Sample-based Rule Extraction for Explainable Reinforcement Learning *

Raphael C. Engelhardt[1][0000−0003−1463−2706], Moritz Lange[2][0000−0001−7109−7813], Laurenz Wiskott[2][0000−0001−6237−740X], and Wolfgang Konen[1][0000−0002−1343−4209]

[1] Cologne Institute of Computer Science, TH Köln, Germany
{Raphael.Engelhardt,Wolfgang.Konen}@th-koeln.de
[2] Institute for Neural Computation, Faculty of Computer Science, Ruhr-University Bochum, Germany {Moritz.Lange,Laurenz.Wiskott}@ini.rub.de

Abstract. In this paper we propose a novel, phenomenological approach to explainable Reinforcement Learning (RL). While the ever-increasing performance of RL agents surpasses human capabilities on many problems, it falls short concerning explainability, which might be of minor importance when solving toy problems but is certainly a major obstacle for the application of RL in industrial and safety-critical processes. The literature contains different approaches to increase explainability of deep artificial networks. However, to our knowledge there is no simple, agent-agnostic method to extract human-readable rules from trained RL agents. Our approach is based on the idea of observing the agent and its environment during evaluation episodes and inducing a decision tree from the collected samples, obtaining an explainable mapping of the environment's state to the agent's corresponding action. We tested our idea on classical control problems provided by OpenAI Gym using handcrafted rules as a benchmark as well as trained deep RL agents with two different algorithms for decision tree induction. The extracted rules demonstrate how this new approach might be a valuable step towards the goal of explainable RL.

Keywords: Reinforcement learning · Explainable RL · Rule learning.

1 Introduction

One of the biggest downsides of powerful Deep Reinforcement Learning (DRL) algorithms is their opacity. The well-performing decision making process is buried in the depth of artificial neural networks, which might constitute a major barrier to the application of Reinforcement Learning (RL) in various areas. While methods exist especially in the field of computer vision to increase explainability of deep networks, a general, simple and agent-agnostic method of extracting human understandable rules from RL agents remains to be found. With this paper

* Supported by the German Ministry of Culture and Science of the State of North Rhine-Westphalia.

we propose a possible candidate for such a method and present results for classic control problems.

Instead of trying to specifically explain the inner mechanism of a trained RL agent, we approach the problem from a phenomenological perspective by constructing a set of simple, explainable rules, which imitate the behavior of the black-box RL agent. Our method consists of three steps:

1. A DRL agent is trained, and – for simpler problems – handcrafted (HC) policies are developed to solve the problem posed by the studied environment.
2. The agent (henceforth called "oracle") acting according to either HC rules or the trained policy is evaluated for a set number of episodes. At each timestep the state of the environment and action of the agent are logged.
3. A Decision Tree (DT) is induced from samples collected in the previous step.

The resulting tree is evaluated by applying its policy in a number of episodes and subsequently comparing average and standard deviation of the episodes' returns with the ones achieved by the oracle.

This approach has notable advantages:

- It is conceptually simple, as it translates the problem of explainable RL into a supervised learning setting.
- DTs are fully transparent and (at least for limited depth) offer a set of easily understandable rules.
- The approach is oracle-agnostic: it does not rely on the agent being trained by a specific RL algorithm, or any algorithm at all. A large enough training set of state-action pairs is sufficient.

This paper is structured in the following manner: In Section 2 we briefly discuss existing methods for improving explainability of RL. Section 3 explains in detail how our new approach works. Our experimental results are presented in Section 4.

2 Related Work

Over the years, several methods for rule deduction from other resources have been developed. Predicate invention [9,10], a subfield of inductive logic programming, is a well-known and established method for finding new predicates or rules from given examples and background knowledge in the symbolic domain. It is, however, difficult to apply to the non-symbolic domain (e.g. continuous observation and action spaces, continuous inputs and outputs of deep learning neural networks).

Rule extraction from opaque AI models has been widely studied in the context of explainable AI. There are many approaches, but none of the existing methods has proven conclusive for all applications. One method, called DeepRED [19], aims to translate feedforward networks layer-by-layer into rules and to simplify these rules. But the feedforward networks used in RL models do

not always directly output policies (actions). Instead they output values or Q-functions, which are transformed by the surrounding RL mechanism into policies. Therefore, surrogate networks or policy networks, which transform the inputs directly into policies would have to be trained based on existing RL oracles in order to apply DeepRED.

Another paradigm for generating explainable models is imitation learning [13,14] where a model with reduced complexity is learnt by guidance of a DRL oracle. The oracle serves as a form of expert demonstration for the desired behavior. Our approach is a form of imitation learning, as is the recent proposal by Verma et al. [17].

