Solving the One-Hole One-Ball Patience Cube Task using Model-based Reinforcement Learning

Minjae Kim[1] and Han Ul Yoon[2]

[1]*Dept. of Computer Science*, *Yonsei University, Mirae Campus*, Wonju 26493, Korea, mjkim00@yonsei.ac.kr

[2]*Dept. of Software*, *Yonsei University, Mirae Campus*, Wonju 26493, Korea, huyoon@yonsei.ac.kr

Abstract

This paper proposes an approach to solve an one-hole one-ball(OHOB) patience cube task using model-based policy optimization (MBPO), which is one of conventional algorithm to implement model-based reinforcement learning (MBRL). The OHOB patience cube task is to place a small ball into a hole located on a plane center. Even though model-free reinforcement learning (RL) algorithm have shown standout performance in a broad range of applications; however, it requires a prolonged learning time to explore a reward functional and a corresponding optimal policy. In contrast, the MBRL has been known to accelerate a learning time by reducing a number of interactions with the environment. In this study, we present an approach to estimate a dynamics model using neural network, and generate short-length "branched rollouts" under the estimated dynamics model to enable an efficient policy training. Experimental results show that the MBPO outperforms hte model-free RL in terms of sample efficiency and performance.

Deep Reinforcement Learning, Model-based Reinforcement Learning, Model-based Policy Optimization

1 Introduction

RL algorithms can generally be categorized into two approaches: model-free RL versus model-based RL. The former, learns the policy without explicitly learning the state transition model according to the environment's dynamics, whereas the latter learns the dynamics model of the environment to predict future states, the corresponding rewards, eventually derive an optimal policy. Many existing studies have shown the promising performance of the model-free RL. For example, Minh, et al showed that an RL trained by low-pixel images marked distinguished score in Atari games [1], and Haarnoja, et al successfully controlled multi-jointed robots in a continuous environment by applying the model-free RL [2]. However, these algorithms have some drawbacks that require a vast number of samples to train the RL agent.

In contrast, the model-based RL generates synthetic data via learned dynamics model. Is is known to reduce the interaction with environment, which in turn accelerates the training speed [4]. For instance, the Dyna algorithm proposed by Sutton [5], showed that sample efficiency can be enhanced by repeatedly learning dynamics model, generating data, and policy iteration. However, the performance of the MBRL directly depends on the estimation accuracy of the dynamics model; hence, it is often combined with model-free RLs [6].

In this paper, we address the OHOB patience cube task, which is the one example of challenging visuomotor control problems. The OHOB Patience cube is a transparent plastic cube in which a small ball I placed on a plate with a hole at the center. The given task is place the small ball into the hole. Although the task can be thought as simple one by taking a glance, it requires dexterous control due to a tiny workspace (1.5 in by 1.5 in) and fine motor control requirements. Therefore, we apply the MBPO [7], which is known to improve sample efficiency while maintaining high performance, to solve the OHOB patience cube. We also performed a comparative analysis, MBPO versus model-free RL, to substantiate the outperformance of the MBPO.

2 Model-based Reinforcement Learning

2.1 Preliminary

Throughout this paper, we consider a Markov decision process (MDP), defined by the tuple $(\mathcal{S}, \mathcal{A}, p, r, \gamma, \rho_0)$, in which $\mathcal{S}$ and $\mathcal{A}$ are state and action spaces, and γ is the discount factor. The transition probability denoted

as $p(s'|s,a)$, the initial state distribution as $\rho_0(s)$, and the reward function as $r(s,a)$. The goal of the RL is to find an optimal policy π^* that maximizes the expected sum of discounted reward:

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)\right].$$

2.2 Dynamics learning via neural network

Typically, a term "the dynamics model" refers to a function that, takes a current state $\mathbf{s}_t$ and an action $\mathbf{a}_t$ as an input, and returns a next state $\mathbf{s}_{t+1}$ as output. However, if the difference of the current state $\mathbf{s}_t$ and the next state $\mathbf{s}_{t+1}$ is small, then the dynamics learning process of the employed neural network becomes problematic. By following [6], hence, we define the dynamics model f_θ, which is represented by the neural network, to predict the difference between $\mathbf{s}_{t+1}$ and $\mathbf{s}_t$ over time Δt instead:

$$\hat{\Delta}_{t+1} = f_\theta(\mathbf{s}_t, \mathbf{a}_t). \tag{1}$$

By eq 1, the prediction of the next state $\hat{\mathbf{s}}_{t+1}$ can be retrieved by adding $\hat{\Delta}_{t+1}$ to $\mathbf{s}_t$

