arXiv:1903.09338v3 [cs.LG] 27 Jan 2020

Optimization Methods for Interpretable Differentiable Decision Trees in Reinforcement Learning

Andrew Silva
Georgia Institute of Technology

Taylor Killian
University of Toronto

Ivan Rodriguez Jimenez
Georgia Institute of Technology

Sung-Hyun Son
MIT Lincoln Laboratory

Matthew Gombolay
Georgia Institute of Technology

Abstract

Decision trees are ubiquitous in machine learning for their ease of use and interpretability. Yet, these models are not typically employed in reinforcement learning as they cannot be updated online via stochastic gradient descent. We overcome this limitation by allowing for a gradient update over the entire tree that improves sample complexity affords interpretable policy extraction. First, we include theoretical motivation on the need for policy-gradient learning by examining the properties of gradient descent over differentiable decision trees. Second, we demonstrate that our approach equals or outperforms a neural network on all domains and can learn discrete decision trees online with average rewards up to 7x higher than a batch-trained decision tree. Third, we conduct a user study to quantify the interpretability of a decision tree, rule list, and a neural network with statistically significant results ($p < 0.001$).

1 Introduction and Related Work

Reinforcement learning (RL) with neural network function approximators, known as "Deep RL," has achieved tremendous results in recent years [Andrychowicz et al., 2018, 2016, Arulkumaran et al., 2017, Espeholt et al., 2018, Mnih et al., 2013, Sun et al., 2018, Rajeswaran et al., 2017]. Deep RL uses multi-layered neural networks to represent policies trained to maximize an agent's expected future reward. Unfortunately, these neural-network-based approaches are largely uninterpretable due to the millions of parameters involved and nonlinear activations throughout.

Proceedings of the 23rdInternational Conference on Artificial Intelligence and Statistics (AISTATS) 2020, Palermo, Italy. PMLR: Volume 108. Copyright 2020 by the author(s).

In safety-critical domains, e.g., healthcare and aviation, insight into a machine's decision-making process is of utmost importance. Human operators must be able to follow step-by-step procedures [Clay-Williams and Colligan, 2015, Gawande, 2010, Haynes, 2009]. Of the machine learning (ML) methods able to generate such procedures, decision trees are among the most highly developed [Weiss and Indurkhya, 1995], persisting in use today [Gombolay et al., 2018a, Zhang et al., 2019]. While interpretable ML methods offer much promise [Letham et al., 2015], they are unable to match the performance of Deep RL [Finney, 2002, Silver, 2016]. In this paper, we advance the state of the art in decision tree methods for RL and leverage their ability to yield interpretable policies.

Decision trees are viewed as the de facto technique for interpretable and transparent ML [Rudin, 2014, Lipton, 2018], as they learn compact representations of relationships within data [Breiman et al., 1984]. Rule [Angelino et al., 2017, Chen and Rudin, 2017] and decision lists [Lakkaraju and Rudin, 2017, Letham et al., 2015] are related architectures also used to communicate a decision-making process. Decision trees have been also applied to RL problems where they served as function approximators, representing which action to take in which state [Ernst and Wehenkel, 2005, Finney, 2002, Pyeatt and Howe, 2001, Shah and Gopal, 2010].

The challenge for decision trees as function approximators lies in the online nature of the RL problem. The model must adapt to the non-stationary distribution of the data as the model interacts with its environment. The two primary techniques for learning through function approximation, Q-learning [Watkins, 1989] and policy gradient [Sutton and Mansour, 2000], rely on online training and stochastic gradient descent [Bottou, 2010, Fletcher and Powell, 1963]. Standard decision trees are not amenable to gradient descent as they are a collection of non-differentiable, nested, if-then rules. As such, researchers have used non-gradient-descent-based methods for training decision trees for RL [Ernst and Wehenkel, 2005, Finney, 2002, Pyeatt and Howe, 2001], e.g., greedy state aggregation, rather than seeking to update the entire model with respect to a global loss function [Pyeatt and Howe, 2001]. Researchers have also attempted to use decision trees for RL by training in batch mode, completely re-learning the tree from scratch to account for the non-stationarity introduced by an improving policy [Ernst and Wehenkel, 2005]. This approach is inefficient when scaling to realistic situations and is not guaranteed to converge. Despite these attempts, success comparable to that of modern deep learning approaches has been elusive [Finney, 2002].

In this paper, we present an novel function approximation technique for RL via differentiable decision trees (DDTs). We provide three contributions. First, we examine the properties of gradient descent over DDTs, motivating policy-gradient-based learning. To our knowledge, this is the first investigation of the optimization surfaces of Q-learning and policy gradients for DDTs. Second, we compare our method with baseline approaches on standard RL challenges, showing that our approach parities or outperforms a neural network; further, the interpretable decision trees we discretize after training achieve an average reward up to 7x higher than a batch-learned decision tree. Finally, we conduct a user study to compare the interpretability and usability of each method as a decision-making aid for humans, showing that discrete trees and decision lists are perceived as more helpful ($p < 0.001$) and are objectively more efficient ($p < 0.001$) than a neural network.

Remark 1 (Analysis Significance) *Our approach builds upon decades of work in machine and RL; yet ours is the first to consider DDTs for online learning. While researchers have shown failings of Q-learning with function approximation, including for sigmoids [Baird, 1995, Bertsekas and Tsitsiklis, 1996, Gordon, 1995, Tsitsiklis and Van Roy, 1996], we are unaware of analysis of Q-learning and policy gradient for our unique architecture. Our analysis provides insight regarding the best practices for training interpretable RL policies with DDTs.*

2 Preliminaries

In this section, we review decision trees, DDTs, and RL.

2.1 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 $\hat{y}$ (Equations 1-3).

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

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

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

There are two node types: *decision* and *leaf* nodes, which have an outdegree of two and zero, respectively. Nodes have an indegree of one except for the root, η_o, whose indegree is zero. Decision nodes η are represented as Boolean expressions, μ_η (Eq. 3), where x_{j_η} and ϕ_η are the selected feature and splitting threshold for decision node η. For each decision node, the left outgoing edge is labeled "true," and the right outgoing edge is labeled "false." E.g., if μ_η is evaluated true, the *left child* node, $\eta_\swarrow$, is considered next. The process repeats until a leaf is reached upon which the tree returns the corresponding label. The goal is to determine the best j^*_η, ϕ^*_η, and y_η for each node and the best structure (i.e., whether, for each η, there exists a child). There are many heuristic techniques for learning decision trees with a batch data [Breiman et al., 1984]. However, one cannot apply gradient updates as the tree is fixed at generation. While some have sought to grow trees for RL [Pyeatt and Howe, 2001], these approaches do not update the entire tree.

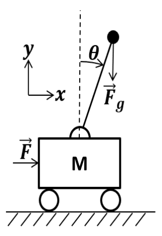
DDTs – Suárez and Lutsko provide one of the first DDT models. Their method replaces the Boolean decision in Eq. 3 with the sigmoid activation function shown in Eq. 4. This function considers a linear combination of features x weighted by β_η compared to a bias value ϕ_η, and augmented by a steepness parameter a_η. The tree is trained via gradient descent for, ϕ_η, β_η, and a_η across nodes η [Suárez and Lutsko, 1999]. This method has been applied to offline, supervised learning but not RL.

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

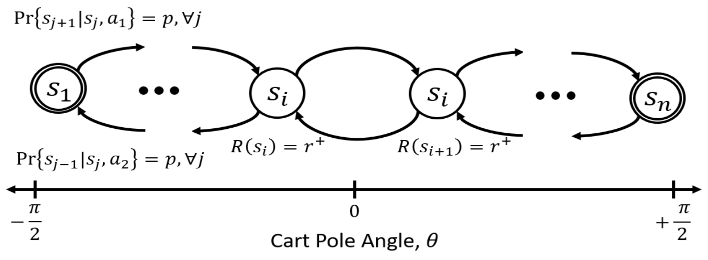
2.2 Reinforcement Learning

RL is a subset of machine learning in which an agent is tasked with learning the optimal action sequence that maximizes future expected reward [Sutton and Barto, 1998]. The problem is abstracted as a Markov Decision Process (MDP), which 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; $P : S \times A \times S \to [0, 1]$ is the transition matrix describing the probability that taking action $a \in A$ in state $s \in S$ results in state $s' \in S$; $\gamma \in [0, 1]$ is the discount factor defining the trade-off between immediate and future reward; and $R : S \times A \to \mathbb{R}$ is the function dictating the reward an agent receives by taking action $a \in A$ in state $s \in S$. The goal is to learn a policy, $\pi : S \to A$, that prescribes which action to take in each state to maximize the agent's long-term expected reward. There are two ubiquitous approaches to learn a policy: Q-learning and policy gradient.

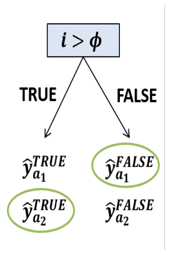
Q-learning seeks to to learn a mapping, $Q^\pi : S \times A \to \mathbb{R}$, that returns the expected future reward when taking action a in state s. This mapping (i.e., the Q-function) is typically approximated by a parameterization θ (e.g., a neural network), Q_θ. One then minimizes the Bellman residual via



(a)



(b)



(c)

Figure 1: Fig. 1a depicts the cart pole analogy for our analysis. Fig. 1b depicts the MDP model for our analysis. Fig. 1c depicts a tree representation of the optimal policy for our analysis, with optimal actions circled.