Verma et al. [17] propose a generative rule induction scheme, called Programmatically Interpretable RL (PIRL), through Neurally Directed Program Synthesis (NDPS). The key challenge is that the space of possible policies is vast and nonsmooth. This is addressed by using the DRL network to guide a local search in the space of programmatically synthesized policies. These policies are formed by a functional language that combines domain-specific operators and input elements. In the domain of classic control problems, these operators can for example mimic the well-known operators from PID control. As a result, this interesting method is able to generate complex yet interpretable rules. We will compare our results with those from [17]. Our approach differs from [17] insofar as it generates simpler rules (only one input per predicate) if we use traditional DTs like CART (of course at the price of a simpler policy space).

The method we describe in this paper is arguably closest to the mimic learning approach by Liu et al. [6]. Our approach differs to theirs in that it learns the policy directly rather than a Q-function. It does not require Q-values but merely recorded states and actions. Finally we use well-known simple decision trees instead of the more complex Linear Model U-trees.

Coppens et al. [4] also use trees for policy learning, but train hierarchical filters to obtain trees with stochastic elements. While we provide explainability through interpretable rules, Coppens et al. visualize the filters and perform statistical analysis of action distributions.

3 Methods

Our method makes use of different RL environments, DRL training algorithms and DT induction techniques, which we briefly describe in the following.

3.1 Environments

We test our approach on four classic control problems made available as RL environments by OpenAI Gym [3]:

MountainCar In this environment, first described by Moore [8, Chapter 4.3], a car initially positioned in a valley is supposed to reach a flag positioned on top of the mountain to the right as fast as possible. As the force of the car's motor

is insufficient to simply drive up the slope, it needs to build up momentum by swinging back and forth in the valley.

- Observation space: continuous, two-dimensional: car's position $x \in [-1.2, 0.6]$, car's velocity $v \in [-0.07, 0.07]$
- Action space: discrete, acceleration to the left, no acceleration, acceleration to the right, encoded as $0, 1, 2$
- Reward: -1 for each timestep as long as flag is not reached (flag is positioned at $x = 0.5$)
- Start state: x_0 random uniform in $[-0.6, -0.4]$, $v_0 = 0$
- Time limit: $t = 200$
- Solved if $\overline{R} \geq -110$ over 100 evaluation episodes

Fig. 1: MountainCar environment. Taken from [3]

MountainCarContinuous has the identical setup and goal as the discrete version described above. However, the action space is continuous and the reward structure is changed to favor less energy consumption. The differences to the discrete *MountainCar* are:

- Action space: continuous acceleration $a \in [-1, 1]$
- Reward: $-0.1 \cdot a^2$ for each timestep as long as the flag ($x = 0.45$) is not reached, additional $+100$ when the goal is reached
- Time limit: $t = 1000$
- Solved if $\overline{R} \geq 90$ over 100 evaluation episodes

MountainCarContinuous has the extra challenge that the cumulative reward of 0 (car stands still) is a local maximum.

CartPole described by Barto et al. [1, Chapter 5], consists of a pole balancing upright on top of a cart. By moving left or right, the cart should keep balancing the pole in an upright position for as long as possible, while maintaining the limits of the one-dimensional track the cart moves on. For better comparability with results of Verma et al. [17] mentioned in Section 2 we mostly use `CartPole-v0`.

- Observation space: continuous, four-dimensional: car's position $x \in [-4.8, 4.8]$, car's velocity $v \in (-\infty, \infty)$, pole's angle $\theta \in [-0.42, 0.42]$ (in radians) and pole's angular velocity $\omega \in (-\infty, \infty)$
- Action space: discrete, accelerate to the left (0) or to the right (1)
- Reward: $+1$ for each timestep as long as $\theta \in [-0.21, 0.21]$ and $x \in [-2.4, 2.4]$
- Start state: $x_0, v_0, \theta_0, \omega_0$ random uniform in $[-0.05, 0.05]$
- Time limit: t=200 (`CartPole-v0`) or t=500 (`CartPole-v1`)
- Solved if $\overline{R} \geq 195$ (`CartPole-v0`) or ≥ 475 (`CartPole-v1`) over 100 evaluation episodes

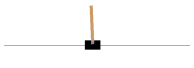


Fig. 2: CartPole environment. Taken from [3]

Pendulum The goal of this environment is to swing up an inverted pendulum, attached to an actuated hinge and keep it in this unstable equilibrium. The goal is the same as in CartPole, but Pendulum is more challenging, because the initial position can be at any angle and several swing-ups may be needed. (CartPole initializes at an near-upright position.) We use `Pendulum-v0`.

- Observation space: continuous, three-dimensional: $(\cos\theta, \sin\theta, \omega)$ with pendulum angle θ (from positive y-axis) and its angular velocity ω.
- Action space: continuous, torque $a \in [-2, 2]$ applied to pendulum
- Reward: $-(\theta^2 + 0.1 \cdot \omega^2 + 0.001 \cdot a^2)$ at each timestep
- Start state: $\theta_0 \in [-\pi, \pi]$, $\omega_0 \in [-1, 1]$ (random uniform)
- Time limit: $t = 200$
- No official solved-condition is given for this environment

Fig. 3: Pendulum environment. Taken from [3]

3.2 Oracles

The samples on which we train the DTs are produced by two "families" of oracles, which we explain in the following.