$$\hat{\mathbf{s}}_{t+1} = \mathbf{s}_t + \hat{\Delta}_{t+1}. \tag{2}$$

The loss is defined as a form of the mean square error (MSE) loss:

$$\begin{aligned} L(\theta) &= \sum_{\mathbf{s},\mathbf{a},s_{t+1}\in\mathcal{D}} ||(\mathbf{s}_{t+1} - \mathbf{s}_t) - f_\theta(\mathbf{s}_t, \mathbf{a}_t)||_2^2 \\ &= \sum_{\mathbf{s},\mathbf{a},s_{t+1}\in\mathcal{D}} ||\Delta_{t+1} - \hat{\Delta}_{t+1}||_2^2. \end{aligned}$$

In practice, the target $\Delta_{t+1} := s_{t+1} - s_t$ should be normalized; hence, we have the following loss function:

$$L(\theta) = \sum_{s,\mathbf{a},s_{t+1}\in\mathcal{D}} ||\text{Normalize}(s_{t+1} - s_t) - f_\theta(s_t, \mathbf{a}_t)||_2^2. \tag{3}$$

Since the neural network f_θ has been trained to output the normalized state difference, we predict the next state as follows:

$$\mathbf{s}_{t+1} = \mathbf{s}_t + \text{Unnormalize}(f_\theta(\mathbf{s}_t, \mathbf{a}_t)). \tag{4}$$

2.3 Action selection

Based on the learned dynamics model, we define the optimal action a^* which minimizes/maximizes the cost/reward function:

$$a_t^* = \arg\min_{\mathbf{a}_{t:\infty}} \sum_{t'=t}^{\infty} c(\hat{s}_{t'}, \mathbf{a}_{t'}) \tag{5}$$

However, in practice, it is rather computationally inefficient to directly calculate equation 5 due to the following reasons. First, it is almost impossible to plan for an infinite-time action sequence. Second, since the learned dynamic model is the estimated difference between the current state and the true next state, the estimation error will be accumulated (in fact, increases linearly) by using an open loop planning. By the above-mentioned reasons, there exists a possibility of that the current state is transitted to unexpected or unknown state. Therefore, we reformulate 5 so that the action is planned for finite time horizon H as follows:

$$A^* = \arg\min_{\{\mathbf{A}^{(0)},\cdots,\mathbf{A}^{K-1}\}} \sum_{t'=t}^{t+H-1} c(\hat{s}_{t'}, \mathbf{a}_{t'}) \tag{6}$$

The $\mathbf{A}^{(k)} = (a_t^{(k)}, \cdots, a_{t+H-1}^{(k)})$ is a random action sequence of which length is H. By applying eq 6, K-random action sequences $A^{(0)}, \cdots, A^{(k-1)}$, will be generated to predict the future states, which in turn allows us to select optimal action sequence $\mathbf{A}^*$ by evaluating the cummulative cost/reward.

2.4 Model predictive control with on-policy data

By adopting the method introduced in Sec. 3.2, we can obtain the optimal action sequence $\mathbf{A}^*$ for a short horizon. For a long horizon, however, the accumulated error between the current state and the true next state will be increasing. To mitigate the above-mentioned issue, we employ model predictive control (MPC) [8].Intead of executing the all actions in $\mathbf{A}^*$, the MPC allows us to excute the next step action and replan through all time steps. Learning the dynamic model f_θ from the data collected under random policy might degrade a state estimation performance. Therefore, the data will be added while running the MPC to collect the data with on-policy manner. Consequently, the model-based RL with the MPC can be summarized as Algorithm 1.

Algorithm 1: Model-Based RL with On-Policy Data

```
Run based policy π_0(a_t, s_t) to collect
  D = {(s_t, a_t, s_{t+1})}
while not done do
    Train f_θ using D;
    s_t ← current state;
    for rollout number r = 0 to R do
        for timestep t = 0 to T do
            A* = π_MPC(a_t, s_t);
            a_t ← first action in A*;
            Execute a_t and get next state s_{t+1};
            Add (s_t, a_t, s_{t+1}) to D
        end
    end
end
```

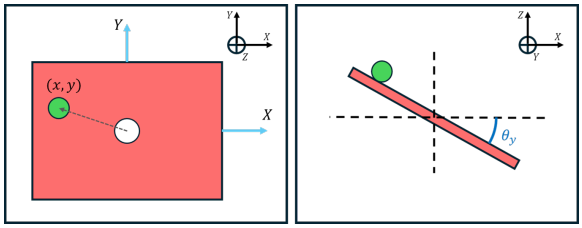

Figure 1: x-y plane of OHOB cube (left), x-z plane (right)

In addition, to improve the prediction accuracy of the dynamics model, we will train N-independent neural networks and average the N-network outputs.

