Symbolic Model-Based Reinforcement Learning

Simone Manti, AU ID 734894, smanti@mpe.au.dk
Final project for Neural Networks and Deep Learning 2024

October 15, 2024

1 Introduction

This project, highly inspired by [1], aims to investigate the performance of Symbolic Regression (SR) in Model-Based Reinforcement Learning (MBRL). Indeed, the most common MBRL algorithm choices consist of black-box methods, known to be data-hungry and prone to overfitting when a small amount of data is available. In this project, it has been shown empirically that SR outperforms standard MBRL on simple environments, providing to the user interpretable and simple models. This study is subdivided as follows:

- section 2 introduces the main components of MBRL;
- section 3 deals with a short description of SR;
- section 4 is dedicated to experiments and implementation details;
- finally, section 5 concludes the work with a short discussion.

2 Model-Based Reinforcement Learning

Reinforcement Learning (RL), contrarily to standard Supervised Learning, requires a continue interaction with data to update the policy. Usually, model-free RL is employed to optimize the policy, where the agent can only use the data sampled interacting with the real environment. In particular, RL algorithms with high sample complexity are non-manageable in a real-world scenario due to its computational cost. An alternative is presented by the so-called Model-Based Reinforcement Learning (MBRL) [3], where apart from optimizing policy the environment model must be learned. When using real-world data, having an environment model drastically reduces the complexity of generating new training samples for RL algorithms. In the following, we briefly overview the main components of MBRL. A more extensive treatment can be found in [3, 4, 5]. MBRL restates the problem as a Markov Decision Process (MDP) [2]. A MDP is a tuple $\{\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, p(s_0), \gamma\}$, where $\mathcal{S}$ and $\mathcal{A}$ are, respectively, the state and the action space, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to p(\mathcal{S})$ is the *transition function*, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the *reward function*, $p(s_0)$ is the initial state distribution and $\gamma \in [0, 1]$ is a discount parameter. A solution to a finite-time MDP consists of a policy $\pi^* : \mathcal{S} \to \mathcal{A}$ which maximizes the expected sum of rewards, that is

$$\pi^* = \arg\max_{\pi} Q^{\pi}(s, a) = \arg\max_{\pi} \mathbb{E}_{\pi,\mathcal{T}} \left[\sum_{k=0}^{K} \gamma^k r_{t+k} | s_t = s, a_t = a \right], \tag{1}$$

where $s_t \in \mathcal{S}, a_t \in \mathcal{A}, r_t := \mathcal{R}(s_t, a_t, s_{t+1})$.
MBRL approaches repeat the following two steps iteratively:

1. collect (or enlarge the current data with) a dataset $\mathcal{D} := \{(s_i, a_i, s_{i+1}, r_i, d_i)\}_{i=1}^{N}$, where d_i is the termination value indicating the end of the episode, and learn an approximate model f^* of the environment's dynamics fitting $\mathcal{D}$, *i.e.*

$$f^* = \arg\max_{f \in \mathcal{F}} \mathbb{E}_{(s_t, a_t, s_{t+1}) \sim \mathcal{D}} \mathcal{L}(s_{t+1}, f(s_t, a_t)), \tag{2}$$

   where $\mathcal{F}$ is a general family of functions (for instance Neural Networks) and $\mathcal{L}$ is a loss function. Note that with the same data reward and termination functions can be learned as well;

2. simulate the transitions with the learned model f^* and optimize the policy maximizing the expected sum of rewards like in (1).

This strategy allows to learn the dynamic model and optimize the policy at the same time.

3 Symbolic Regression

SR consists of finding a mathematical expressions that best fits a given dataset without assuming a particular structure of the expression. In this project, SR is handled with Genetic Programming (GP) [6, 7]. GP is an evolutionary strategy that explores a space of candidate models/functions by iteratively updating an initial population through genetic operations. A candidate model is also called *individual* and can be easily represented by a tree, in which each node is either a *primitive* (*i.e.* an operator) or a *terminal* (*i.e.* a variable or constant).

Initialization The population is typically initialized randomly. Some well-known techniques are the *full* and the *grow* method (see Section 2.1 in [7]), and a combination of those gives rise to *ramped half and half*. Each of them assumes that the starting population contains individuals that do not exceed a fixed maximum depth.