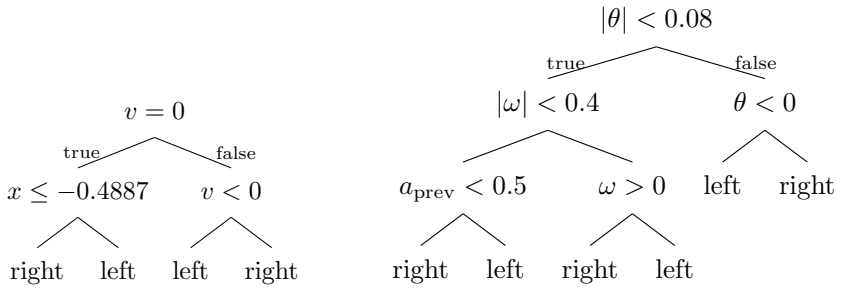
Deep Reinforcement Learning agents For the training of agents we use DRL algorithms as implemented in the RL Python framework Stable Baselines3 [12]. Specifically, we work with DQN [7], PPO [15] and TD3 [5].

We apply all three algorithms to all environments, but show later on in Section 4 only the most successful one out of three. It should be noted that the environments are not necessarily easy for DRL: For example, on MountainCar, DQN succeeds but PPO does not. Vice versa, on CartPole, PPO succeeds but DQN does not.

Handcrafted policies For three problems, we present simple deterministic policies (HC), which serve a triple purpose: they constitute a proof of existence for simple, well-performing rules, they may be used as a surrogate for an oracle and they serve as a benchmark for our core idea. They enable us to validate our approach. The fourth problem Pendulum currently has no HC policy. In the following, we briefly explain the HC rules for the different environments:

- MountainCar is solved by the rule set presented in Figure 4a. The intuition behind it is as follows: The velocity is zero if and only if we are in the starting step of an episode. If $v = 0$, we push the car left or right, depending on the car's position relative to -0.4887. This constant was found empirically, it minimizes the number of swing-ups needed. In all other cases with $v \neq 0$ we push the car further in the direction of its current velocity to build up momentum. – We will later see that this HC rule can be applied to MountainCarContinuous as well.

– CartPole: The following intuition motivates the rule set defined in Figure 4b. Once the pole is in the upright position (both absolute angle $|\theta|$ and absolute angular velocity $|\omega|$ are small) it merely needs to maintain its state. In these cases the CartPole environment imitates 'no action' (which is not available) by executing the opposite of the previous action. The previous action is made available as an additional observable. If the absolute angle $|\theta|$ is small but absolute angular velocity $|\omega|$ large, the action reduces $|\omega|$. In all other cases the cart is pushed in the direction in which the pole leans, which means that the pole comes closer to the upright position.



(a) MountainCar (b) CartPole

Fig. 4: HC policies for MountainCar, MountainCarContinuous, and CartPole

3.3 Decision Trees

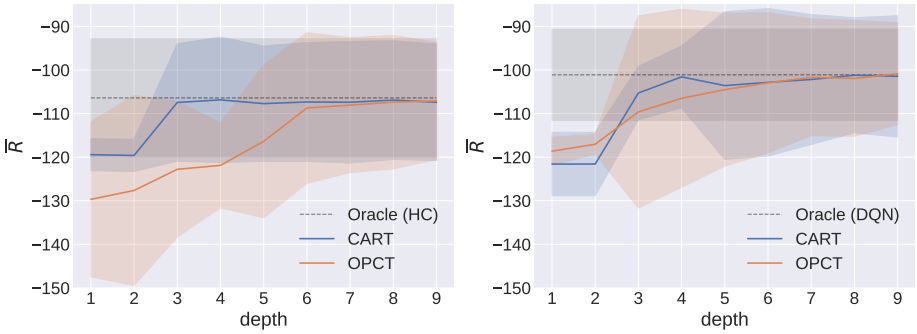
For the induction of DTs we rely on two different algorithms:

– Classification and Regression Trees (CART) as described by Breiman et al. [2] and implemented in [11]. We use the classification or the regression form depending on whether the environment's action space is discrete or continuous, respectively.
– Oblique Predictive Clustering Trees (OPCT) as described in [16].

We chose OPCT as one representative of the whole class of oblique DTs (trees with slanted lines, such as CART-LC, OC1 and HHCART). We tested also HHCART [18] and found it to give similar results to OPCT.

4 Results

We present in this section first the results on the four selected environments individually. Section 4.5 gives a summarizing table and discusses common findings.



(a) HC oracle (b) DQN oracle

Fig. 5: MountainCar. Performance comparison with different depths of trained DTs on samples produced by HC and DQN oracles.

