Upside-Down Reinforcement Learning for More Interpretable Optimal Control

Juan Cardenas-Cartagena, Massimiliano Falzari, Marco Zullich, and Matthia Sabatelli

University of Groningen

Abstract. Model-Free Reinforcement Learning (RL) algorithms either learn how to map states to expected rewards, or search for policies that can maximize a certain performance function. Model-Based algorithms instead, aim to learn an approximation of the underlying model of the RL environment and then use it in combination with planning algorithms. Upside-Down Reinforcement Learning (UDRL) is a novel learning paradigm that aims to learn how to predict actions from states and desired commands. This task is formulated as a Supervised Learning (SL) problem and has successfully been tackled by Neural Networks (NNs). In this paper, we investigate whether function approximation algorithms other than NNs can also be used within a UDRL framework. Our experiments, performed over several popular optimal control benchmarks, show that tree-based methods like Random Forests and Extremely Randomized Trees perform just as well as NNs with the significant benefit of resulting in policies which are inherently more interpretable than NNs, therefore paving the way for more transparent, safe, and robust RL.

Keywords: Upside-Down Reinforcement Learning · Neural Networks · Random Forests · Interpretability · Explainable AI

1 Upside-Down Reinforcement Learning

The idea of Upside-Down Reinforcement Learning (UDRL) [4] is that of tackling Reinforcement Learning (RL) problems via Supervised Learning (SL) techniques. To understand how this can be achieved, we define a Markov Decision Process as a tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$ where $\mathcal{S}$ corresponds to the state space of the environment, $\mathcal{A}$ is the action space modeling all the possible actions available to the agent, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is the transition function and $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$ is the reward function. When interacting with the environment, at each time-step t, the RL agent performs action a_t in state s_t, and transitions to state s_{t+1} alongside observing reward signal r_t. The goal is then to learn a policy $\pi : \mathcal{S} \rightarrow \Delta(\mathcal{A})$ which maximizes the expected discounted sum of rewards $\mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k r_{t+k}]$, with $\gamma \in [0, 1)$ and $\Delta(\mathcal{A})$ being the set of all probability distributions over $\mathcal{A}$. In RL, the agent can either learn to predict expected rewards, assuming the model-free RL set-up, or learn to predict s_{t+1} and r_t, assuming a model-based RL setting. However, a UDRL agent doesn't do either, and its main goal is learning to

predict actions instead. Given an RL transition $\tau = \langle s_t, a_t, r_t, s_{t+1} \rangle$, an UDRL agent uses the information derived from τ to learn to map a certain state s_t, alongside commands d_r and d_t, to action a_t. These commands correspond to the desired reward d_r the agent wants to achieve within a certain time horizon d_t when being in state s_t. The ultimate goal of a UDRL agent is, therefore, that of learning $f : (s_t, d_r, d_t) \rightarrow a_t$, a task that can be formalized as an SL problem. This function f comes under the name Behaviour Function and is typically modeled by a neural network (NN); however, this is not strictly required. In this paper, we implement it as tree-based methods, specifically as Random Forests (RFs) [1] and Extremely Randomized Trees (ETs) [2].

2 Results

In Figure 2, we show the training curves for three different Behaviour Functions f: NNs, RFs, and ETs on the popular optimal control benchmarks `CartPole`, `Acrobot`, and `Lunar-Lander`. Our results show, on average, on-par performance across tested algorithms. The tree-based methods, however, allow us to get global explanations for the Behaviour Function, which in the case of NNs is not straightforward. In Figure 1, we show a snapshot of the feature importance scores estimated as mean impurity decrease. The scores are computed at inference time based on the work presented by Louppe et al. [3]. We compute these for the different features modeling a specific state s_t of the `CartPole` environment once the Behaviour Function has successfully learned how to solve the task. We can see that the Behavior Function considers the pole angle of the cart denoted by θ to be the most important feature, allowing the cart to stay in balance. We hope that these results pave the way for studies around interpretable and explainable optimal control.

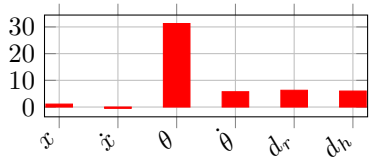

Fig. 1: Feature importance scores for a state of the `CartPole` environment coming from a RF Behaviour Function.

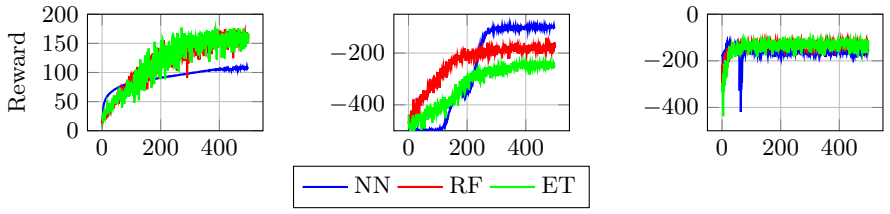

Fig. 2: Training curves obtained on the `CartPole`, `Acrobot`, and `LunarLander` by the three tested Behaviour Functions f.

References

1. Leo Breiman. Random forests. *Machine learning*, 45:5–32, 2001.
2. Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. *Machine learning*, 63:3–42, 2006.
3. Gilles Louppe, Louis Wehenkel, Antonio Sutera, and Pierre Geurts. Understanding variable importances in forests of randomized trees. *Advances in neural information processing systems*, 26, 2013.
4. Juergen Schmidhuber. Reinforcement learning upside down: Don't predict rewards–just map them to actions. *arXiv preprint arXiv:1912.02875*, 2019.