0.6577	0.7948
DDT AUC	0.6077	0.7693	0.9910

TABLE III: The Precision Recall Curve's AUC for a tree trained with C4.5, A differentiable decision tree and a discretized differentiable decision tree.

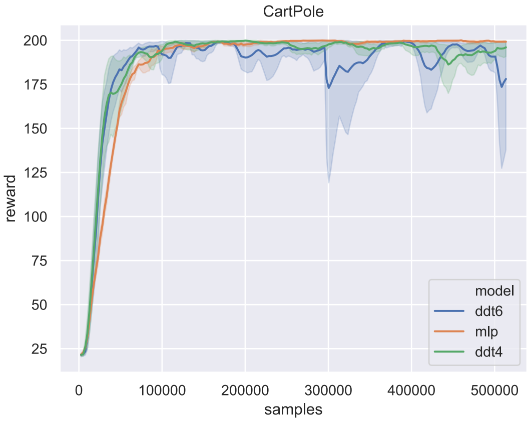


Fig. 7: Training curves for the CartPole environment. We show results for DDTs of depth 4 and 6 with regularization applied when the reward of the policy converges. The training curves above show the mean performance of each model for a particular number of samples over 5 seeds. The shaded region corresponds to one standard deviation of the performance.

B. Reinforcement Learning

Our ultimate goal is in showing DDTs can learn competent, interpretable policies online for RL tasks. To show this, we evaluated our DDT algorithm using the CartPole, LunarLander and Acrobot OpenAI Gym environments [4] using PPO [19] [13] as our chosen optimization algorithm. Since our interest is interpretability, the policies are trained on the observed environment states. For comparison, we used an MLP with one hidden layer of 256×512 hidden units. We compared the performance of full binary trees of depth 4 and depth 6. To show the variance of the policy being trained, we ran 5 seeds for each policy-environment combination. Finally, to assess the performance of each sharp tree, we computed an average reward over 10 episodes initialized to different seeds. In Figure 7, it is evident there was comparable performance to MLP in terms of sample complexity.

After training, we retained the best performing seeds for the MLP and the crisp DDTs as measured on the evaluation dataset. The performance achieved by these policies for each environment is reported in Figure 8.

1) Resulting Trees: For our most challenging domain for learning (Acrobot), we were still able to find a good, crisp policy as depicted in Figure 9. Due to space limitations, we were unable to report the crisp version of each DDT we learned.

C. Implementation

These results were gathered on decision trees implemented with PyTorch [16]. Our code is open-source and available here: https://github.com/ivandariojr/ddt.

VI. Discussion

In this paper, we provide a theoretical argument in favor of PG as a training method for DDTs over Q-learning in

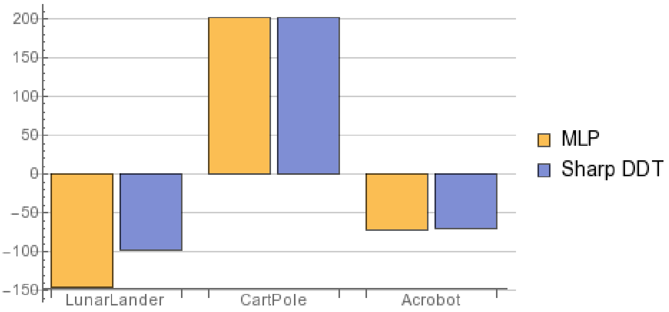


Fig. 8: The peak sharp DDT performance with the peak MLP performance for comparison. Results for the DDT are for the interpretable version of the tree. Specifically, our method provides a performance advantage compared to a regular MLP of $+33.2\%$, $+0\%$ and -2.4% for the LunarLander, CartPole and Acrobot respectively.

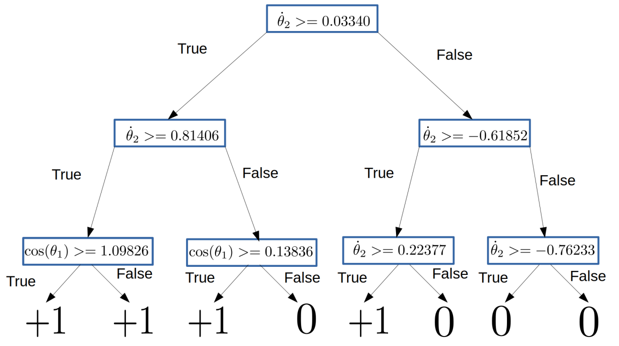

$\dot{\theta}_2 >= 0.03340$
True
False
$\dot{\theta}_2 >= 0.81406$
$\dot{\theta}_2 >= -0.61852$
True
False
True
False
$\cos(\theta_1) >= 1.09826$
$\cos(\theta_1) >= 0.13836$
$\dot{\theta}_2 >= 0.22377$
$\dot{\theta}_2 >= -0.76233$
True
False
True
False
True
False
True
False
+1
+1
+1
0
+1
0
0
0