Selection Usually genetic operations are performed on individuals that are selected based on fitness, *i.e.* that are "probably good". One way to do so is *tournament selection*. This consists of performing a tournament for each k (tournament size) individuals, competing in terms of their fitnesses. The higher the score in these tournaments, the higher the probability for these individuals to be chosen.

Crossover and Mutation Genetic operations consists of *crossover* and *mutation*. The most common crossover used is the *subtree crossover* (or *one-point crossover*): given two parent trees, a random subtree and a crossover point are chosen for both; then, two new trees are generated, named *offsprings*, replacing the subtree rooted in the first parent with the one in the second parent and vice-versa. Instead, the most common mutation creates an offspring for a given parent tree by replacing a (randomly chosen) subtree with a newly generated subtree.

4 Experiments and implementations details

SR for MBRL performance is assessed through 2 different experiments, inspired by [1]. In both of them, a comparison with MBRL through Neural Networks (NN) is made. The MBRL algorithm chosen for SR and NN is the Probablistic Ensembles with Trajectory Sampling (PETS) [12], implemented with MBRL-lib [4] for NN and Operon [13] for SR. The action optimizer used is the Cross-Entropy Method (CEM) [14]. The implementation of this project is open-source and it is accessible through a public GitHub repository[1]. The hyperparameters selection is based on [1].

4.1 Simple 1-dimensional MDP

The following simple 1-dimensional MDP, with episode length 10, is considered

$$s_{t+1} := s_t + a_t, \tag{3}$$
$$r_t := \cos(2\pi s_{t+1}) \exp(|s_{t+1}|/3). \tag{4}$$

In this case, the goal is learning both the dynamics eq. (3) and the reward function eq. (4). The training (500) and validation (100) transitions were collected by sampling uniformly in the action space [-1,1] (random policy). A single run of PETS was used for both SR and NN.

SR hyperparameters The following hyperparameters are used: 10 optimizer iterations, population size 5000, 10000 generations, `add,sub,mul,div,constant,variable,sin,exp,abs` as allowed symbols and 32 threads. Note that to further complicate the problem `cos` is not in the primitive set.

NN hyperparameters A standard Multi-Layer Perceptron (MLP) is considered with the following hyperparameters: 4 hidden layers with 200 nodes each, SiLU activation function, Adam optimizer with 2000 epochs, batch size 256, patience epochs 25, learning rate 7.5×10^{-4} and weight decay 3×10^{-5}.

Action Optimizer CEM is used with planning horizon 3, 10 iterations, elite ratio 0.1, population size 1000, alpha 0.1 and clipped normal action distribution.

Results Figure 1 demonstrates that SR learned solution generalizes better outside the training region, while the NN model overfits in the training distribution. SR learned the following models (rounded to the first 3 decimal digits):

$$s_{t+1} = 1.0 s_t + 1.0 a_t, \tag{5}$$
$$r_t = 1.0 \exp(|0.333 s_{t+1}|) \sin(6.283 s_{t+1} - |-0.023 s_t + \sin(0.023 s_t) + 1.571|). \tag{6}$$

Note that eq. (3) coincides with eq. (3), while eq. (6) is slightly different than eq. (4). However, the first of the two factors matches completely with the ground truth, while the latter is approximately $\sin(2\pi s_{t+1} + 1.571)$ using that $6.283 \approx 2\pi$ and that $\sin(x) \approx x$ for x sufficiently close to 0. A similar model for the reward function was also found in the original paper [1].