3 An Approach to Solve the OHOB Patience Cube Task

3.1 The OHOB patience cube task

Fig. 1 illustrates the one-hole one-ball (OHOB) patience cube. Let (x, y) be a position of the ball with respect to the hole of a horizontal x-y plane, thus the hole is regarded as an origin (0,0), and $(\dot{x}, \dot{y})$ is the linear velocity of the ball. Also, let θ_x, θ_y be the orientation of the plane along x-axis and y-axis, and $(\dot{\theta}_x, \dot{\theta}_y)$ are the angular velocity. Now, we define the state of the RL agent for the OHOB patience cube problem

$$\mathbf{s} = \left[x, \dot{x}, \theta_y, \dot{\theta}_y, y, \dot{y}, \theta_x, \dot{\theta}_x\right].$$

The agent action $\mathbf{a} = [a_x, a_y]^T = [\tau_y, \tau_x]^T$ are torque which rotates the plane along x and y axes. According to [9], a human task-performing strategy for solving a visuomotor task can be modeled as an optimal controller to regualte the quadratic form of a reward function. Therefore, we define the reward function $r(\mathbf{s}, \mathbf{a})$ as

$$r(\mathbf{s}, \mathbf{a}) = -[a_k^T R a_k + s_k^T Q s_k] + r_K$$

where r_K is a final reward. The weight matrix of reward function R, Q are constructed as

$$R = \begin{bmatrix} c_{ax} & 0 \\ 0 & c_{ay} \end{bmatrix}, \quad Q = \begin{bmatrix} c_p & 0 & 0 & 0 & \cdots & 0 \\ 0 & c_v & 0 & 0 & \cdots & 0 \\ 0 & 0 & c_\theta & 0 & \cdots & 0 \\ 0 & 0 & 0 & c_\omega & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & c_\omega \end{bmatrix}$$

The subscripts represent the correspondence of each coefficeint; hence, ax and ay stand for the actions along x and y axes; so p, v, θ, ω do a position, a linear velocity, an orientation, an angular velocity, respectively (see Kim, et al. [10] for details).

$$c_{ax} = 0.0201, \quad c_{ay} = 0.0201$$
$$c_p = 1.04, \; c_\theta = 1.2538, \; c_v = 0.0001, c_\omega = 0.0004.$$

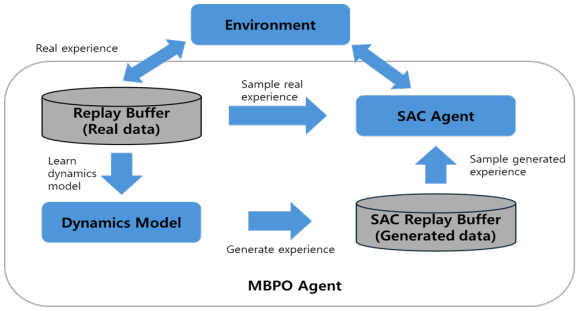

Figure 2: MBPO agent learning framework

3.2 Model-Based Policy Optimization

Fig. 2 shows the MBPO agent learning framework. The MBPO mitigates the problematic issues of the conventional model-based RL method, especially the accumulating error of the dynamics model and model exploration, by learning a dynamics model as well. The MBPO redeploys the learned dynamics model during a policy optimization process; which in turn improves a sampling efficiency and outperforms model-free RL algorithm. The algorithm 2 presents the MBPO algorithm.

Algorithm 2: Model-Based Policy Optimization

```
Initialize policy π_φ, predictive model f_θ,
  environment dataset D_env, model dataset
  D_model ;
for N epochs do
    Train model f_θ on D_env ;
    for E steps do
        Take action in environment by π_φ; add
          to D_env ;
        for M model rollouts do
            Sample s_t uniformly from D_env;
            Perform k-step model rollout using
              π_φ; add to D_model;
        end
        for G gradient updates do
            Update policy on model data:
              φ ← φ − λ_π ∇̂_φ J_π(φ, D_model);
        end
    end
end
```

The MBPO algorithm utilizes a learned dynamics model not only as a sole replacement of the environment, but also as an auxiliary tool to improve a data sampling efficiency. By restricting the model rollouts into a short horizon, which is referred to as short-length "Branched Rollouts," the MBPO effectively mitigates a model bias and prevents an error accumulation.

