Learning Off-Policy with Online Planning

Harshit Sikchi [1] **Wenxuan Zhou** [1] **David Held** [1]

Abstract

We propose Learning Off-Policy with Online Planning (LOOP), combining the techniques from model-based and model-free reinforcement learning algorithms. The agent learns a world model, and then uses trajectory optimization with the learned model to select actions. To sidestep the myopic effect of fixed horizon trajectory optimization, a value function is attached to the end of the planning horizon. This value function is learned through off-policy reinforcement learning, using trajectory optimization as its behavior policy. Furthermore, we introduce "actor-guided" trajectory optimization to mitigate the actor-divergence issue in the proposed method. We benchmark our methods on continuous control tasks and demonstrate a significant improvement over the underlying model-based and model-free algorithms.

1. Introduction

Off-policy reinforcement learning is a widely used category of model-free reinforcement learning. It usually aims to learn a value function that encapsulates long-horizon reasoning of the future reward. The policy is obtained by directly taking the action that has the largest action-value (Kalashnikov et al., 2018) in discrete action spaces or by using a parameterized actor (Lillicrap et al., 2015) in continuous settings. The effectiveness of the value function makes model-free reinforcement learning achieve state-of-the-art performance. However, it usually requires a huge amount of interactions with the environment.

Model-based reinforcement learning provides a mechanism for an agent to learn to perform a task by building a model of the environment through experience. It can scale to highly complex tasks while being orders of magnitude more sample efficient than model-free algorithms (Janner et al., 2019) (Chua et al., 2018). One way to use the learned model is to perform online planning with the model (Nagabandi et al., 2019) during both training and testing. At each timestep, the agent selects the best action by imagining possible rollouts with its learned model using Model Predictive Control (MPC). However, the number of timesteps it looks into the future is usually fixed to a small number. This is because the required computation for planning grows exponentially with the horizon. At the same time, the accuracy of the learned model usually deteriorates with longer horizons (Feinberg et al., 2018). Thus, this method often suffers from myopic decisions for complex tasks.

To combine the advantages of sample efficiency of model-based reinforcement learning and long-horizon reasoning of model-free reinforcement learning, we propose Learning Off-Policy with Online Planning (LOOP). During online planning, it augments the model-based rollout trajectories with a terminal value function learned using off-policy model-free reinforcement learning. We refer to this policy as the MPC policy. In this way, the agent can select actions by evaluating short-horizon model rollouts as well as the future consequences after the rollout terminates using the value function. From another perspective, this model-based planning agent can be treated as the behavior policy for the off-policy algorithm that it uses to obtain the value function. Thus, compared to the underlying off-policy algorithm, LOOP can interact with the environment more effectively by planning over the learned model.

This combination introduces an issue in learning the value function. Directly estimate the value function for the MPC policy is computationally inefficient. Instead, if we use the MPC policy as a behavior policy and learn the value-function using an off-policy algorithm, the parameterized actor that is used to update the value function might diverge from the behavior policy and cause the "extrapolation issue" (Fujimoto et al., 2018a) in Q-learning. We identify this divergence to be critical in this combination of model-based and model-free reinforcement learning. We propose actor-guided trajectory optimization which improves learning performance by guiding trajectory optimization using the model-free actor. We evaluate our method on OpenAI Gym MuJoCo environments and demonstrate that the final algorithm is able to learn more efficiently than the underlying model-based and off-policy method.

[1] Carnegie Mellon University, Pittsburgh, US.

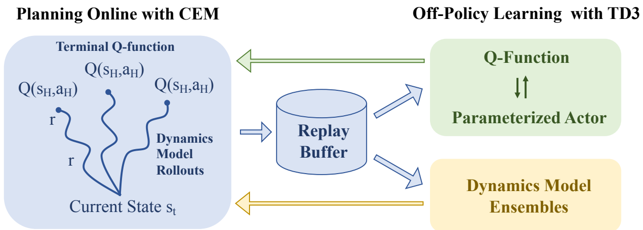

Figure 1. Overview: We use online planning with learned dynamic models and a terminal Q-function as the behavior policy. The transitions are saved into the replay buffer to train the Q-function and the dynamics model. The parameterized actor is also used to guide online planning.

2. Related Work

Model-free reinforcement learning algorithms achieve high performance for a lot of tasks, but these methods are notoriously sample-inefficient. Particularly, on-policy methods like TRPO (Schulman et al. (2015)) and PPO (Schulman et al. (2017)) require new samples to be collected for every update to the policy. Off-policy methods like SAC (Haarnoja et al., 2018) and TD3 (Fujimoto et al., 2018b), on the other hand, are more sample-efficient than on-policy methods, as they utilize all the experiences obtained in the past.