Eq. 5, where ${}^{Q}\Delta\theta$ is the estimated change in θ, and s_{t+1} is the state the agent arrives in after applying action a_t in state s_t at time step t with learning rate α.

$$\begin{aligned} {}^{Q}\Delta\theta := \alpha \Big(& R(s_t, a_t) + \gamma \max_{a' \in A} Q_\theta(s_{t+1}, a') \\ & - Q_\theta(s_t, a_t) \Big) \nabla_\theta Q_\theta^\pi(s_t, a_t) \end{aligned} \tag{5}$$

A complementary approach is the set of policy gradient methods in which one seeks to directly learn a policy, $\pi_\theta(s)$, parameterized by θ, that maps states to actions. The update rule maximizes the expected reward of a policy, as shown in Eq. 6, where ${}^{PG}\Delta\theta$ indicates the change in θ for a timestep under policy gradient and $A_t = \sum_{t'=t}^{T} \gamma^{(T-t')} r_{t'}$. T is the length of the trajectory.

$${}^{PG}\Delta\theta := \alpha \sum_t A_t \nabla_\theta \log \left(\pi_\theta\left(s_t, a_t\right)\right) \tag{6}$$

We provide an investigation into the behavior of ${}^{Q}\Delta\theta$ and ${}^{PG}\Delta\theta$ as for DDTs in Section 5.

3 DDTs as Interpretable Function Approximators

In this section, we derive the Q-learning and policy gradient updates for DDTs as function approximators in RL. Due to space considerations, we show the simple case of a DDT with a single decision node and two leaves with one feature s with feature coefficient β, as shown in Eq. 7 with the gradient shown in Eq. 8-12.

$$f_T(s,a) = \mu(s)\hat{y}_a^{\text{TRUE}} + (1-\mu(s))\,\hat{y}_a^{\text{FALSE}} \tag{7}$$

$$\nabla f_T(s,a) = \left[\frac{\partial f_T}{\partial \hat{y}_a^{TRUE}}, \frac{\partial f_T}{\partial \hat{y}_a^{FALSE}}, \frac{\partial f_T}{\partial \alpha}, \frac{\partial f_T}{\partial \beta}, \frac{\partial f_T}{\partial \phi}\right]^\intercal \tag{8}$$

$$\frac{\partial f_T}{\partial \hat{y}_a^{TRUE}} = 1 - \frac{\partial f_T}{\partial \hat{y}_a^{FALSE}} = \mu(s) \tag{9}$$

$$\frac{\partial f_T}{\partial \alpha} = (\hat{q}_a^{TRUE} - \hat{q}_a^{FALSE})\mu(s)(1-\mu(s))(\beta s - \phi) \tag{10}$$

$$\frac{\partial f_T}{\partial \beta} = (\hat{q}_a^{TRUE} - \hat{q}_a^{FALSE})\mu(s)(1-\mu(s))(a)(s) \tag{11}$$

$$\frac{\partial f_T}{\partial \phi} = (\hat{q}_a^{TRUE} - \hat{q}_a^{FALSE})\mu(s)(1-\mu(s))(a)(-1) \tag{12}$$

When utilizing a DDT as a function approximator for Q-learning, each leaf node returns 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 criterion of it's parent node (Eq. 13).

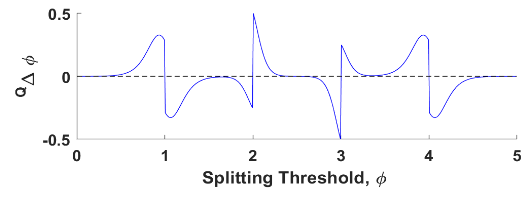
$$f_T(s,a) \rightarrow Q(s,a) = \mu(s)\hat{q}_a^{\text{TRUE}} + (1-\mu(s))\,\hat{q}_a^{\text{FALSE}} \tag{13}$$

Likewise, when leveraging policy gradient methods for RL with DDT function approximation, 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 (Eq. 14). We impose the constraint that the probabilities of all actions sum to one ($\hat{y}_{a_1}^{TRUE} + \hat{y}_{a_2}^{TRUE} = 1$).

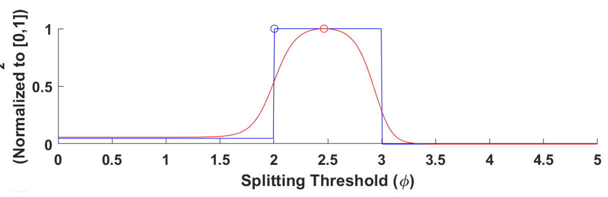
$$f_T(s,a) \rightarrow \pi(s,a) = \mu(s)\hat{\pi}_a^{\text{TRUE}} + (1-\mu(s))\,\hat{\pi}_a^{\text{FALSE}} \tag{14}$$

4 Interpretability for Online Learning

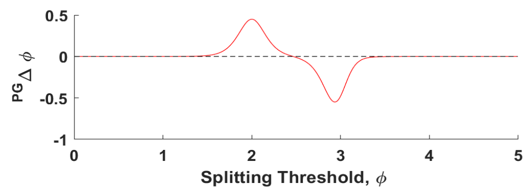
We seek to address the two key drawbacks of the original DDT formulation by Suárez and Lutsko [Suárez and Lutsko, 1999] in making the tree interpretable. First, the operation $\beta_\eta^\top x$ at each node produces a linear combination of the features, rather than a single feature comparison. Second, use of the sigmoid activation fuction means that there is a smooth transition between the $TRUE$ and $FALSE$ evaluations of a node, rather than a discrete decision. We address these limitations below; we demonstrate the extensibility of our approach by also differentiating over a rule list architecture [Letham et al., 2015] and extracting interpretable rule lists. Using the mechanisms from Sections 4.1 and 4.2, we produce interpretable policies for empirical evaluation (Section 6) and a user study (Section 7).



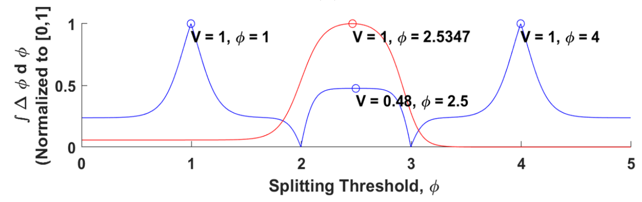
(a)



(b)



(c)



(d)

Figure 2: Figures 2a and 2c depict the Q-learning and policy gradient update curves, respectively. Figures 2b and 2d depict the policy value, V^π (Figure 2b), and the integrated gradient updates (Figure 2d) for Q-learning (blue) and policy gradient (red) for the MDP depicted in Figure 1b.

4.1 Discretizing the Differentiable Decision Tree

Due to the nature of the sigmoid function, even a sparse β_η is not sufficient to guarantee a discrete decision at each node. Thus, to obtain a truly discrete tree, we convert the differentiable tree into a discrete tree by employing an $\arg\max_j(\beta^j_\eta)$ to obtain the index of the feature of j that the node will use. We set β_η to a one-hot vector, with a 1 at index j and 0 elsewhere. We also divide ϕ_η by the node's weight β^j_η, normalizing the value for comparison against the raw input feature x_j. Each node then compares a single raw input feature to a single ϕ_η, effectively converting from Eq. 4 back into Eq. 3. We repeat this process for each decision node, obtaining discrete splits throughout the tree. Finally, each leaf node must now return a single action, as in an ordinary decision tree. We again employ an $\arg\max_j(\beta^j_\eta)$ on each leaf node and set the leaves to be one-hot vectors with $\beta^j_\eta = 1$ and all other values set to 0. The result of this process is an interpretable decision tree with discrete decision nodes, a single feature comparison per node, and a single decision output per leaf.

4.2 Differentiable Rule Lists

In addition to discretizing the optimized tree parameterization, we also consider a specific sub-formulation of tree proposed by [Letham et al., 2015] to be particularly interpretable: the rule- or decision-list. This type of tree restricts the symmetric branching allowed for in Eq. 1 by stating that the TRUE branch from a decision node leads directly to a leaf node. We define a discrete rule list according to Eq. 15.

$$T_\eta(x) := \begin{cases} y_\eta, & \text{if leaf} \\ \mu_\eta(x) y_{\eta_\swarrow} + (1-\mu_\eta(x)) T_{\eta_\searrow}(x) & \text{o/w} \end{cases} \quad (15)$$

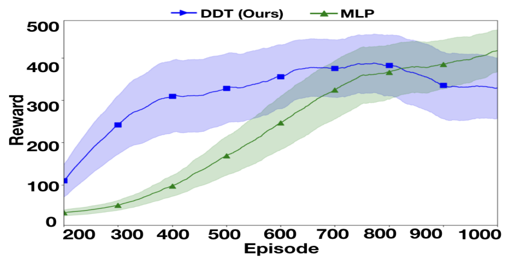
In Section 6, we demonstrate that these mechanisms for interpretability achieves high-quality policies for online RL and are consistent with the the legal [Voigt and Von dem Bussche, 2017] and practical criteria for interpretability [Doshi-Velez and Kim, 2017, Letham et al., 2015].

5 Analysis of Gradient Methods for DDTs

In this section, we analyze Q-learning and policy gradient updates for DDTs as function approximators in RL, providing a theoretical basis for how to best deploy DDTs to RL. We show that Q-learning introduces additional critical points that impede learning where policy gradient does not. This analysis guides us to recommend policy gradient for these interpretable function approximators and yields high-quality policies (Section 6).