Fig. 9: This is the best performing decision tree for the Acrobot environment. For reference the Acrobot environment's state consists of a state $S = (\cos(\theta_1), \sin(\theta_1), \cos(\theta_2), \sin(\theta_2), \dot{\theta_1}, \dot{\theta_2})$ where θ_1 and θ_2 correspond to the angles of the first and second joints respectively. The actions correspond to the discrete torques of $(+1, 0, 1)$ applied to the second joint.

the RL setting based on analysis of a simple, single-criterion, stochastic gradient descent update on the Cart Pole domain. We also provide results for fuzzy and discrete DDTs for several reinforcement learning and classification settings.

Given the flexibility of MLPs and their large number of parameters,we anticipated an advantage in raw performance. After all, our DDT's have a much smaller number of parameters, were regularized to encourage sparsity, and were constrained even further through discretization. We provide promising results that even after converting the trained DDT into a discretized (i.e., interpretable) tree the training process would yield tree policies that outperformed even the best MLP. Before triggering regularization, DDTs would even manage to improve on MLP's sample efficiency as is the case with the CartPole and the LunarLander. Similarly, in supervised learning, we found that discretized DDTs sometimes outperformed traditional decision trees.

Choosing monte-carlo sampling of seeds to explore better trees was a potential limitation, although ubiquitous in the field of RL. By effect of sampling policies every certain number of epochs, one policy could have happened to perform well on the evaluation environments, a possibility considering the high variance DDTs exhibited across seeds during training. Although possible for a very expressive policy class such as MLPs, it is exceedingly unlikely in the case of DDTs as they are designed to underfit in achieving interpretability. Further, we evaluated each discretized tree across 10 random episodes.

A hypothesis for future work, we suspect DDTs perform better in the Acrobot and CartPole scenarios because, in part, we believe Bang Bang Controllers would likely also perform well in those scenarios. Although MLPs are more expressive in a continuous domain of policies, they may be poor approximators for discrete decision boundaries. In the future, we would also like to explore ways of pruning DDTs. As seen in Figure 9, there are redundant nodes in the policy that could be removed. Finally, we aim to further analyze the convergence and performance of DDT policies.

VII. Conclusion

Ultimately, we show how differentiable decision trees can be used in the context of reinforcement learning to generate interpretable policies. We provide a motivating example for why PG should be used to train this particular policy class and demonstrate results in both classification and reinforcement learning settings.

References

[1] K Arulkumaran, MP Deisenroth, M Brundage, and AA Bharath. A brief survey of deep reinforcement learning. *IEEE Signal Processing Magazine*, 34(6):26–38, 2017.

[2] L Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[3] L Breiman, J Friedman, CJ Stone, and RA Olshen. *Classification and regression trees*. CRC press, 1984.

[4] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[5] R Clay-Williams and L Colligan. Back to basics: checklists in aviation and healthcare. *BMJ Qual Saf*, 24(7):428–431, 2015.

[6] Dua Dheeru and Efi Karra Taniskidou. UCI machine learning repository, 2017.

[7] D Ernst, P Geurts, and L Wehenkel. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6(Apr):503–556, 2005.

[8] S Finney, NH Gardiol, LP Kaelbling, and T Oates. The thing that we tried didn't work very well: deictic representation in reinforcement learning. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, pages 154–161. Morgan Kaufmann Publishers Inc., 2002.

[9] R Fletcher and MJD Powell. A rapidly convergent descent method for minimization. *The computer journal*, 6(2):163–168, 1963.

[10] A Gawande. *Checklist Manifesto, The (HB)*. Penguin Books India, 2010.

[11] AB Haynes, TG Weiser, WR Berry, SR Lipsitz, AHS Breizat, EP Dellinger, T Herbosa, S Joseph, PL Kibatala, MCM Lapitan, et al. A surgical safety checklist to reduce morbidity and mortality in a global population. *New England Journal of Medicine*, 360(5):491–499, 2009.

[12] P Kontschieder, M Fiterau, A Criminisi, and S Rota-Bulo. Deep neural decision forests. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1467–1475, 2015.

[13] Ilya Kostrikov. Pytorch implementations of reinforcement learning algorithms. https://github.com/ikostrikov/pytorch-a2c-ppo-acktr, 2018.

[14] B Letham, C Rudin, TH McCormick, D Madigan, et al. Interpretable classifiers using rules and bayesian analysis: Building a better stroke prediction model. *The Annals of Applied Statistics*, 9(3):1350–1371, 2015.