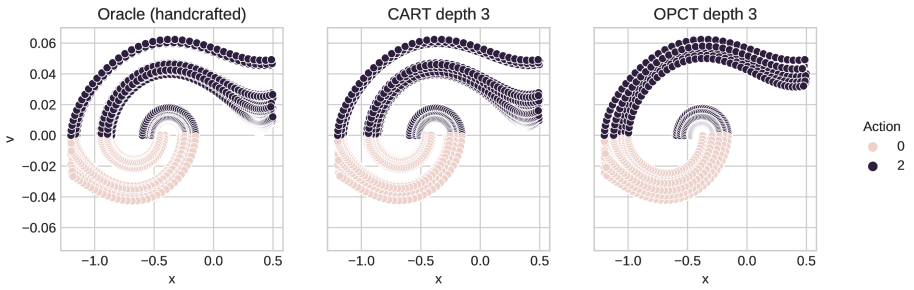
4.1 MountainCar-v0

Our results for the MountainCar environment are as follows. The HC rule reaches an average return of $\overline{R} = -106 \pm 13$.[3] As shown in Figure 5a, CART matches the oracle's performance at depth 3. OPCT only yields oracle-like performance at depths of at least 6, no longer considered to be easily interpretable. Trained with the DQN algorithm, the oracle yields a return of $\overline{R} = -101 \pm 10$. Again, CART has similar returns for depth ≥ 3 (Figure 5b).

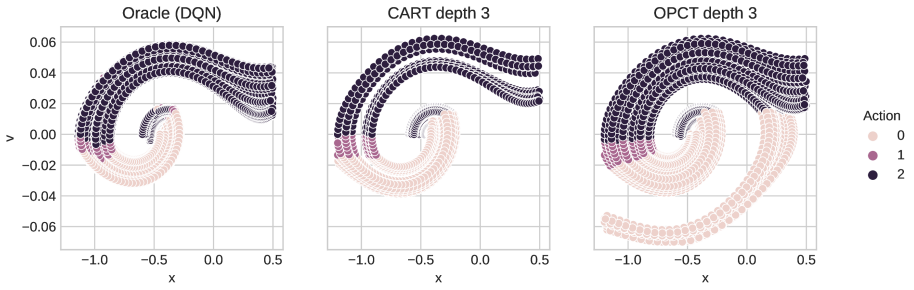
Given the two-dimensional state space of this environment, the characteristics of oracle and DT can be compared visually (see Figure 6). Since the HC rule is formulated as a DT, it can be easily compared with the DT induced by CART from samples. Even if the trees in Figure 4a and 7a are structured differently, the underlying rules are extremely similar. The latter shows how CART circumvents its inability to represent a predicate such as $v = 0$.

If we induce DTs from samples of a deep learning oracle DQN, it is not a priori clear that similar rules emerge. As an example Figure 7b shows the 'CART from DQN' rules at depth 3. It is interesting to note that the left subtree mainly contains leaves with action 'left'. If we replaced the 'no acceleration' leaf with 'left', a much simpler tree, structurally similar to Figure 7a, would result with virtually same return $\overline{R} = -105 \pm 6$. The tree has just inserted the 'no acceleration' leaf to mimic the DQN samples, but it turns out not to be important for the overall return. It is noteworthy that the simplified tree is again structurally similar to both trees in Figures. 7a & 4a related to the HC policy.

[3] We report the average mean return ± average standard deviation of returns for the 5 repetitions with different seeds.



(a) HC oracle



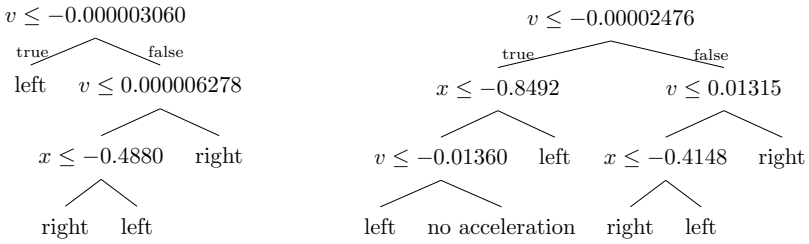
(b) DQN oracle

Fig. 6: MountainCar. Comparison between oracle and DT samples. Each point in the coordinate system represents the state of the environment at a timestep. The corresponding decision of the oracle or the induced DT is represented by the marker's color.

4.2 MountainCarContinous-v0

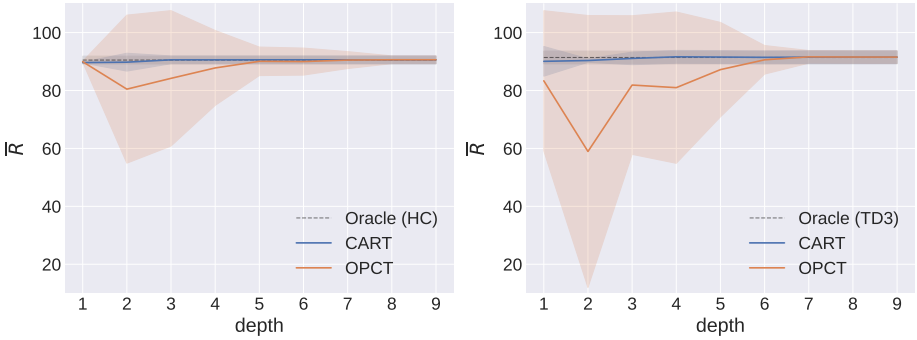
The continuous version of MountainCar is successfully solved by a TD3-trained DRL agent reaching a score of $\overline{R} = 91 \pm 2$. CART at depth 3 is able to reproduce this performance (see Figure 8b). The performance of OPCT is somewhat weaker, but also almost satisfactory. The behaviours of oracle and trees can be compared in Figure 9.