5.1 Analysis: Problem Setup

We consider an MDP with states $S = \{s_1, s_2, ..., s_n\}$ and actions $A = \{a_1, a_2\}$ (Fig. 1b). The agent moves to a state with a higher index (i.e., $s = s + 1$) when taking action a_1 with probability p and $1-p$ for transitioning to a lower index. The opposite is the case for action a_2. Within Fig. 1b, a_1 corresponds to "move right" and a_2 corresponds to "move left." The terminal states are s_1 and s_n. The rewards are zero for each state except for $R(s_{i^*}) = R(s_{i^*+1}) = +1$ for some i^* such that $1 < i^* < n-1$. It follows that the optimal policy, π^*, is $\pi(s) = a_1$ ("move right") in s_j such that $1 \leq j \leq i^*$ and $\pi(s) = a_2$ otherwise. A proof is given in supplementary material. We optimistically assume $p = 1$; despite this hopeful assumption, we show unfavorable results for Q-learning and policy-gradient-based agents using DDTs as function approximators.



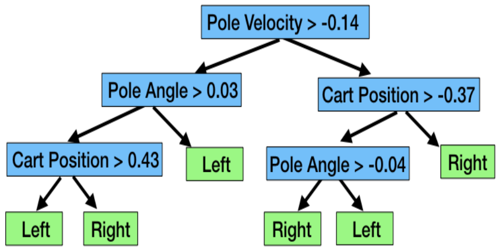


Figure 3: Training curves for the cart pole domain (left), and the resulting discrete decision tree (right)

5.2 Analysis: Tree Initialization

For our investigation, we assume that the decision tree's parameters are initialized to the optimal setting. Given our MDP setup, we only have one state feature: the state's index. As such, we only have two degrees of freedom in the decision node: the steepness parameter, α, and the splitting criterion, ϕ. In our analysis, we focus on this splitting criterion, ϕ, showing that even for an optimal tree initialization, ϕ is not compelled to converge to the optimal setting $\phi = i^* + \frac{1}{2}$. We set the leaf nodes as follows for Q-learning and policy gradient given the optimal policy.

For Q-learning, we set the discounted optimal action reward as $\hat{q}_{a_2}^{TRUE} = \hat{q}_{a_1}^{FALSE} = \sum_{t=0}^{\infty} \gamma^t r^+ = \frac{r^+}{1-\gamma}$, which assumes $s_o = s_{i^*}$. Likewise, we set $\hat{y}_{a_1}^{TRUE} = \hat{y}_{a_2}^{FALSE} = \frac{r^+}{1-\gamma} - r^+ - r^+\gamma$, 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.

For policy gradient, we set $\hat{y}_{a_2}^{TRUE} = \hat{y}_{a_1}^{FALSE} = 0.99$ and $\hat{y}_{a_1}^{TRUE} = \hat{y}_{a_2}^{FALSE} = 0.01$. These settings correspond to a decision tree that focuses on exploiting the current (optimal if $\phi = i^*$) policy. While we consider this setting of parameters for our analysis of DDTs, the results generalize.

5.3 Computing Critical Points

The ultimate step in our analysis is to assess whether Q-learning or policy gradient introduces critical points that do not coincide with global extrema. To do so, we can set Eq. 5 and 6 to zero, with $\nabla Q(s,a) = \nabla f_T s, a$ from Eq. 13 and $\nabla \pi(s,a) = \nabla f_T(s,a)$ from Eq. 14, respectively. We would then solve for our parameter(s) of interest and determine whether any zeros lie at local extrema. In our case, focusing on the splitting criterion, ϕ, is sufficient to show the weaknesses of Q-learning for DDTs.

Rather than exactly solving for the zeros, we use numerical approximation for these Monte Carlo updates (Eq. 5 and 6). In this setting, we recall that the agent experiences episodes with T timesteps. Each step generates its own update, which are combined to give the overall update $\Delta\phi = \sum_{t=0}^{T} \Delta\phi^{(t)}$. Pseudo-critical points exist, then, whenever $\Delta\phi = 0$. A gradient descent algorithm would treat these as extrema, and the gradient update would push ϕ towards these points. As such, we consider these "critical points."

5.4 Numerical Analysis of the Updates

The critical points given by $\Delta\phi = 0$ are shown in Figures 2a and 2c for Q-learning and PG, respectively. For the purpose of illustration, we set $n = 4$ (i.e., the MDP has four states). As such, $i^* = 2$ and the optimal setting for $\phi = \phi^* = i^* + \frac{1}{2} = 2.5$.

For Q-learning, there are five critical points, only one of which is coincident with $\phi = \phi^* = i^* + \frac{1}{2}$. For PG, there are fewer, with a single critical point in the domain of $\phi \in (-\infty, \infty)$, which occurs at $\phi \approx 2.465$[1]. Thus, we can say that the expectation of the critical point for a random, symmetric initialization is $\mathbb{E}_{s_o \sim U(2,3)}[\Delta\phi = 0 | s_o] = i^* + \frac{1}{2}$, which supports the adoption of policy gradient as an approach for DDTs.

Additionally, by integrating $\Delta\phi$ with respect to ϕ from 0 to ϕ, i.e., Optimality$(\phi) = \int_{\phi'=0}^{\phi} \Delta\phi' d\phi'$, we infer the "optimality curve," which should equal the value of the policy, V^{π_ϕ}, implied by Q-learning and policy gradient. We numerically integrate using Riemann's method normalized to $[0, 1]$. One would expect that the respective curves for the policy value (Fig. 2b) and integrated gradient updates (Fig. 2d) would be identical; however, this does not hold for Q-learning. Q-learning with DDT function approximation introduces undesired extrema, shown by the blue curve in Fig. 2d. Policy gradient, on the other hand, maintains a single maximum coincident with $\phi = \phi^* = i^* + \frac{1}{2} = 2.5$.

This analysis provides evidence that Q-learning exhibits weaknesses when applied to DDT models, such as an excess of critical points which serve to impede gradient descent. We therefore conclude that policy gradient is a more promising approach for learning the parameters of DDTs and proceed accordingly. As such, we have shown that Q-learning with DDT function approximators introduces additional extrema that policy gradients, under the same conditions, do not, within our MDP case study.

This analysis provides the first examination of the potential pitfalls and failings of Q-learning with DDTs. We believe

[1]We recall that, for this analysis, $s_o = i^*$; if we set $s_o = i^* + 1$ (i.e., the symmetric position with respect to vertical), this critical point for policy gradient is $\phi = 2.535$.

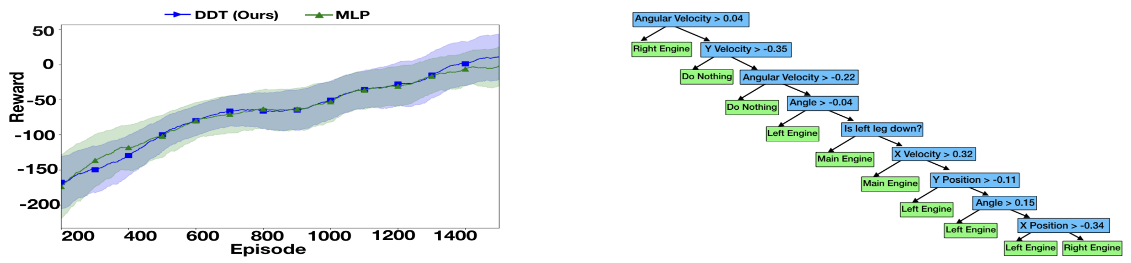


Figure 4: Training curves for the lunar lander domain (left), and the resulting discrete rule list (right)

that this helpful analysis will guide researchers in the application of these function approximators. Given this analysis and our mechanisms for interpretability (Section 4), we now show convincing empirical results (Section 6) of the power of these function approximators to achieve high-quality and interpretable policies in RL.

6 Demonstration of DDTs for Online RL

Our ultimate goal is to show that DDTs can learn competent, interpretable policies online for RL tasks. To demonstrate this, we evaluate our DDT algorithm using the cart pole and lunar lander OpenAI Gym environments [Brockman et al., 2016], a simulated wildfire tracking problem, and the FindAndDefeatZerglings mini-game from the StarCraft II Learning Environment [Vinyals et al., 2017]. All agents are trained via Proximal Policy Optimization (PPO) [Kostrikov, 2018, Schulman et al., 2017]. We use a multilayer perceptron (MLP) architecture as a baseline for performance across all tasks. We provide further details on the evaluation domains below, as well as examples of extracted interpretable policies, trained using online RL with DDTs. Due to space constraints, we present pruned versions of the interpretable policies in which redundant nodes are removed for visual clarity. The full policies are in the supplementary material.

We conduct a sensitivity analysis comparing the performance of MLPs with DDTs (DDTs) across a range of depths. For the trees, the set of leaf nodes we consider is {2, 4, 8, 16, 32}. For comparison, we run MLP agents with between {0, 1, 2, 4, 8, 16, 32} hidden layers, and a rule-list architecture with {1, 2, 4, 8, 16, 32} rules. Results from this sensitivity analysis are given in Figures 11 & 12 in the supplementary material. We find that MLPs succeed only with a narrow subset of architectures, while DDTs and rule lists are more robust. In this section, we present results from the agents that obtained the highest average cumulative reward in our sensitivity analysis. Table 1 compares mean reward of the highest-achieving agents and shows the mean reward for our discretization approach applied to the best agents. For completeness, we also compare against standard decision trees which are fit using scikit-learn [Pedregosa, 2011] on a set of state-action pairs generated by the best-performing model in each domain, which we call *State-Action DT*.

In our OpenAI Gym [Brockman et al., 2016] environments we use a learning rate of 1e-2, and in our wildfire tracking and FindAndDefeatZerglings [Vinyals et al., 2017] domains we use a learning rate of 1e-3. All models are updated with the RMSProp [Tieleman and Hinton, 2012] optimizer. All hyperparameters are included in the supplementary material.

