Finding the Relevant Samples for Decision Trees in Reinforcement Learning

Raphael C. Engelhardt [1] Moritz Lange [2] Laurenz Wiskott [2] Wolfgang Konen [1]

[1]Cologne Institute of Computer Science, TH Köln [2]Institute for Neural Computation, Ruhr-University Bochum

Gefördert von Ministerium für Kultur und Wissenschaft des Landes Nordrhein-Westfalen

Motivation

The use of neural networks to solve challenges posed by various reinforcement learning environments comes at the cost of complex, opaque models. In our previous work [1], we investigated the possibility of inducing simple decision trees (DT) of low depth matching the deep reinforcement learning (DRL) performance while requiring orders of magnitude fewer parameters. Our previous approach based on collecting samples during DRL agent's evaluation episodes and subsequently inducing DTs has proven successful on some environments but faced difficulties on others. We extended our investigations to overcome encountered challenges and compare three different techniques of generating training data for DTs.

Approach

The three investigated methods of producing a training set for DT induction are:

1. *Episodes*: The purely episode-based approach consists of logging the observables and the corresponding actions during evaluation episodes of the DRL agent ("oracle")
2. *Bounding Box*: Based on the statistics of visited points in the observation space during successful oracle episodes, a given number of samples are taken from the uniform distribution within a hyper-rectangle of side lengths $[L_i - 1.5 \cdot \mathrm{IQR}_i, U_i + 1.5 \cdot \mathrm{IQR}_i]$ with L_i and U_i the lower and upper bound of the visited points in the i^{th} dimension of the observation space and IQR_i their interquartile range.
3. *Iterative*: An initial DT $T^{(0)}$ (based on oracle episode samples) is evaluated. The visited points in the observations space $\mathcal{O}_T$ are labeled with the oracle's decisions $\mathcal{A}_O$ and merged with the current training set. Afterwards a new DT $T^{(1)}$ is induced from the enriched training set and the next iteration begins. A schematic representation is given in Figure 1.

Methods

We evaluate all three methods on all OpenAI Gym's [2] classic control problems, LunarLander, and CartPole-SwingUp as implemented in [3]. For the training of oracles we rely on DQN [4], PPO [5], and TD3 [6] in their implementation of stable-baselines3 [7]. For the induction of DTs we use Classification and Regression Trees (CART) as described by [8] and implemented in [9] which makes axis-parallel splits in the feature space, and the Oblique Predictive Clustering Trees (OPCT) as described and implemented in [10]. Since the DTs exhibit rather strong fluctuations depending on random initialization, we always generate 10 DTs with different seeds and pick the best-performing one.

All algorithms for sample generation are tested on all environments with DTs of depth $d = 1, \ldots, 10$. For every experiment the average return $\overline{R}$ in 100 evaluation episodes is computed. Each experiment is repeated 10 times, and μ and σ are reported.

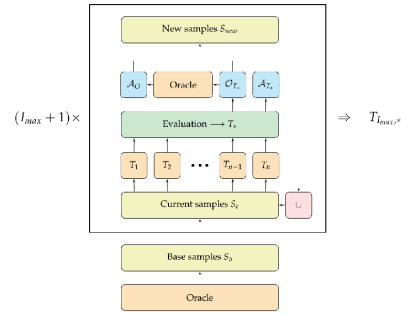

Figure 1. Flowchart of *Iterative*

Results

Figure 2 shows OPCT performance of different sampling methods as a function of DT depth d. The performance of the oracle and the threshold at which an environment is considered "solved" are shown as horizontal lines. DTs induced from samples obtained by the *iterative* method show best performance and solve the environments at moderate depths $d \leq 7$. Often DTs even outperform the oracles, which is remarkable since they require orders of magnitude fewer parameters (see Table 1). Using the same total number of samples, the *iterative* method requires about ten times more computation time than the other two methods.