Given the structural similarity between the discrete and continuous version of MountainCar, it seems reasonable to apply the same HC rule, that solved the discrete version. With a performance of $\overline{R} = 91 \pm 1$ in fact, the HC rule is able to solve the continuous version as well. Our approach proves successful in that CART is able to reproduce the performance and to extract the known HC rule (neglecting minor numerical differences, the extracted DT is the same as the one shown in Figure 7a). OPCT could not reach the oracle's performance at depth 3 (see Figure 8a). Even at depth 1 both DTs show fairly good performance: CART finds a simplified version of the HC rule by splitting the state space at $v \approx 0$ and pushing the car in the direction of v.



(a) Induced from HC samples (b) Induced from DQN samples

Fig. 7: MountainCar. DTs induced by CART from samples of different oracles



(a) HC oracle (b) TD3 oracle

Fig. 8: MountainCarContinuous. Performance comparison with different depths of trained DTs on samples produced by HC and TD3 oracles.

4.3 CartPole-v0

The CartPole environment is in a sense more challenging than the previous ones since it has four input dimensions. Interpretable models are less easy to visualize. On CartPole, we test our approach with a HC oracle and with a PPO oracle.

Our HC oracle, taking into consideration the action of the previous timestep, reaches a perfect score of $\overline{R} = 200 \pm 0$ (Figure 12). At depth 3, both CART and OPCT solve or nearly solve this environment with returns of $\overline{R} = 200 \pm 0$ and $\overline{R} = 192 \pm 15$, respectively, when trained on samples generated with said HC agent. The DRL oracle based on PPO performs as well as the HC oracle. In this case, the inputs are just the original CartPole variables x, v, θ, ω (no previous action a_{prev}). Both DT algorithms replicate the perfect score from depth 3 on.

Example rule sets found by depth-3 CART from HC and PPO agents are shown in Figure 10. The tree in Figure 10a is structurally different from the HC rule it is derived from (Figure 4b). But upon careful inspection it becomes clear how it actually represents the same decision process, which explains the equal performance.

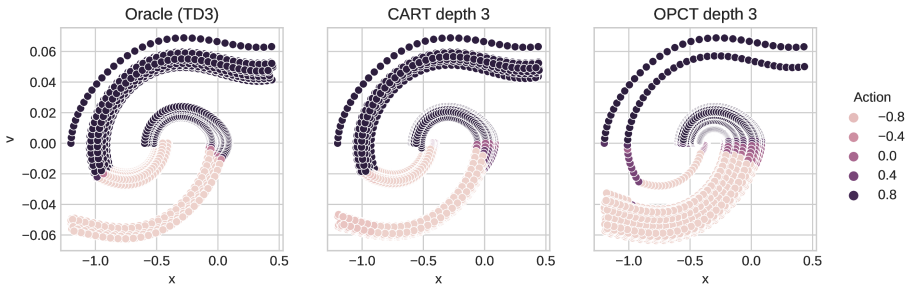
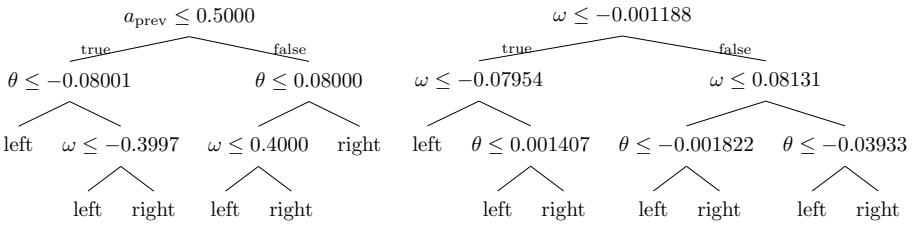


Fig. 9: MountainCarContinous. Comparison between TD3, CART, and OPCT.



(a) CART based on HC oracle (b) CART based on PPO agent

Fig. 10: CartPole. DTs induced by CART algorithm from samples of HC and PPO oracles. Note how, given the encoding of action 'left' and 'right' as 0 and 1, respectively, the condition of the root node in (a) effectively checks whether or not the previous action was 'left'.

Figure 10b is interesting in its own right: By careful inspection we see that the middle branches can be combined into the rule 'If $|\omega| < 0.08$ then apply action *left* or *right* depending on $\theta < 0$ or ≥ 0, respectively' In other words, for small angular velocities push just in the direction where the pole is leaning. The outer branches mean: For larger $|\omega|$ push into the direction that reduces $|\omega|$.[4] We can summarize these findings as a new rule set (Figure 11), which is even simpler than the one in Figure 4b.