6.1 Open AI Gym Evaluation

We plot the performance of the best agent for each architecture in our OpenAI Gym [Brockman et al., 2016] experiments, as well as pruned interpretable policies, in Figures 3 and 4. To show the variance of the policies, we run five seeds for each policy-environment combination. Given the flexibility of MLPs and their large number of parameters, we anticipate an advantage in raw performance. We find that the DDT offers competitive or even superior performance compared to the MLP baseline, and even after converting the trained DDT into a discretized, interpretable tree, the training process yields tree policies that are competitive with the best MLP. Our interpretable approach yields a 3x and 7x improvement over a batch-trained decision tree (DT) on lunar lander and cart pole, respectively. Table 1 depicts the average reward across domains and agents.

6.2 Wildfire Tracking

Wildfire tracking is a real-world problem in which interpretability is critical to an agent's human teammates. While RL is a promising approach to develop assistive agents in wildfire monitoring [Haksar and Schwager, 2018], it is important to maintain trust between these agents and humans in this dangerous domain. An agent that can explicitly give its policy to a human firefighter is therefore highly desirable.

We develop a Python implementation of the simplified FARSITE [Finney, 1998] wildfire propagation model. The environment is a 500x500 map in which two fires propagate slowly from the southeast end of the map to the northwest end of the map and two drones are randomly instantiated in the map. Each drone receives a 6D state containing distances to each fire centroid and Boolean flags indicating which fire the drone is closer to. The RL agent is duplicated at the start of each episode and applied to each drone, and the drones do not have any way of communicating. The challenge is to identify which fire is closest to the drone, and to then take action to get as close as possible to that

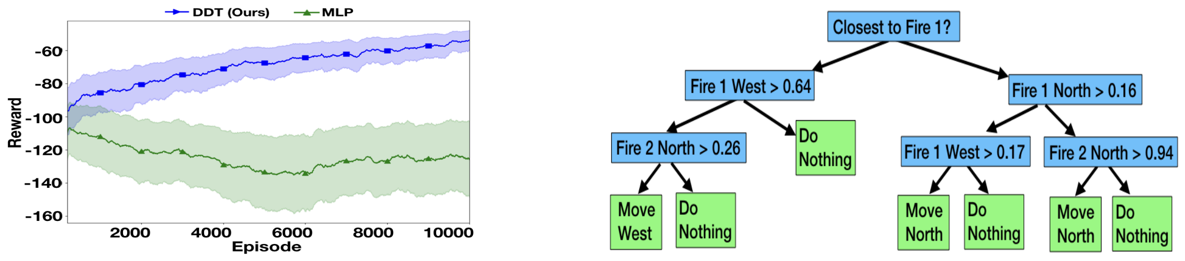


Figure 5: Training curves for the wildfire tracking environment (left) and the resulting discrete decision tree (right)

Table 1: Average cumulative reward for top models across methods and domains. Bold denotes highest-performing method.

Agent Type	Cart Pole	Lunar Lander	Wildfire Tracking	FindAndDefeatZerglings
DDT Balanced Tree (ours)	**500 $\pm$ 0**	**97.9 $\pm$ 10.5**	**-32.0 $\pm$ 3.8**	6.6 $\pm$ 1.1
DDT Rule List (ours)	**500 $\pm$ 0**	84.5 $\pm$ 13.6	-32.3 $\pm$ 4.8	**11.3 $\pm$ 1.4**
MLP	**500 $\pm$ 0**	87.7 $\pm$ 21.3	-86.7 $\pm$ 9.0	6.6 $\pm$ 1.2
Discretized DDT (ours)	**499.5 $\pm$ 0.8**	-88 $\pm$ 20.4	**-36.4 $\pm$ 2.6**	**4.2 $\pm$ 1.6**
Discretized Rule List (ours)	414.4 $\pm$ 63.9	**-78.4 $\pm$ 32.2**	-39.8 $\pm$ 1.8	0.7 $\pm$ 1.3
State-Action DT	66.3 $\pm$ 18.5	-280.9 $\pm$ 60.6	-67.9 $\pm$ 7.9	-3.0 $\pm$ 0.0

fire centroid, with the objective of flying above the fires as they progress across the map. Available actions include four move commands (north, east, south, west) and a “do nothing” command. The reward function is the negative distance from drones to fires, given in Eq. 16 where D is a distance function, d_i are the drones, and f_i are the fires.

$$\begin{aligned} R = & -\min\left[(D(d_1, f_1), D(d2, f1))\right] \\ & -\min\left[D(d1, f2), D(d2, f2)\right] \end{aligned} \tag{16}$$

The reward over time for the top performing DDT and MLP agents is given in Fig. 5, showing the DDT significantly outperforms the MLP. We also present the interpretable policy for the best DDT agent, which has the agent neglect the south and east actions, instead checking for north and west distances and moving in those directions. This behavior reflects the dynamics of the domain, in which the fire always spreads from southeast to northwest. The best interpretable policy we learn is $\approx$2x better than the best batch-learned tree and $>$2x better than the best MLP.

6.3 StarCraft II Micro-battle Evaluation

To further evaluate the DDT and discretized tree, we use the FindAndDefeatZerglings minigame from the StarCraft II Learning Environment [Vinyals et al., 2017]. For this challenge, three allied units explore a partially observable map and defeat as many enemy units as possible within three minutes. We assign each allied unit a copy of the same learning agent. Rather than considering the image input and keyboard and mouse output, we manufacture a reduced state-action space. The input state is a 37D vector of allied and visible enemy state information, and the action space is 10D consisting of move and attack commands. More information is in supplementary material.

As we can see in Fig. 6, our DDT agent is again competitive with the MLP agent and is $>$2x better than a batch-learned decision tree. The interpretable policy for the best DDT agent reveals that the agent frequently chooses to attack, and never moves in conflicting directions. This behavior is intuitive, as the three allied units should stay grouped to increase their chances of survival. The agent has learned not to send units in conflicting directions, instead moving units southwest while attacking enemies en route.

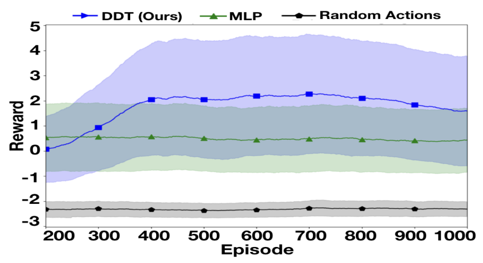
7 Interpretability Study

To emphasize the interpretability afforded by our approach, we conducted a user study in which participants were presented with policies trained in the cart pole domain and tasked with identifying which decisions the policies would have made given a set of state inputs. We compared interpretability between a discrete decision tree, a decision list, and a one-hot MLP without activation functions.

7.1 Study Setup

We designed an online questionnaire to survey 15 participants, giving each a discretized DDT, a discretized decision list, and a sample one-hot MLP. The discretized policies are actual policies from our experiments, presented in Table 1. Rather than include the full MLP, which is available in the supplementary material, we binarized the weights, thereby make the calculation much easier and less frustrating for participants. This mechanism is similar to current approaches to interpretability with deep networks that use attention [Serrano and Smith, 2019] so that human operators can see what the agent is considering when it makes a decision.

After being given a policy, participants were presented with five sample domain states. They were then asked to trace the



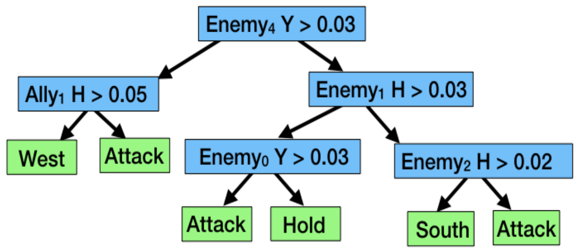


Figure 6: Training curves for the FindAndDefeatZerglings environment (left) and the resulting discrete decision tree (right)

policy with a given input state and predict what the agent would have done. After predicting which decisions the agent would have made, participants were presented with a set of Likert scales assessing their feelings on the interpretability and usability of the given policy as a decision-making aid. We timed participants for each method.

We hypothesize that: **H1:** *A decision-based classifier is more interpretable than an MLP*; **H2:** *A decision-based classifier is more efficient than a MLP.* To test these hypotheses, we report on participant Likert scale ratings (**H1**) and completion time for each task (**H2**).

7.2 Study Results

Results of our study are shown in Fig. 7. We perform an ANOVA and find that the type of decision-making aid had a statistically significant effect on users' Likert scale ratings for usability and interpretability ($F(2, 28) = 19.12, p < 0.0001$). We test for normality and homoscedasticity and do not reject the null hypothesis in either case, using Shapiro-Wilk ($p > 0.20$) and Levene's Test ($p > 0.40$), respectively. A Tukey's HSD post-hoc test shows that the tree ($t = 6.02$, $p < 0.0001$) and decision list ($t = 4.24$, $p < 0.0001$) both rated significantly higher than a one-hot MLP.

We also test the time participants took to use each decision-making aid for a set of five prompts. We applied Friedman's test and found the type of aid had a significant effect on completion time ($Q(2) = 26, p < 0.0001$). Dunn's test showed that the tree ($z = -4.07$, $p < 0.0001$) and decision list ($z = -5.23$, $p < 0.0001$) times were statistically significantly shorter than the one-hot MLP completion times.

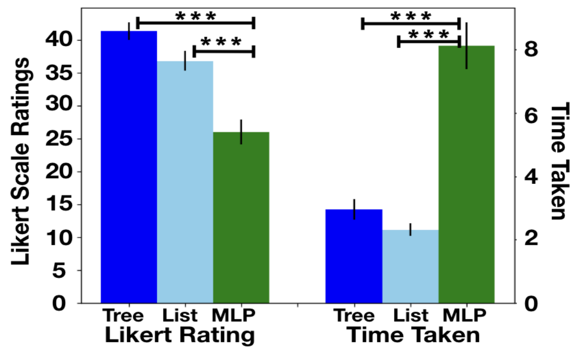


Figure 7: Results from our user study. Higher Likert ratings are better, lower time taken is better.