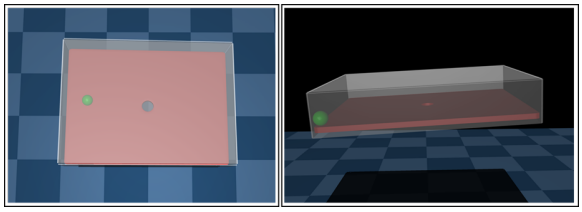

Figure 3: OHOB Patience cube environment in MuJoCo

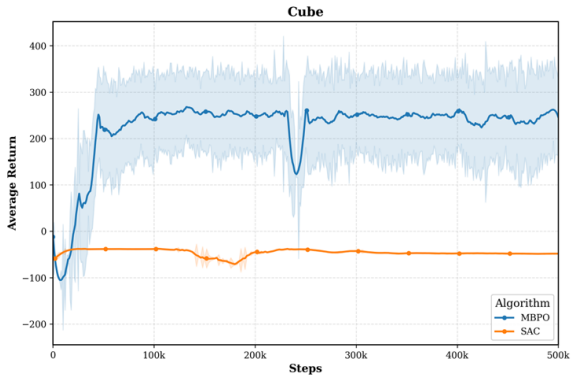

Figure 4: Performance Comparison of MBPO and SAC on the Cube Task

4 Simulation Result

Fig. 3 depicts the environment of the OHOB patience cube experiment. A comparative analysis, the MBPO agent versus the SAC agent, was performed based on the simulation result using MuJoCo simulator. Fig. 4 shows the average returns of the two different agents while solving the OHOB patience cube task. As shown in the Fig. 4, the MBPO agent notably outperformed the SAC agent due to sampling efficiency.

We note that, sometimes, the subtle errors of dynamics model might cause the large fluctuations of the resulting trajectory, which makes the standard deviation of the MBPO performance to be higher than that of SAC.

5 Conclusion

Throughout this paper, we verified the effectiveness of the MBPO, a model-based reinforcement learning algorithm, for solving the OHOB patience cube task, which is the one example of the visuomotor tasks. The MBPO could successfully solve the given OHOB task by learning a dynamics model as well as finding an optimal policy with efficient sampling during a restricted horizon. Based on the MuJoCo simulation result, we perfomed the comparative analysis between the MBPO agent and the SAC agent, which showed the outperformance of the MBPO.

Acknowledgment

This research was partially supported by the Technology Innovation Program funded by the Ministry of Trade, Industry & Energy (MOTIE, Korea) (Grant No. RS-2024-00418941) and the Regional Innovation System & Education(RISE) program through the Gangwon RISE Center, funded by the Ministry of Education(MOE) and the Gangwon State(G.S.), Republic of Korea(2025-RISE-10-006).

References

[1] V. Mnih et al. “Human-level control through deep reinforcement learning,” Nature, vol. 518, no. 7540, pp. 529–533, 2015.

[2] T. Haarnoja et al. “Soft actor-critic: off-policy maximum entropy deep reinforcement learning with a stochastic actor,” in Proceedings of the 35th International Conference on Machine Learning, Stockholm, Sweden, pp. 1861–1870, July, 2018.

[3] Schulman, John, et al. “Trust region policy optimization,” Proceedings of the 32nd International Conference on Machine Learning, Lille, France, pp. 1889–1897, July, 2015.

[4] M. Deisenroth and C. Rasmussen, “PILCO: A model-based and data-efficient approach to policy search,” Proceedings of the 28th International Conference on Machine Learning, Bellevue, WA, USA, pp. 465–472, June, 2011.

[5] R. Sutton, “Dyna, an integrated architecture for learning, planning, and reacting,” ACM SIGART Bulletin, vol. 2, no. 4, pp. 160–163, 1991.

[6] A. Nagabandi et al. “Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning,” Proceedings of the 2018 IEEE International Conference on Robotics and Automation (ICRA), Brisbane, Australia, pp. 7559–7566, May, 2018.

[7] M. Janner et al. “When to trust your model: Model-based policy optimization,” Advances in Neural Information Processing Systems, vol. 32, pp. 12519–12530, 2019.

[8] A. Richards, Robust Constrained Model Predictive Control, PhD Thesis, Massachusetts Institute of Technology, Cambridge, MA, United States, June, 2005.

[9] Todorov, Emanuel, and Michael I. Jordan. ”Optimal feedback control as a theory of motor coordination.” Nature neuroscience 5.11 (2002): 1226-1235.

[10] J. Kim et al. “An approach to design a biomechanically-inspired reward function to solve a patience cube under reinforcement learning

framework," Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Detroit, MI, USA, pp. 1234–1239, October, 2023.