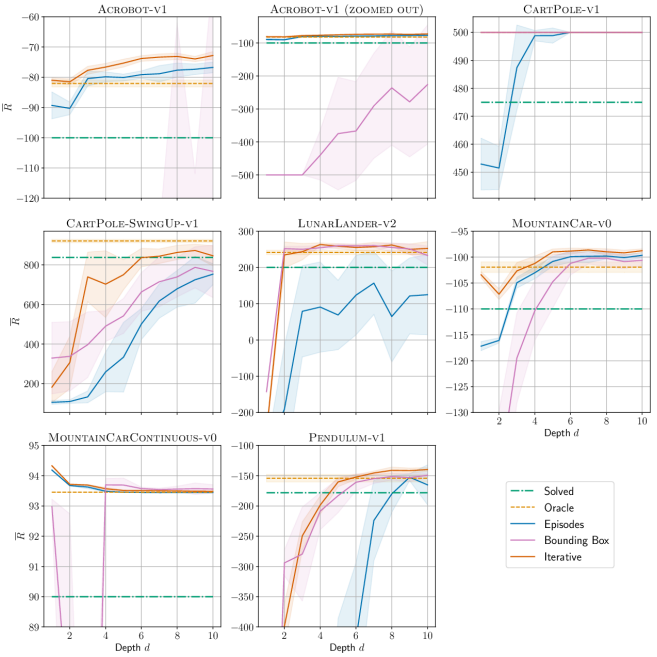

Figure 2. Return $\overline{R}$ as a function of DT depth d for all environments and all presented algorithms. The solved-threshold is shown as dash-dotted green line, the average oracle performance as dashed orange line, and the DTs performances as solid lines with average and $\pm 1\sigma$ of ten repetitions as shaded area

Decision Trees as Lightweight Models

The number of parameters in DRL models strongly depends on the network architecture (we used default parameters for our experiments) but is generally in the order of $10^4 - 10^6$. DTs on the other hand require far fewer parameters. The exact number depends on the dimension of the input space D and heavily on depth d

Table 1. Oracle and tree complexity. (D denotes the dimension of the observation space)

			Number of parameters		
Environment	dim D	Model	Oracle	OPCT	depth d
`Acrobot-v1`	6	DQN	136,710	9	1
`CartPole-v1`	4	PPO	9 155	7	1
`CartPole-SwingUp-v1`	5	DQN	534,534	890	7
`LunarLander-v2`	8	PPO	9 797	31	2
`MountainCar-v0`	2	DQN	134,656	5	1
`MountainCarContinuous-v0`	2	TD3	732,406	5	1
`Pendulum-v1`	3	TD3	734,806	156	5

Summary

- All considered environments can be solved by DTs at moderate depths
- The *iterative* approach is most successful generating a suitable training set
- The *iterative* approach requires about ten times more computation time compared to the other two
- DTs often even outperform DRL agents
- DTs require orders of magnitude fewer parameters than DRL counterparts
- Our method still requires DRL agents for building the DT surrogate model

Acknowledgment

This research was supported by the research training group 'Dataninja" (Trustworthy AI for Seamless Problem Solving: Next Generation Intelligence Joins Robust Data Analysis) funded by the German federal state of North Rhine-Westphalia.

References

[1] Raphael C. Engelhardt, Moritz Lange, Laurenz Wiskott, and Wolfgang Konen. Sample-Based Rule Extraction for Explainable Reinforcement Learning. In Giuseppe Nicosia, Varun Ojha, Emanuele La Malfa, et al., editors, *Machine Learning, Optimization, and Data Science*, pages 330–345. Springer Nature, 2023.

[2] Greg Brockman, Vicki Cheung, Ludwig Pettersson, et al. OpenAI Gym. 2016. `https://doi.org/10.48550/arXiv.1606.01540`.

[3] Ângelo Gregório Lovatto. CartPole Swingup - A simple, continuous-control environment for OpenAI Gym, 2021. `https://github.com/0xangelo/gym-cartpole-swingup`.

[4] Volodymyr Mnih, Koray Kavukcuoglu, et al. Playing Atari with deep reinforcement learning, 2013. `https://doi.org/10.48550/arXiv.1312.5602`.

[5] John Schulman, Filip Wolski, Prafulla Dhariwal, et al. Proximal policy optimization algorithms, 2017. `https://doi.org/10.48550/arXiv.1707.06347`.

[6] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In Jennifer Dy and Andreas Krause, editors, *Proc. 35th ICML*, pages 1587–1596. PMLR, 2018.

[7] Antonin Raffin, Ashley Hill, et al. Stable-baselines3: Reliable reinforcement learning implementations. *JMLR*, 22(268):1–8, 2021.

[8] Leo Breiman, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone. *Classification And Regression Trees*. Routledge, 1984.

[9] F. Pedregosa, G. Varoquaux, et al. Scikit-learn: Machine learning in Python. *JMLR*, 12:2825–2830, 2011.

[10] Tomaž Stepišnik and Dragi Kocev. Oblique predictive clustering trees. *Knowledge-Based Systems*, 227:107228, 2021.

[11] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym, 2016.

[12] Antonin Raffin, Ashley Hill, Maximilian Ernestus, et al. Stable Baselines3, 2019. `https://github.com/DLR-RM/stable-baselines3`.