We note that participants were shown the full MLP after the questionnaire's conclusion, and participants consistently reported they would have abandoned the task if they had been presented with a full MLP as their aid. These results support the hypothesis that decision trees and lists are significantly superior decision-making aids in reducing human frustration and increasing efficiency. This study, coupled with our strong performance results over MLPs, shows the power of our approach to interpretable, online RL via DDTs.

8 Future Work

We propose investigating how our framework could leverage advances in other areas of deep learning, e.g. inferring feature embeddings. For example, we could learn subject-specific embeddings via backpropagation but within an interpretable framework for personalized medicine [Killian et al., 2017] or in apprenticeship learning [Gombolay et al., 2018b], particularly when heterogeneity precludes a one-size-fits-all model [Chen et al., 2020]. We could also invert our learning process to a prior specify a decision tree policy given expert knowledge, which we could then train via policy gradient [Silva and Gombolay, 2019].

9 Conclusion

We demonstrate that DDTs can be used in RL to generate interpretable policies. We provide a motivating analysis showing the benefit of using policy gradients to train DDTs in RL challenges over Q-learning. This analysis serves to guide researchers and practitioners alike in future research and application of DDTs to learn interpretable RL policies. We show that DDTs trained with policy gradient can provide comparable and even superior performance against MLP baselines. Finally, we conduct a user study which demonstrates that DDTs and decision lists offer increased interpretability and usability over MLPs while also taking less time and providing efficient insight into policy behavior.

10 Acknowledgements

This work was funded by the Office of Naval Research under grant N00014-19-1-2076 and by MIT Lincoln Laboratory under grant # MIT Lincoln Laboratory grant 7000437192.

References

Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems*, pages 3981–3989, 2016.

Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. *arXiv preprint arXiv:1808.00177*, 2018.

Elaine Angelino, Nicholas Larus-Stone, Daniel Alabi, Margo Seltzer, and Cynthia Rudin. Learning certifiably optimal rule lists for categorical data. *The Journal of Machine Learning Research*, 18(1):8753–8830, 2017.

Kai Arulkumaran, Mark Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. A brief survey of deep reinforcement learning. *IEEE Signal Processing Magazine*, 34(6):26–38, 2017.

Leemon Baird. Residual algorithms: Reinforcement learning with function approximation. In *Machine Learning Proceedings 1995*, pages 30–37. Elsevier, 1995.

Dimitri P Bertsekas and John N Tsitsiklis. *Neuro-dynamic programming*, volume 5. Athena Scientific Belmont, MA, 1996.

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

Leo Breiman, Jerome Friedman, Charles Stone, and Richard Olshen. *Classification and regression trees*. CRC press, 1984.

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

Chaofan Chen and Cynthia Rudin. An optimization approach to learning falling rule lists. *arXiv preprint arXiv:1710.02572*, 2017.

Letian Chen, Rohan Paleja, Muyleng Ghuy, and Matthew Gombolay. Joint goal and strategy inference across heterogeneous demonstrators via reward network distillation. In *Proceedings of the International Conference on Human-Robot Interaction*, 2020.

Robynn Clay-Williams and Lacey Colligan. Back to basics: checklists in aviation and healthcare. *BMJ Qual Saf*, 24 (7):428–431, 2015.

Finale Doshi-Velez and Been Kim. Towards a rigorous science of interpretable machine learning. *arXiv preprint arXiv:1702.08608*, 2017.

Pierre Geurts Ernst, Damien and Louis Wehenkel. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6(Apr):503–556, 2005.

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymir Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. *arXiv preprint arXiv:1802.01561*, 2018.

Gardiol Natalia H. Kaelbling Leslie Pack Oates Tim Finney, Sara. 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.

Mark A Finney. Farsite: Fire area simulator-model development and evaluation. *Res. Pap. RMRS-RP-4, Revised 2004. Ogden, UT: US Department of Agriculture, Forest Service, Rocky Mountain Research Station. 47 p.*, 4, 1998. URL `https://www.fs.fed.us/rm/pubs/rmrs_rp004.pdf`.

Roger Fletcher and Michael JD Powell. A rapidly convergent descent method for minimization. *The computer journal*, 6(2):163–168, 1963.

Atul Gawande. *Checklist Manifesto, The (HB)*. Penguin Books India, 2010.

Matthew Gombolay, Reed Jensen, Jessica Stigile, Toni Golen, Neel Shah, Sung-Hyun Son, and Julie Shah. Human-machine collaborative optimization via apprenticeship scheduling. *Journal of Artificial Intelligence Research*, 63:1–49, 2018a.

Matthew Gombolay, Xi Jessie Yang, Bradley Hayes, Nicole Seo, Zixi Liu, Samir Wadhwania, Tania Yu, Neel Shah, Toni Golen, and Julie Shah. Robotic assistance in the coordination of patient care. *The International Journal of Robotics Research*, 37(10):1300–1316, 2018b.

Geoffrey J Gordon. Stable function approximation in dynamic programming. In *Machine Learning Proceedings 1995*, pages 261–268. Elsevier, 1995.

Ravi N Haksar and Mac Schwager. Distributed deep reinforcement learning for fighting forest fires with a network of aerial robots. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1067–1074. IEEE, 2018.

Thomas G. Weiser William R. Berry Stuart R. Lipsitz Abdel-Hadi S. Breizat E. Patchen Dellinger Teodoro Herbosa et al. Haynes, Alex B. A surgical safety checklist to reduce morbidity and mortality in a global population. *New England Journal of Medicine*, 360(5):491–499, 2009.

Taylor W Killian, Samuel Daulton, George Konidaris, and Finale Doshi-Velez. Robust and efficient transfer learning with hidden parameter markov decision processes. In *Advances in Neural Information Processing Systems*, pages 6250–6261, 2017.

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

Himabindu Lakkaraju and Cynthia Rudin. Learning cost-effective and interpretable treatment regimes. In *Artificial Intelligence and Statistics*, pages 166–175, 2017.

Benjamin Letham, Cynthia Rudin, Tyler H McCormick, David 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.

Zachary C Lipton. The mythos of model interpretability. *Queue*, 16(3):31–57, 2018.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Gaël Varoquaux Alexandre Gramfort Vincent Michel Bertrand Thirion Olivier Grisel Mathieu Blondel et al. Pedregosa, Fabian. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Larry D. Pyeatt and Adele E. Howe. 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.

Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *arXiv preprint arXiv:1709.10087*, 2017.

Cynthia 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.

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

Sofia Serrano and Noah A Smith. Is attention interpretable? *arXiv preprint arXiv:1906.03731*, 2019.

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

Andrew Silva and Matthew Gombolay. Prolonets: Neural-encoding human experts' domain knowledge to warm start reinforcement learning. *arXiv preprint arXiv:1902.06007*, 2019.

Aja Huang Chris J. Maddison Arthur Guez Laurent Sifre George Van Den Driessche Julian Schrittwieser et al. Silver, David. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

Alberto Suárez and James F Lutsko. Globally optimal fuzzy decision trees for classification and regression. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(12):1297–1311, 1999.

Peng Sun, Xinghai Sun, Lei Han, Jiechao Xiong, Qing Wang, Bo Li, Yang Zheng, Ji Liu, Yongsheng Liu, Han Liu, et al. Tstarbots: Defeating the cheating level builtin ai in starcraft ii in the full game. *arXiv preprint arXiv:1809.07193*, 2018.

David A. McAllester Satinder P. Singh Sutton, Richard S. and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *NIPS*, pages 1057–1063, 2000.

Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: An introduction.* MIT Press, 1998.

Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

John N Tsitsiklis and Benjamin Van Roy. Feature-based methods for large scale dynamic programming. *Machine Learning*, 22(1-3):59–94, 1996.

Oriol Vinyals, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John Agapiou, Julian Schrittwieser, et al. Starcraft ii: A new challenge for reinforcement learning. *arXiv preprint arXiv:1708.04782*, 2017.

Paul Voigt and Axel Von dem Bussche. The eu general data protection regulation (gdpr). *A Practical Guide, 1st Ed., Cham: Springer International Publishing*, 2017.

Christopher John Cornish Hellaby Watkins. *Learning from delayed rewards*. PhD thesis, King's College, Cambridge, 1989.

Sholom M. Weiss and Nitin Indurkhya. Rule-based machine learning methods for functional prediction. *Journal of Artificial Intelligence Research*, 3:383–403, 1995.

Quanshi Zhang, Yu Yang, Haotian Ma, and Ying Nian Wu. Interpreting cnns via decision trees. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6261–6270, 2019.