[15] C Olaru and L Wehenkel. A complete fuzzy decision tree technique. *Fuzzy sets and systems*, 138(2):221–254, 2003.

[16] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[17] LD Pyeatt, AE Howe, et al. Decision tree function approximation in reinforcement learning. In *Proceedings of the third international symposium on adaptive systems: evolutionary computation and probabilistic graphical models*, volume 2, pages 70–77. Cuba, 2001.

[18] C Rudin. Algorithms for interpretable machine learning. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1519–1519. ACM, 2014.

[19] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[20] H Shah and M Gopal. Fuzzy decision tree function approximation in reinforcement learning. *International Journal of Artificial Intelligence and Soft Computing*, 2(1-2):26–45, 2010.

[21] D Silver, A Huang, CJ Maddison, A Guez, L Sifre, G Van Den Driessche, J Schrittwieser, I Antonoglou, V Panneershelvam, M Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[22] A Suárez and JF Lutsko. Globally optimal fuzzy decision trees for classification and regression. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(12):1297–1311, 1999.

[23] RS Sutton and AG Barto. *Reinforcement learning: An introduction*. MIT Press, 1998.

[24] RS Sutton, DA McAllester, SP Singh, and Y Mansour. Policy gradient methods for reinforcement learning with function approximation. In *NIPS*, pages 1057–1063, 2000.

[25] CJCH Watkins. *Learning from delayed rewards*. PhD thesis, King's College, Cambridge, 1989.

[26] SM Weiss and N Indurkhya. Rule-based machine learning methods for functional prediction. *Journal of Artificial Intelligence Research*, 3:383–403, 1995.

Appendix

Appendix A: Derivation of the Optimal Policy

In this section, we provide a derivation of the optimal policy for the MDP in Figure 1. For this derivation, we use the definition of the Q-function described in Equation 19, where s' is the state resulting from applying action a in state s. In keeping with the investigation in this paper, we assume deterministic transitions between states (i.e., $p = 1$ from Equation 8). As such, we can ignore $P(s'|s,a)$ and simply apply Equation 20.

$$Q(s,a) := R(s,a) + \gamma \max_{a'} \sum_{s'} P(s'|s,a) Q(s',a') \tag{19}$$

$$Q(s,a) := R(s,a) + \gamma \max_{a'} Q(s',a') \tag{20}$$

Theorem 1. *The optimal policy for the MDP in Figure 1 is to apply action a_1 in state s_2 and action a_2 in state s_3 assuming deterministic transitions between states (i.e., $p = 1$ from Equation 8).*

Proof. We begin by asserting in Equation 21 that the Q-values for $Q(s,a)$ are $r^=$ given $s \in \{1,4\}$ and for any action $a \in \{a_1,a_2\}$. This result is due to the definition that states s_1 and s_4 are terminal states and the reward for those states is r^- regardless of the action applied.

$$Q(s_1,a_1) = Q(s_1,a_2) = Q(s_4,a_1) = Q(s_4,a_2) = r^- \tag{21}$$

Next, we must compute the Q-values for the remaining state-action pairs, as shown in Equations 22-25.

$$Q(s_2,a_1) = R(s_2,a_1) + \gamma \max\{Q(s_3,a_1), Q(s_3,a_2)\} \tag{22}$$

$$Q(s_2,a_2) = R(s_2,a_1) + \gamma \max\{Q(s_1,a_1), Q(s_1,a_2)\} \tag{23}$$

$$Q(s_3,a_1) = R(s_3,a_1) + \gamma \max\{Q(s_4,a_1), Q(s_4,a_2)\} \tag{24}$$

$$Q(s_3,a_2) = R(s_3,a_2) + \gamma \max\{Q(s_2,a_1), Q(s_2,a_2)\} \tag{25}$$

By the definition of the MDP in Figure 1, we substitute in for $R(s_2,a_1) = R(s_2,a_2) = R(s_3,a_1) = R(s_3,a_2) = r^+$ as shown in Equations 26-29.

$$Q(s_2,a_1) = r^+ + \gamma \max\{Q(s_3,a_1), Q(s_3,a_2)\} \tag{26}$$

$$Q(s_2,a_2) = r^+ + \gamma r^- \tag{27}$$

$$Q(s_3,a_1) = r^+ + \gamma r^- \tag{28}$$

$$Q(s_3,a_2) = r^+ + \gamma \max\{Q(s_2,a_1), Q(s_2,a_2)\} \tag{29}$$

We can substitute in for $Q(s_3,a_1)$ and $Q(s_2,a_2)$ given Equations 26 and 29.

$$Q(s_2,a_1) = r^+ + \gamma \max\{\left(r^+ + \gamma r^-\right), Q(s_3,a_2)\} \tag{30}$$

$$Q(s_3,a_2) = r^+ + \gamma \max\{Q(s_2,a_1), \left(r^+ + \gamma r^-\right)\} \tag{31}$$