This new HC rule derived from *'CART based on PPO agent'* has perfect performance $\overline{R} = 200$ when applied to the environment `CartPole-v0` and it has a higher performance $\overline{R} = 499$ on `CartPole-v1`, where the algorithm in Figure 4b reaches only $\overline{R} = 488$.

As shown in Figure 12b, OPCT trained on samples from PPO oracle reaches perfect performance at a depth of 1. This means it solves the CartPole problem with *one* single rule, shown in Figure 13. While oblique DTs are generally more difficult to interpret, sign and magnitude of the coefficients might provide some

[4] The condition $\omega > 0.08 \wedge \theta < -0.04$ is neglected here, since it can be assumed to appear very seldom: Positive ω will normally lead to positive θ.

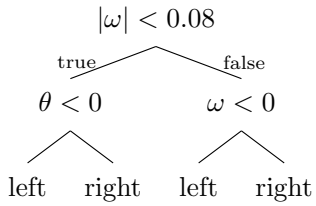
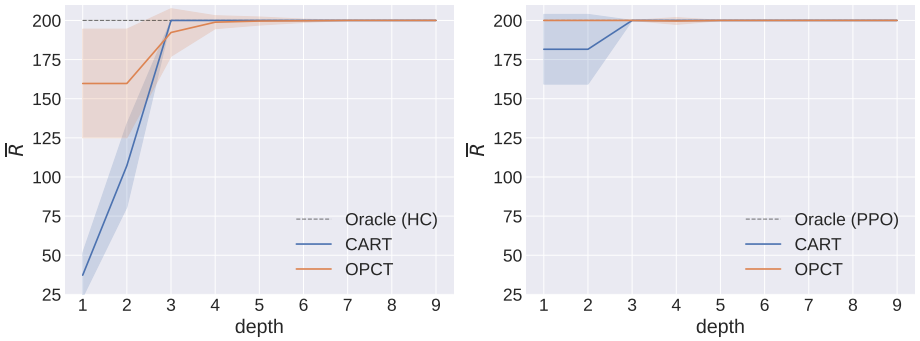


Fig. 11: CartPole. Simplified rule set derived from the *'CART based on PPO agent'* tree shown in Figure 10b.



(a) HC oracle

(b) PPO oracle

Fig. 12: CartPole. Performance comparison with different depths of trained DTs on samples produced by HC and PPO.

helpful insights: In Figure 13, θ has by far the largest coefficient, indicating that the model is most sensitive to this variable. The sign of θ will decide in most cases whether action *left* or *right* is chosen.

Interestingly, also the CART-induced tree performs quite well at depth 1 and 2 (better than its counterpart trained on HC-samples at this depth). The respective tree decides solely on the basis of the sign of ω. The good performance shows that the PPO-samples are easier to segment by linear cuts than the HC-samples. This may be due to the fact that PPO avoids the non-linear function $|\cdot|$ (needs two cuts in a DT) and the extra input a_{prev} (needs another cut).

4.4 Pendulum-v0

Now we turn to the environment where our method so far is *not* able to produce good DTs at low depth: On the Pendulum environment a DRL oracle using TD3 reaches a good performance of $\overline{R} = -154 \pm 85$. The pole is stabilized in its unstable upright position in nearly every episode. Both tested DTs fail to reach TD3 oracle's performance at reasonable depths (see Figure 14). While the

$$2.788x - 1.843v + 41.16\theta + 1.870\omega < -0.0080$$

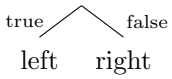


Fig. 13: Single rule solving the CartPole problem. This rule was found by OPCT based on PPO samples.

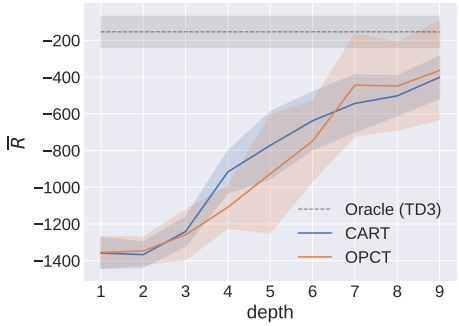


Fig. 14: Pendulum. Performance comparison with different depths of trained DTs on samples produced by HC and TD3 oracles.

performance clearly increases with the depth of the DTs, such trees no longer satisfy the criterion of explainability.

The original `Pendulum-v0` environment represents the angle as $\cos(\theta)$ and $\sin(\theta)$. We kept this representation throughout the experiments. For better visualization and without loss of information, we map $(\cos(\theta), \sin(\theta))$ back to the angle θ itself. In Figure 15, the difference between the performances of oracle and DTs is reflected in the samples they produce: The TD3 oracle has learned to stabilize the pole in an upright position ($\theta \approx 0$) as can be seen from the white areas (absence of samples) for ($\theta \approx 0, |\omega| > 0$) in the left part of Figure 15. The DTs fail to do so: there are many samples with $|\omega| > 0$ in region $\theta \approx 0$, meaning that the pole will rotate fast through the upright position.