Appendix A: Derivation of the Optimal Policy

In this section, we provide a derivation of the optimal policy for the MDP in Fig. 1b. For this derivation, we use the definition of the Q-function described in Eq. 18, 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 Eq. 17). As such, we can ignore $P(s'|s,a)$ and simply apply Eq. 19.

$$P(s'|s,a) = \begin{cases} 0, & \text{if } s' \in 1,4 \\ p, & \text{if } (s' = s+1, a = a_1) \\ & \vee (s' = s-1, a = a_2) \\ \frac{1-p}{|S|-1}, & \text{otherwise} \end{cases} \tag{17}$$

$$Q(s,a) := R(s,a) + \gamma \max_{a'} \sum_{s'} P(s'|s,a) Q(s',a') \tag{18}$$

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

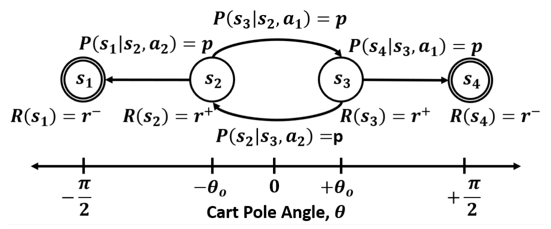


Figure 8

Theorem 1 *The optimal policy for the MDP in Fig. 8 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 Eq. 17).*

We begin by asserting in Eq. 20 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. We note that, in our example, $r^- = 0$, but we leave it here for the sake of generality.

$$Q(s_1,a_1) = Q(s_1,a_2) = Q(s_4,a_1) = Q(s_4,a_2) = r^- \tag{20}$$

Next, we must compute the Q-values for the remaining state-action pairs, as shown in Eq. 21-24.

$$Q(s_2,a_1) = R(s_2,a_1) + \gamma \max\{Q(s_3,a_1), Q(s_3,a_2)\} \tag{21}$$

$$Q(s_2,a_2) = R(s_2,a_1) + \gamma \max\{Q(s_1,a_1), Q(s_1,a_2)\} \tag{22}$$

$$Q(s_3,a_1) = R(s_3,a_1) + \gamma \max\{Q(s_4,a_1), Q(s_4,a_2)\} \tag{23}$$

$$Q(s_3,a_2) = R(s_3,a_2) + \gamma \max\{Q(s_2,a_1), Q(s_2,a_2)\} \tag{24}$$

By the definition of the MDP in Fig. 8, 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 Eq. 25-28.

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

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

$$Q(s_3,a_1) = r^+ + \gamma r^- \tag{27}$$

$$Q(s_3,a_2) = r^+ + \gamma \max\{Q(s_2,a_1), Q(s_2,a_2)\} \tag{28}$$

We can substitute in for $Q(s_3,a_1)$ and $Q(s_2,a_2)$ given Eq. 25 and 28.

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

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

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 Eq. 30 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 Eq. 29. We can make a symmetric argument for $Q(s_3,a_2)$ in Eq. 30. Given this relation, we arrive at Eq. 31 and 32.

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

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

Eq. 31 and 32 represent a recursive, infinite geometric series, as depicted in Eq. 34.

$$\begin{aligned} Q(s_2,a_1) = Q(s_3,a_2) &= r^+ + \gamma r^+ + \gamma^2 r^+ + \dots \\ &= r^+\left(\gamma^0 + \gamma + \gamma^2 + \dots\right) \quad (33) \\ &= r^+ \sum_{t=0}^{T} \gamma^t \quad (34) \end{aligned}$$

In the case that $T = \infty$, Eq. 34 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,

Table 2: 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	$t=0$	$t=1$	$t=2$	$t=3$
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 3: Derived from Table 2, the values V^{π_ϕ} of Boolean decision tree policies π_ϕ with varying ϕ and assuming $s_o = 3$.

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^+	$r^-\gamma$			$r^+ + r^-\gamma$

$Q(s_2, a_1) = Q(s_3, a_2) = r^+(1+\gamma+\gamma^2+\gamma^3)$, as shown in Eq. 35.

$$Q(s_2, a_1) = Q(s_3, a_2) = r^+(1+\gamma+\gamma^2+\gamma^3) \tag{35}$$

Recall that $r^- < 0$ given our definition of the MDP in Fig. 8. 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 Eq. 20.

The optimal policy is then given by Eq. 36.

$$\pi^*(s,a) = \begin{cases} 1, & \text{if } s = 2, a_1 \text{ or } s = 3, a_2 \\ 0, & \text{if } s = 2, a_2 \text{ or } s = 3, a_1 \\ 1/2, & \text{otherwise} \end{cases} \tag{36}$$

Appendix B: Policy Traces and Value

This section reports the execution traces and corresponding value calculations of a Boolean decision treee with varying ϕ for the simple MDP model from Fig. 8.

Appendix C: Q-learning Leaf Values

For the decision tree in Fig. 9, 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 4 contains the settings of those parameters. In Table 4, 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 Eq. 36.

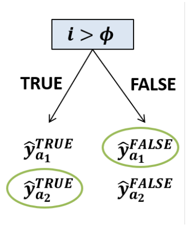


Figure 9: This figure depicts the tree for our case study.

Table 4: Derived from Table 2, the values V^{π_ϕ} of Boolean decision tree policies π_ϕ with varying ϕ and assuming $s_o = 3$.

Leaf	Q-function	Eq.	Q-value
$\hat{y}_{a_2}^{FALSE}$	$Q(s_2, a_2)$	Eq. 26	$r^+ + \gamma r^-$
$\hat{y}_{a_1}^{TRUE}$	$Q(s_3, a_1)$	Eq. 27	$r^+ + \gamma r^-$
$\hat{y}_{a_1}^{FALSE}$	$Q(s_2, a_1)$	Eq. 35	$r^+(1+\gamma+\gamma^2+\gamma^3)$
$\hat{y}_{a_2}^{TRUE}$	$Q(s_3, a_2)$	Eq. 35	$r^+(1+\gamma+\gamma^2+\gamma^3)$

Appendix D: Probability of Incorrect Action

$$\pi_T(s,a) = \mu(s)\hat{y}_a^{\text{TRUE}} + (1-\mu(s))\,\hat{y}_a^{\text{FALSE}} \tag{37}$$

The output of the differentiable tree is a weighted, nonlinear combination of the leaves (Eq. 37). 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 $\frac{1}{2}\left(\pi_T(s_2,a_2)+\pi_T(s_3,a_1)\right)$. These probabilities are depicted in Fig. 10, which shows how the optimal setting, ϕ^*, for ϕ should be $\phi^* = 2.5$ using PG.

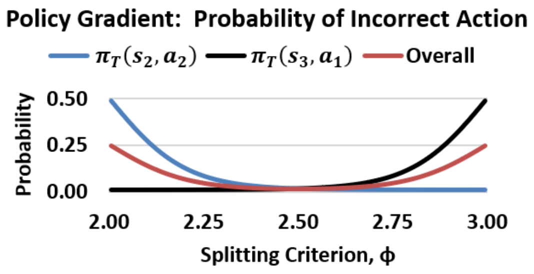


Figure 10: This figure demonstrates the probability of taking a "wrong" action for PG with $\gamma = 0.95$, $a = 10$, $r^+ = 1$, and $r^- = -1$.

Appendix E: Architecture Sweeps

We performed architecture sweeps, as mentioned in the main paper, across all types of models. We found that the MLP requires small models for simple domains, the DDT methods are all relatively unaffected by increased depth, representing a benefit of applying DDTs to various RL tasks. For this result, see Fig. 11. As shown in Fig. 12, in more complex domains, the results are less conclusive and increased depth does not show clear trends for any approach. Nonetheless, we show evidence that DDTs are at least competitive with MLPs for RL tasks of varying complexity, and that they are more robust to hyperparameter tuning with respect to depth and number of layers.

We find that the MLP with no hidden layers performs the best on the two OpenAI Gym domains, cart pole and lunar lander. The best differentiable decision tree architectures for the cart pole domain are those with two leaves and two rules, while the best architectures for lunar lander include 32 leaves and 16 rules.

In the wildfire tracking domain, the 8-layer MLP performed the best of the MLPs, while the 32-leaf differentiable decision tree was the top differentiable decision tree, and the 32-rule differentiable rule list performed the best of the differentiable rule lists.

Finally, the MLP in the FindAndDefeatZerglings domain is an 8-layer MLP, and the differentiable decision tree uses 8 leaves while the differentiable rule list uses 8 rules.

MLP hidden layer sizes preserve the input data dimension through all hidden layers until finally downsampling to the action space for the final layer. MLP networks all use the ReLU activation after it performed best in a hyperparameter sweep.

Appendix F: Domain Details

.1 Wildfire Tracking

The wildfire tracking domain input space is:

- Fire 1 Distance North (float)
- Fire 1 Distance West (float)
- Closest To Fire 1 (Boolean)
- Fire 2 Distance North (float)
- Fire 2 Distance West (float)
- Closest To Fire 2 (Boolean)

Distance features are floats, representing how far north or west the fire is, relative to the drone. Distances can also be negative, implying that the fire is south or east of the drone.