For the Q-value of state-action pair, $Q(s_2,a_1)$, we must determine whether $(r^+ + \gamma r^-)$ is less than or equal to $Q(s_3,a_2)$. If the agent were to apply action a_2 in state s_3, we can see from Equation 31 that the agent would receive at a minimum $Q(s_3,a_2) \geq r^+ + \gamma\left(r^+ + \gamma r^-\right)$, because $r^+ + \gamma\left(r^+ + \gamma r^-\right) > r^+ + \gamma r^-$, $Q(s_3,a_2)$ must be the maximum from Equation 30. We can make a symmetric argument for $Q(s_3,a_2)$ in Equation 31. Given this relation, we arrive at Equations 32 and 33.

$$Q(s_2,a_1) = r^+ + \gamma Q(s_3,a_2) \tag{32}$$

$$Q(s_3,a_2) = r^+ + \gamma Q(s_2,a_1) \tag{33}$$

Equations 32 and 33 represent a recursive, infinite geometric series, as depicted in Equation 35.

$$\begin{aligned} Q(s_2,a_1) = Q(s_3,a_2) &= r^+ + \gamma r^+ + \gamma^2 r^+ + \ldots \\ &= r^+\left(\gamma^0 + \gamma + \gamma^2 + \ldots\right) \qquad (34) \\ &= r^+ \sum_{t=0}^{T} \gamma^t \qquad (35) \end{aligned}$$

In the case that $T = \infty$, Equation 35 represents an infinite geometric series, the solution to which is $\frac{r^+}{1+\gamma}$. In our case however, $T = 3$ (i.e., four-time steps). As such, $Q(s_2,a_1) = Q(s_3,a_2) = r^+(1 + \gamma + \gamma^2 + \gamma^3)$, as shown in Equation 36.

$$Q(s_2,a_1) = Q(s_3,a_2) = r^+(1 + \gamma + \gamma^2 + \gamma^3) \tag{36}$$

Recall that $r^- < 0$ given our definition of the MDP in Figure 1. Therefore, $Q(s_2,a_1) = Q(s_3,a_2) = \frac{r^+}{1-\gamma} \geq Q(s_2,a_2) = Q(s_3,a_1) = r^+ + \gamma r^-$. If the RL agent is non-myopic, i.e., $\gamma \in (0,1]$, then we have the strict inequality $Q(s_2,a_1) = Q(s_3,a_2) > Q(s_2,a_2) = Q(s_3,a_1)$. For these non-trivial settings of γ, we can see that the optimal policy for the RL agent is to apply action a_1 in state s_2 and action a_2 in state s_3. Lastly, because s_1 and s_4 are terminal states, the choice of action is irrelevant, as seen in Equation 21. □

The optimal policy is then given by Equation 37.

$$\pi^*(s,a) = \begin{cases} 1, & \text{if } s = 2, a_1 \text{ or } s = 3, a_2 \\ 0, & \text{if } s = 2, a_2 \text{ or } s = 3, a_1 \\ 1/2, & \text{otherwise} \end{cases} \tag{37}$$

Appendix B: Q-learning Leaf Values

For the decision tree in Figure 3, there are four leaf values: $\hat{y}_{a_2}^{TRUE}$, $\hat{y}_{a_1}^{TRUE}$, $\hat{y}_{a_2}^{FALSE}$, and $\hat{y}_{a_1}^{FALSE}$. Table IV contains the settings of those parameters. In Table IV, the first column depicts the leaf parameters; the second column depicts the Q-function state-action pair; the third column contains the equation reference to Appendix A, where the Q-value is calculated; and the fourth column contains the corresponding Q-value. These Q-values assume that the agent begins in a non-terminal state (i.e., s_2 or s_3) and follows the optimal policy represented by Equation 37.

Leaf	Q-function	Equation	Q-value
$\hat{y}_{a_2}^{FALSE}$	$Q(s_2,a_2)$	Equation 27	$r^+ + \gamma r^-$
$\hat{y}_{a_1}^{TRUE}$	$Q(s_3,a_1)$	Equation 28	$r^+ + \gamma r^-$
$\hat{y}_{a_1}^{FALSE}$	$Q(s_2,a_1)$	Equation 36	$r^+(1 + \gamma + \gamma^2 + \gamma^3)$
$\hat{y}_{a_2}^{TRUE}$	$Q(s_3,a_2)$	Equation 36	$r^+(1 + \gamma + \gamma^2 + \gamma^3)$

TABLE IV: Derived from Table I, the values V^{π_ϕ} of Boolean decision tree policies π_ϕ with varying ϕ and assuming $s_o = 3$.