# Symbolic Model-Based Reinforcement Learning

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

October 12, 2024

## 1 Introduction

## 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. In real-world scenario, 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 reward as well;

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

# 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 our case, 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. In our method, $k = 2$ and we assign a non-zero probability to pick the loser instead of the winner in each tournament. This is calibrated in the hyperparameter selection phase.

**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

Add intro + libraries used

## 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). add details on training data used

**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 complicate the problem `cos` is not in the primitive set.

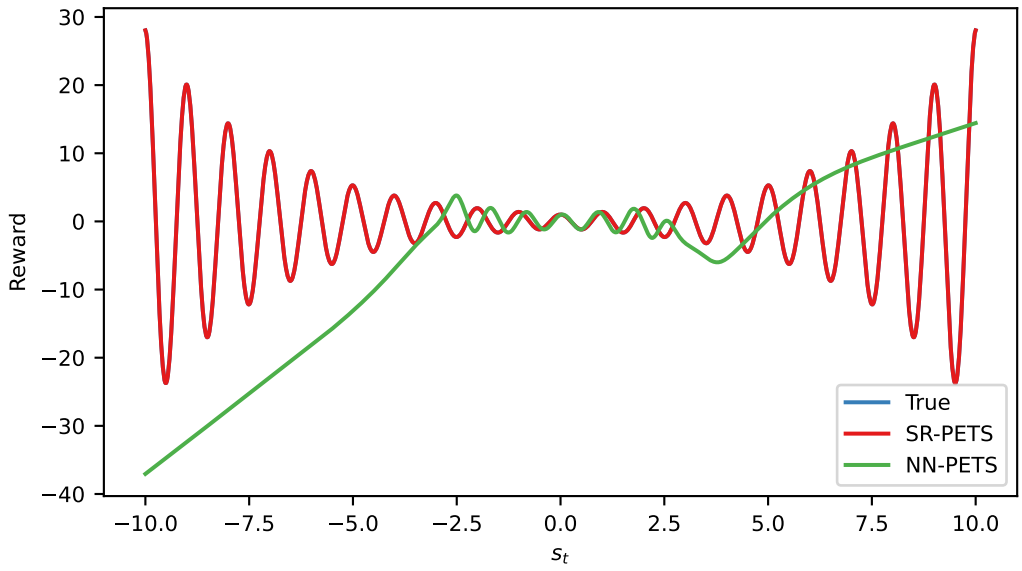


Figure 1: Reward function, evaluated with $a_t = 0$, predicted by the learned models

**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 $7.5e-4$ and weight decay $3e-5$.

**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 $x > 0$ sufficiently small. A similar model for the reward function was also found in the original paper [1].

## 4.2 Cartpole

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

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

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

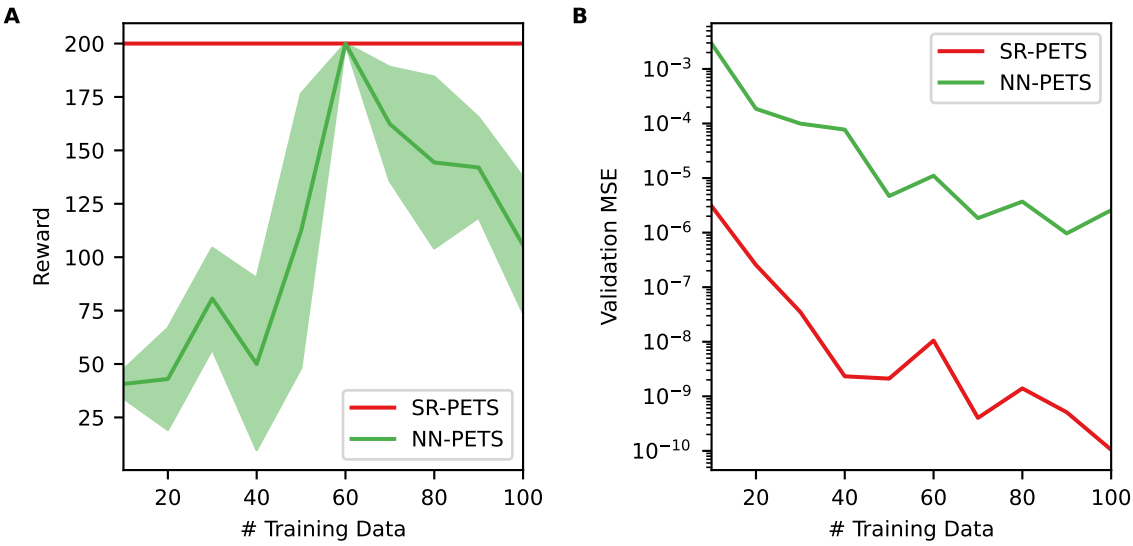


Figure 2: ??

**NN hyperparameters** As in 4.1.

**Results**

## 5 Conclusion and discussion

## 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*, vol. 4, 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.