.2 StarCraft II Micro-battle Evaluation

The FindAndDefeatZerglings manufactured input space is:

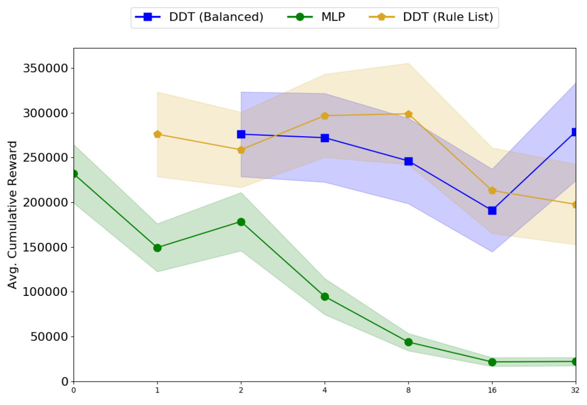
- X Distance Away (float)
- Y Distance Away (float)
- Percent Health Remaining (float)
- Percent Weapon Cooldown Remaining (float)

for each agent-controlled unit and 2 allied units, as well as:

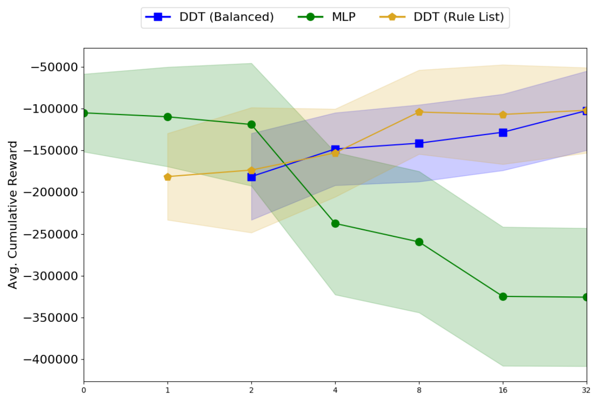
- X Distance Away (float)
- Y Distance Away (float)
- Percent Health Remaining (float)
- Percent Weapon Cooldown Remaining (float)
- Enemy Unit Type (Boolean)

for the five nearest enemy units. Missing data is filled in with -1. The action space for this domain consists of:

- Move North
- Move East
- Move West
- Move South
- Attack Nearest Enemy
- Attack Second Nearest Enemy
- Attack Third Nearest Enemy
- Attack Second Farthest Enemy
- Attack Farthest Enemy
- Do Nothing

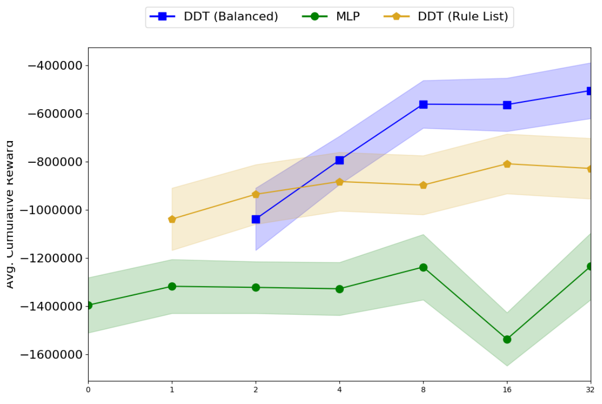


(a) Cart Pole

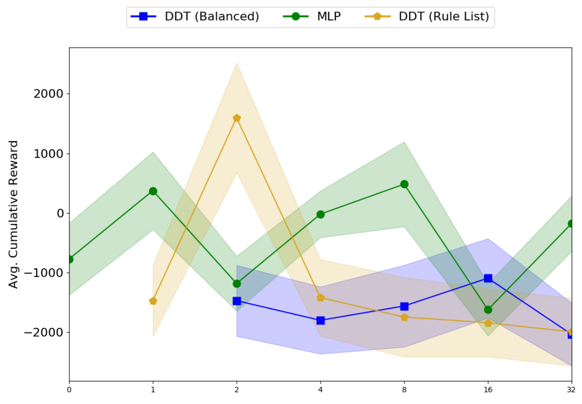


(b) Lunar Lander

Figure 11: Average cumulative reward and standard deviation across architectures of various sizes in the Gym domains. MLP with number of hidden layers, DDT (Rule List) with number of rules, and DDT (Balanced) with number of leaves.



(a) Wildfire Tracking



(b) FindAndDefeatZerglings

Figure 12: Average cumulative reward and standard deviation across architectures of various sizes in the wildfire and SC2 domains. MLP with number of hidden layers, DDT (Rule List) with number of rules, and DDT (Balanced) with number of leaves.

Interpretable Policies

Here we include interpretable policies for each domain, without the pruning that is included in versions in the main body. See Figures 13, 14, 15, and 16. Finally, we also include examples of two MLPs represented as decision-making aids. The first is the one-hot MLP that was given to study participants for evaluation of interpretability and efficiency, shown in Fig. 17. The second is the true cart pole MLP, available in Fig. 18. This decision-making aid turned out to be exceptionally complicated, even with no activation functions and no hidden layer.

Sample Survey Questions

Survey questions included Likert scale questions ranging from 1 (Very Strongly Disagree) to 7 (Very Strongly Agree). For both the MLP and decision trees, some questions included:

1. I understand the behavior represented within the model.
2. The decision-making process does not make sense.
3. The model's logic is easy to follow
4. I like the level of readability of this model.
5. The model is difficult to understand.

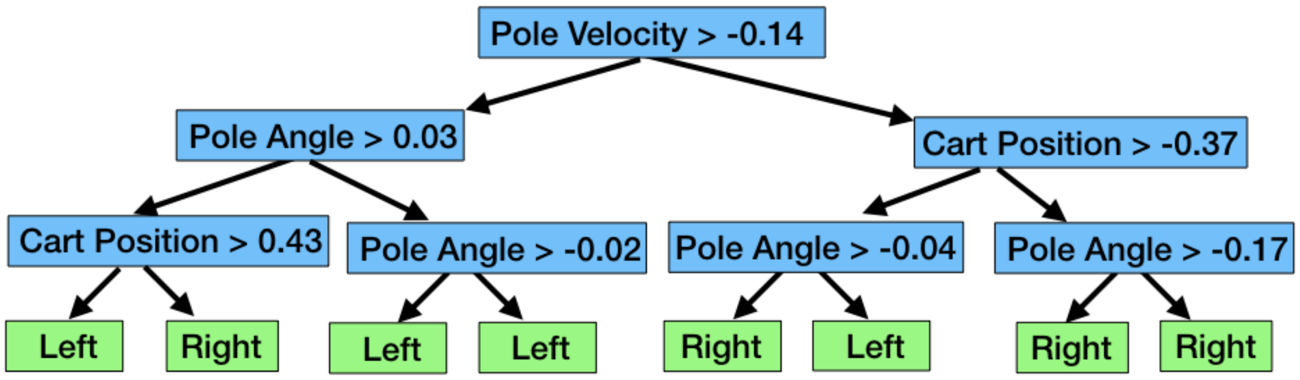


Figure 13: Full interpretable cart pole policy. Two decision nodes are redundant, leading to the same action regardless of how the node is evaluated.

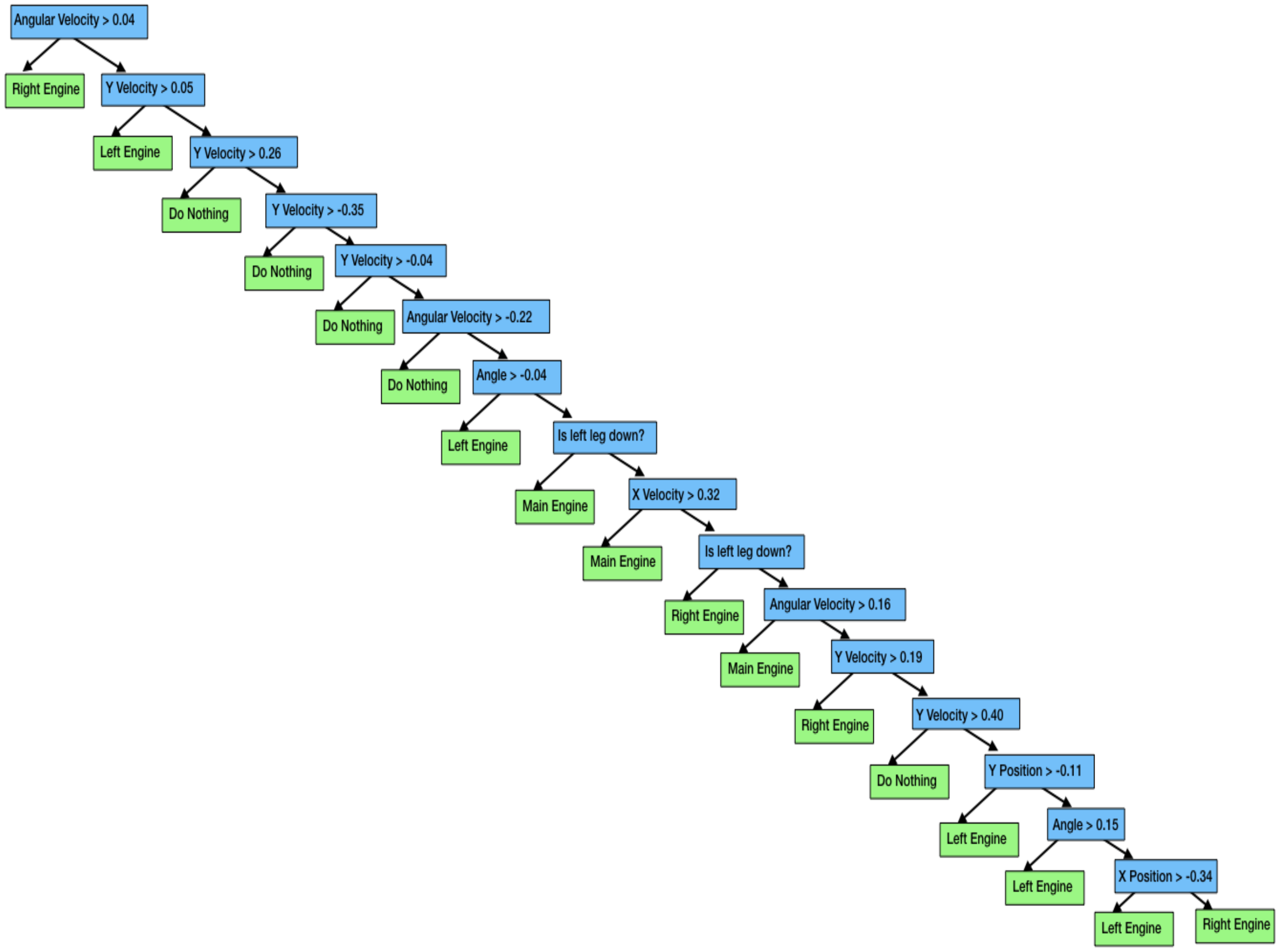


Figure 14: Full interpretable lunar lander rule list policy. Many nodes in the list are not reachable due to previous nodes.