It is an open question *why* our method is not able to produce good DTs at low depth for this environment. We speculate that the complex functional relationship between input space and continuous action space might be responsible for this. If we compare the TD3 picture in the left part of Figure 15 with the other continuous-action environment MountainCarContinuous in the left part of Figure 9, we see in the former intricate overlaps of many different action levels while in the latter mainly two action levels appear that can be separated by a few cuts. This argument is supported by the fact that DTs with higher depths (more complex functions) come closer to the oracle's performance in Figure 14.

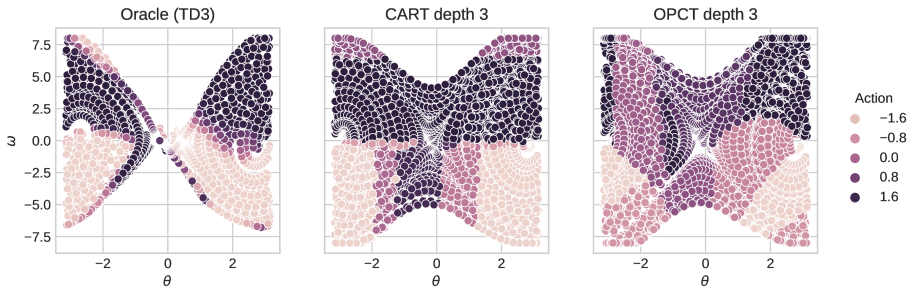


Fig. 15: Pendulum. Comparison between TD3 oracle, CART, and OPCT samples.

This does not mean that there *is* no simple explainable model: It might be that there exists another representation of the input space (yet to be found!) where the desirable action levels fall into simpler regions.

4.5 Comparison of Approaches

We summarize all obtained results in Table 1. All oracles meet the *solved* condition of the environments and all DTs of only depth 3 meet it as well or come at least close (with the exception of the Pendulum environment).

With two slightly different optimization schemes, Verma et al. [17, App. A, Tab. 6] report two numbers, which we report in row *[Verma]* of Table 1. The rules extracted by means of our method exhibit better performance than the rules obtained by the PIRL approach of Verma et al. [17]. Furthermore, we could not reproduce these results: When we apply the rules explicitly stated in [17, App. B, Figs. 9 & 10] to the respective environments, we obtain only the lower results shown in rows *[Verma_R]* of Table 1. Finally, the rules given in [17] are not easy to interpret.

Given their capacity to find optima in a broader class of splits, OPCT models were expected to map DRL oracles' behaviours better (at lower depth) than CART, which is limited to axis-parallel cuts. These expectations are *not* fulfilled by our results. OPCT often only had a similar or lower performance than CART at the same depth level. In the rare cases where OPCT is better than CART (e.g. CartPole at depth 1, Figure 12), the single OPCT rule obtained (Figure 13) is compact, but still difficult to interpret.

5 Conclusion

In this paper we provide a new method of extracting plain, human-readable rules from DRL agents. The method is simple, provides transparent rules and is not specific to any particular kind of oracle. We show that for the problems MountainCar, MountainCarContinuous and CartPole our approach is able to

Table 1: Results for combinations of environment, oracle and DT algorithm. The DTs are of depth 3 in all cases. Reported are the average mean return $\overline{R}$ $\pm$ average standard deviation of returns $\overline{\sigma}$ for 5 repetitions with different seeds, each evaluated for 100 episodes. Agents with $\overline{R} \geq R_{solved}$ are said to *solve* an environment. See main text for the meaning of rows *[Verma]* and *[Verma_R]*.

Environment (R_{solved})	**Oracle**	**Oracle's $\overline{R} \pm \overline{\sigma}$**	**DT**	**DT's $\overline{R} \pm \overline{\sigma}$**
MountainCar-v0 (−110)	HC	-106.42 ± 13.57	CART	-107.46 ± 13.47
			OPCT	-122.80 ± 15.55
	DQN	-101.13 ± 10.45	CART	-105.29 ± 6.11
			OPCT	-109.63 ± 22.06
	[Verma]	-143.9, -108.1		
	[Verma_R]	-162.6 ± 3.8		
MountainCarCont.-v0 (90)	TD3	91.40 ± 2.19	CART	91.08 ± 2.16
			OPCT	81.89 ± 23.93
	HC	90.51 ± 1.33	CART	90.60 ± 1.33
			OPCT	84.20 ± 23.38
CartPole-v0 (195)	HC	200.00 ± 0.00	CART	200.00 ± 0.00
			OPCT	192.30 ± 15.20
	PPO	200.00 ± 0.00	CART	200.00 ± 0.00
			OPCT	200.00 ± 0.00
	[Verma]	143.2, 183.2		
	[Verma_R]	106.0 ± 16.9		
Pendulum-v0	TD3	-154.17 ± 85.02	CART	-1241.24 ± 80.38
			OPCT	-1258.98 ± 136.37

extract simple rule sets, which perform as well as the oracles they are based on. Our method currently has its limitations in the Pendulum environment, where it is so far not possible to automatically extract simple and well-performing rules.