[1] https://github.com/Smantii/symbolic_mbrl

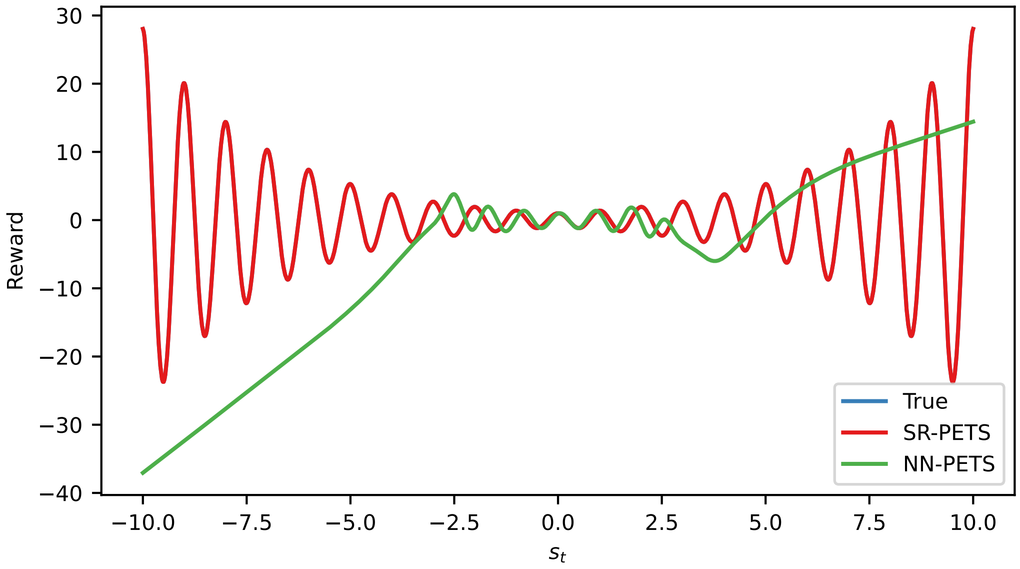

Figure 1: Reward function, evaluated with $a_t = 0$, predicted by the learned models

4.2 Cartpole

The continuous Cartpole [10, 11] is considered, where the agent's action a influences the system's state based on the following equations [9]:

$$\ddot{\theta} = \frac{g\sin(\theta) + \cos(\theta)\left(\frac{-K_{\text{mag}}a - m_p l \dot{\theta}^2 \sin(\theta)}{m_c + m_p}\right)}{l\left(\frac{4}{3} - \frac{m_p \cos^2(\theta)}{m_c + m_p}\right)}, \tag{7}$$

$$\ddot{x} = \frac{K_{\text{mag}}a + m_p l(\dot{\theta}^2 \sin(\theta) - \ddot{\theta}\cos(\theta))}{m_c + m_p}, \tag{8}$$

where $g, K_{\text{mag}}, m_p, m_c$ and l are constants. The agent state is $s_t := (x_t, \theta_t, \dot{x}_t, \dot{\theta}_t)$, and the task is purely related to dynamics modeling, since both termination and reward functions are known to the agent [4, 12].
The initial training (10) and validation (40) transition were collected using a random policy. Then, PETS was run for 10 iterations, and in each of those 10 transitions where added to the training data using the current policy. This procedure was repeated 3 times and the results were averaged accordingly.

SR hyperparameters As in 4.1 but with allowed symbols `add,sub,mul,div,constant, variable,sin,cos,square`.

NN hyperparameters As in 4.1.

Action Optimizer CEM is used with planning horizon 15, 5 iterations, elite ratio 0.1, population size 350, alpha 0.1.

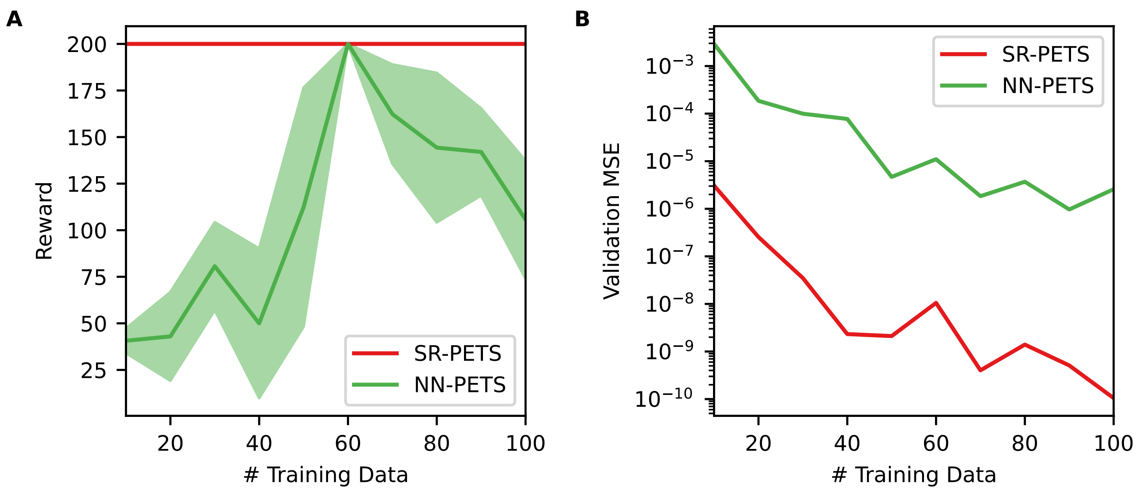

Figure 2: Evolution of cumulative reward (A) and validation MSE (B) as the number of training transitions increases. The results were averaged over 3 episodes with 3 random seeds.