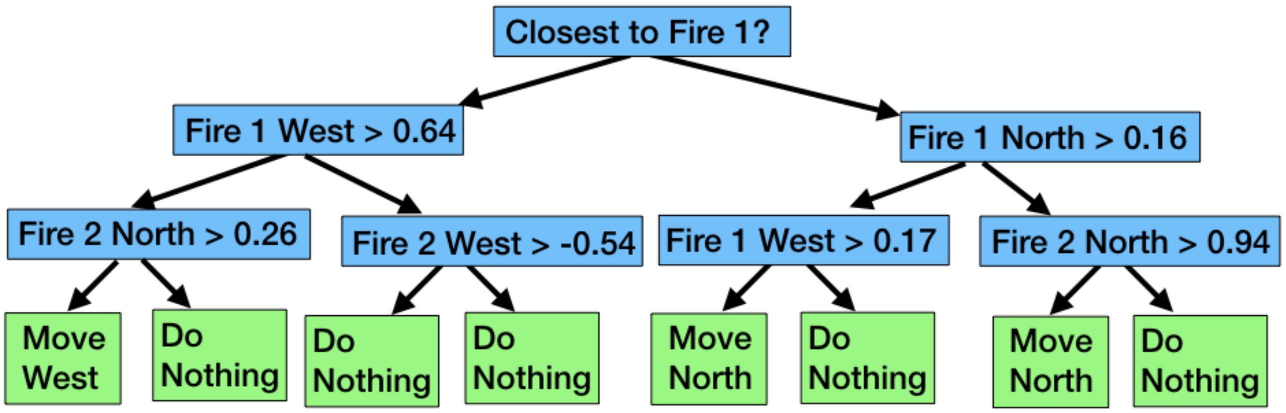


Figure 15: Full interpretable wildfire tracking policy. One node is redundant, leading to the same action regardless of how it is evaluated.

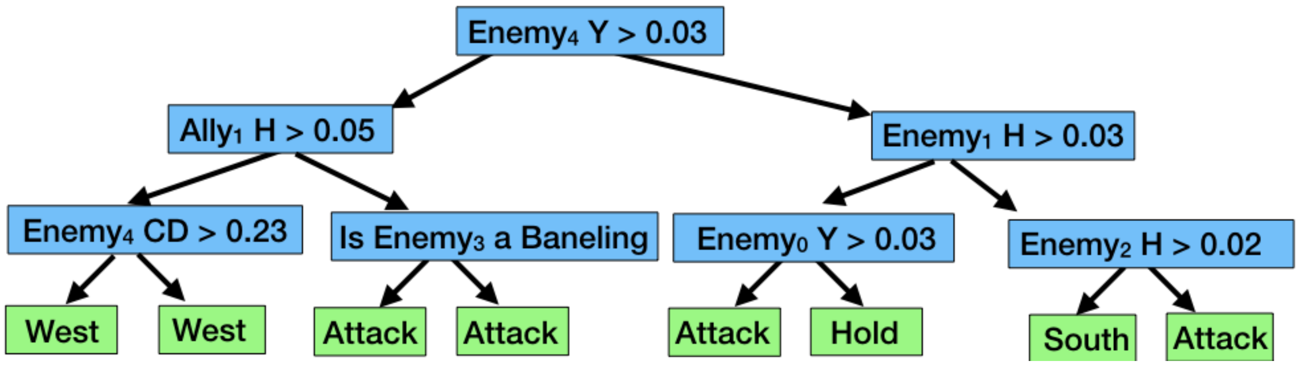


Figure 16: Full interpretable FindAndDefeatZerglings policy. One node is redundant, leading to the same action regardless of how it is evaluated.

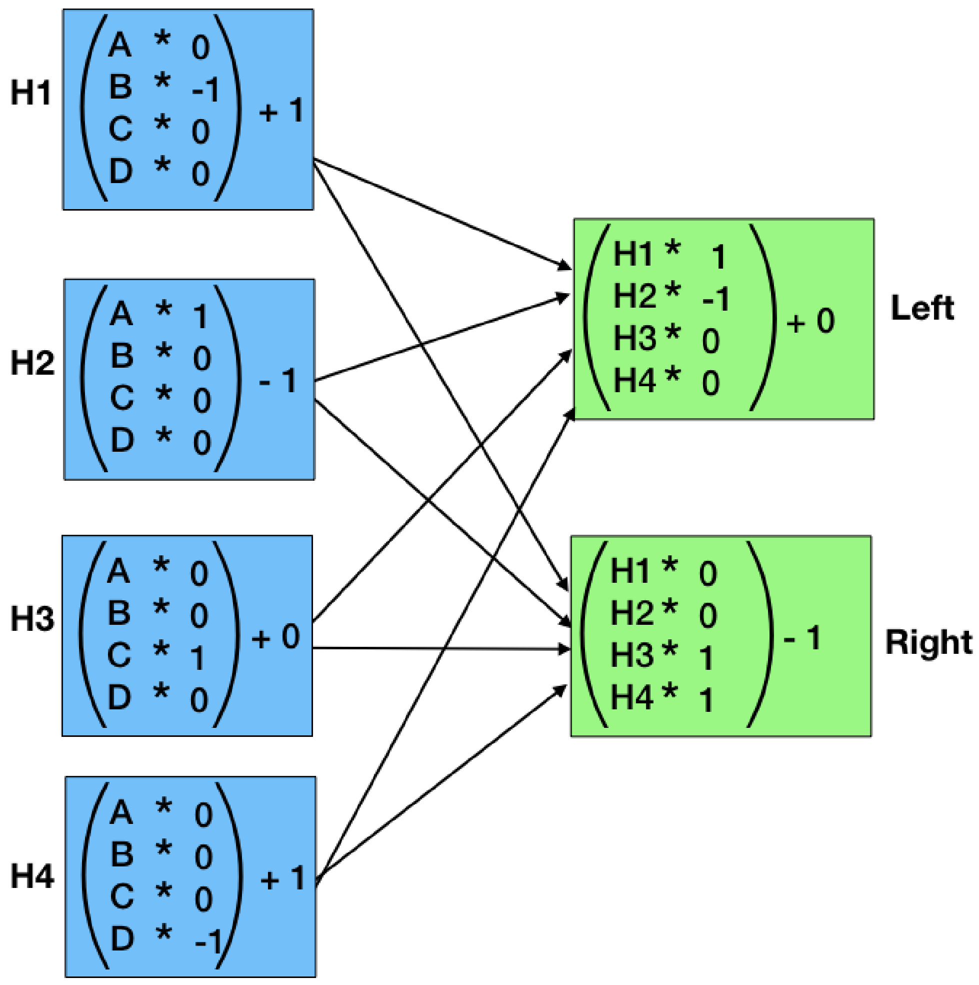


Figure 17: The MLP given to participants for our user study.

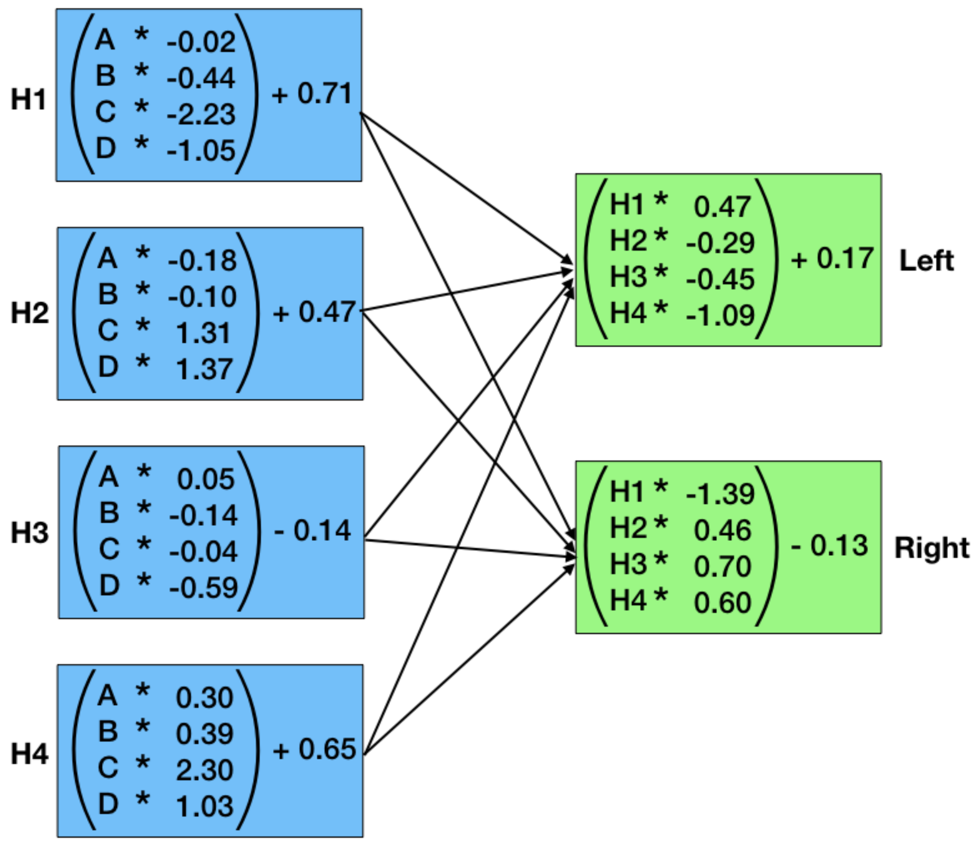


Figure 18: The actual MLP originally intended to go into the user study. Note that it is markedly more complicated than the version given to participants.