We show in several cases that the automatically induced trees are 'understandable' in the sense that they are similar or equivalent to handcrafted rule sets that we provided as benchmarks.

Another aspect is that given the transparency of the extracted rule sets, these can later be inspected and possibly optimized or simplified (MountainCar, discussion of Figure 7b). In another case (CartPole, CART induced from DRL oracle PPO, Figure 10b), we could simplify the tree and obtain an even simpler and better-performing handcrafted rule than the previously known one.

Future work will consist in determining the reason for the failure of the method when applied to the Pendulum problem. The method should then be improved and applied to more complex environments. We also want to study in all these environments the unsupervised learning of representations, which are then fed into DTs as additional features.

We hope that this work and our future research will help to establish better explainable models in the field of deep reinforcement learning.

References

1. Barto, A.G., Sutton, R.S., Anderson, C.W.: Neuronlike adaptive elements that can solve difficult learning control problems. IEEE Trans. Syst., Man, Cybern. **13**(5), 834–846 (1983). https://doi.org/10.1109/TSMC.1983.6313077
2. Breiman, L., Friedman, J.H., Olshen, R.A., Stone, C.J.: Classification And Regression Trees. Routledge (1984)
3. Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba, W.: OpenAI Gym (2016), http://arxiv.org/abs/1606.01540
4. Coppens, Y., Efthymiadis, K., et al.: Distilling deep reinforcement learning policies in soft decision trees. In: Proc. IJCAI 2019 workshop on explainable artificial intelligence. pp. 1–6 (2019)
5. Fujimoto, S., van Hoof, H., Meger, D.: Addressing function approximation error in actor-critic methods. In: Dy, J., Krause, A. (eds.) Proc. 35th ICML. pp. 1587–1596. PMLR (2018)
6. Liu, G., et al.: Toward interpretable deep reinforcement learning with linear model U-trees. In: Berlingerio, M., et al. (eds.) Machine Learning and Knowledge Discovery in Databases. LNCS, vol. 11052, pp. 414–429. Springer (2019). https://doi.org/10.1007/978-3-030-10928-8_25
7. Mnih, V., Kavukcuoglu, K., et al.: Playing Atari with deep reinforcement learning (2013), http://arxiv.org/abs/1312.5602
8. Moore, A.W.: Efficient memory-based learning for robot control. Tech. rep., University of Cambridge (1990)
9. Muggleton, S.: Predicate invention and utilization. J. Exp. Theor. Artif. Intell. **6**(1), 121–130 (1994). https://doi.org/10.1080/09528139408953784
10. Muggleton, S.H., Lin, D., Tamaddoni-Nezhad, A.: Meta-interpretive learning of higher-order dyadic datalog: Predicate invention revisited. Machine Learning **100**(1), 49–73 (2015). https://doi.org/10.1007/s10994-014-5471-y
11. Pedregosa, F., Varoquaux, G., et al.: Scikit-learn: Machine learning in Python. Journal of Machine Learning Research **12**(85), 2825–2830 (2011)
12. Raffin, A., Hill, A., et al.: Stable-baselines3: Reliable reinforcement learning implementations. Journal of Machine Learning Research **22**(268), 1–8 (2021)
13. Ross, S., Gordon, G., Bagnell, D.: A reduction of imitation learning and structured prediction to no-regret online learning. In: Proc. 14th Int. Conf. Artif. Intell. Stat. vol. 15, pp. 627–635 (2011)
14. Schaal, S.: Is imitation learning the route to humanoid robots? Trends in Cognitive Sciences **3**(6), 233–242 (1999). https://doi.org/10.1016/S1364-6613(99)01327-3
15. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms (2017), http://arxiv.org/abs/1707.06347
16. Stepišnik, T., Kocev, D.: Oblique predictive clustering trees. Knowledge-Based Systems **227**, 107228 (2021). https://doi.org/10.1016/j.knosys.2021.107228
17. Verma, A., Murali, V., et al.: Programmatically interpretable reinforcement learning. In: Dy, J., Krause, A. (eds.) Proc. 35th ICML. pp. 5045–5054. PMLR (2018)
18. Wickramarachchi, D., Robertson, B., Reale, M., Price, C., Brown, J.: HHCART: An oblique decision tree. Computational Statistics & Data Analysis **96**, 12–23 (2016). https://doi.org/10.1016/j.csda.2015.11.006
19. Zilke, J.R., et al.: DeepRED – rule extraction from deep neural networks. In: Calders, T., et al. (eds.) Int. Conf. on Discovery Science. pp. 457–473. Springer (2016). https://doi.org/10.1007/978-3-319-46307-0_29