Results Also in this case SR outperforms NN, being able to learn a good controller using only 10 training transitions (Figure 2A). The SR models found at the end of a random PETS episode are the following

$$x_{t+1} = 1.0x_t + 0.020\dot{x}_t, \tag{9}$$
$$\theta_{t+1} = (\theta_t - 0.014\dot{x}_t + 0.195a_t)\cos(0.228\dot{x}_t), \tag{10}$$
$$\dot{x}_{t+1} = 1.0\dot{x}_t + 0.020\dot{\theta}_t, \tag{11}$$
$$\dot{\theta}_{t+1} = 0.315\dot{x}_t + 1.0\dot{\theta}_t - 3.87a_t\cos(0.311\dot{x}_t) + 3.577a_t - 0.000. \tag{12}$$

Even though we can recognize some similarities with the ground-truth eq. (7)-(8), *e.g.* the time-discretization interval 0.02, the SR models are slightly different. This can be explained by the fact that a good candidate can have large validation MSE (Figure 2B).

5 Conclusion and discussion

This project demonstrates that SR-based MBRL can be more effective than standard MBRL in simple yet challenging environments, discovering models with many less parameters and better generalization capabilities. Also, interpretability of SR models can help scientists to manage and explain more easily the learned dynamics/rewards, paving the way to use cheaply and effectively these models in real-world scenarios.

References

[1] P.A. Kamienny, S. Lamprier, "Symbolic-Model-Based Reinforcement Learning". *NeurIPS 2022 AI for Science: Progress and Promises*, 2022.

[2] M. L. Puterman, "Markov Decision Processes: Discrete Stochastic Dynamic Programming." *John Wiley and sons*, 2014.

[3] T.M. Moerland, J. Broekens, A. Plaat, C.M. Jonker, "Model-based Reinforcement Learning: A Survey". *Foundations and Trends in Machine Learning*, 2023.

[4] L. Pineda, B. Amos, A. Zhang, N.O Lambert, R. Calandra, “Mbrl-lib: A Modular Library for Model-Based Reinforcement Learning”. *arXiv preprint arXiv:2104.10159*, 2021.

[5] F. Luo, T. Xu, H. Lai, X. Chen, W. Zhang, Y. Yu, “A Survey on Model-Based Reinforcement Learning”. *Science China Information Sciences*, 2024.

[6] J.R. Koza, “Genetic programming as a means for programming computers by natural selection”, *Statistics and Computing*, 1994.

[7] R. Poli et al., “A Field Guide to Genetic Programming”. Published via `http://lulu.com` and freely available at `http://www.gp-field-guide.org.uk`, 2008.

[8] A.G. Barto, R.S. Sutton,C.W. Anderson, “Neuronlike adaptive elements that can solve difficult learning control problems”. *IEEE transactions on systems, man, and cybernetics*, 1983.

[9] R.V Florian, “Correct equations for the dynamics of the cart-pole system”. *Center for Cognitive and Neural Studies (Coneural)*, 2007.

[10] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, W. Zaremba, “Openai gym”. *arXiv preprint arXiv:1606.01540*, 2016.

[11] E. Todorov, T. Erez, Y. Tassa, “Mujoco: A physics engine for model-based control”. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012

[12] K. Chua, R. Calandra, R. McAllister, Sergey. Levine, “Deep reinforcement learning in a handful of trials using probabilistic dynamics models”. *Advances in Neural Information Processing Systems*, 2018.

[13] B. Burlacu, G. Kronberger, M. Kommenda, “Operon C++: An efficient genetic programming framework for symbolic regression”. *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion, GECCO* 2020.

[14] P.T. De Boer, D.P. Kroese, S. Mannor, R.Y. Rubinstein, “A tutorial on the cross-entropy method”. *Annals of operations research*, 2005.