Model based-RL has seen a surge of interest recently, as the benefits involve reducing the sample complexity while maintaining asymptotic performance. Previous work approach model-based reinforcement learning using a learned model and trajectory optimization (Chua et al., 2018; Nagabandi et al., 2019). These methods can reach asymptotic performance when a large enough planning horizon is used. They can also scale to complex tasks like rotating Baoding balls in hand, but have the limitation of not being able to reason for rewards beyond the planning horizon. Increasing the planning horizon increases the number of trajectories that should be sampled, and incurs a heavy computing cost.

Another line of work attempts to get the best of both model-free and model-based reinforcement learning, Feinberg et al. (2018) and Buckman et al. (2018) uses the model to improve target value estimates and thus accelerates model-free reinforcement learning. Schrittwieser et al. (2019) uses Monte-Carlo Tree search with value estimates to constrain the length and uses a policy to constrain the width of the search tree. Their method utilizes on on-policy samples to train the Q-function, making it sample inefficient. It works on discrete action spaces with a latent dynamics structure. (Hamrick et al., 2019) combines MCTS with Q-learning but work under the setting of known model and discrete space.

The most related work to ours is Lowrey et al. (2018) where they use trajectory optimization in the form of Model Predictive Control (MPC) as the behavior policy, and updates the Q function by obtaining a target lookahead value estimate using another instance of trajectory optimization. This method is extremely slow as each batch of sampled data for training the Q-function will require instances of trajectory optimization that scales with batch size. Moreover, they consider access to ground truth dynamics. Our method uses trajectory optimization using a learned model and a terminal value function as an exploratory policy, but the Q function updates are performed entirely off policy.

3. Learning off-policy with online planning

In LOOP, we use trajectory optimization with a terminal value function as the behavior policy that interacts with the environment. This trajectory optimization in its naive form is myopic and in many cases may not produce optimal policies for the task, since it does fixed horizon planning using a learned model. We address this deficiency by having a value function that reasons for the expected long term rewards under the policy. Different from Lowrey et al. (2018), we propose to utilize advances in off-policy learning to make the algorithm computationally efficient. In this section, we start by discussing the methods for learning the value function and the dynamics model. After that, we will further discuss how these two are combined and how we deal with the actor-divergence issue. Going forward, we will use the notation π_{opt} to denote the MPC policy that uses trajectory optimization by forward simulation over learned models in the model-based part and π_ϕ to denote the parameterized actor in the model-free part.

3.1. Learning Q-Function with the Model-free Actor

To learn a value function, we build our method upon TD3 (Fujimoto et al., 2018b) which is an off-policy algorithm with an actor π_ϕ and a value function Q_θ. Note that other off-policy algorithms can also be used here. We refer to the actor π_ϕ used to update the value function as the "parameterized actor", in the sense that this actor is a parameterized function, in our case a neural network. It will only be used to update the value function and not be used to collect data as standard TD3. The target value is calculated based on the bellman equation:

$$Q^{\text{target}}(s_t, a_t) = r(s_t, a_t) + \gamma Q'_\theta(s_{t+1}, \pi_\phi(s_{t+1})) \quad (1)$$

To reduce overestimation error, TD3 calculates the target value by taking the minimum over two Q-functions Q_{θ_1} and Q_{θ_2}, also called Clipped Double Q learning. Both Q-functions are updated by minimizing the mean squared error:

$$\text{MSE} = E_{(s,a)\sim D}[Q^{\text{target}}(s, a) - Q_{\theta_1,\theta_2}(s, a))]^2 \quad (2)$$

where s, a are the state action pairs sampled from D, the replay buffer of past experiences. The policy is updated by maximizing $Q_{\theta_1}(s, \pi_\phi(s))$.

3.2. Learning the Dynamics Model

Using the transitions collected, we train a dynamics model using supervised learning. Given a state-action pair (s, a), the network is trained to regress the difference δ between the next state and the current state, parameterized as a Gaussian distribution with a diagonal covariance matrix. We use probabilistic ensembles of dynamics models that capture both epistemic and aleatoric uncertainty in forward predictions (Chua et al., 2018). Each model in the ensemble is initialized with different weights and samples shuffled batches of data during training.

3.3. Trajectory Optimization with a Terminal Value Function

Given the dynamics model, the trajectory optimization policy π^{opt} is the MPC-based policy that uses Cross-Entropy Method (CEM) to select actions. CEM is a strong optimizer and is shown (Nagabandi et al., 2019) to perform much better than the random-shooting methods. This MPC policy will be used as the behavior policy to collect data in the environment. For each timestep, this policy will sample N action sequences $(a_1, a_2, ...a_H)$, up to a fixed horizon from a sampling distribution, and use the probabilistic dynamics model to unroll the trajectory resulting from the action sequence. The cumulative return for each rollout is calculated by

$$G = \sum_{i=0}^{H-1} (\gamma^i r(s_i, a_i)) + \gamma^H Q_\theta(s_{H+1}, a_{H+1}) \tag{3}$$

Note that in previous model-based reinforcement learning methods such as (Nagabandi et al., 2019)(Chua et al., 2018), the last term $Q(s_H, a_H)$ is not present, resulting in an optimization of the action sequences over a fixed horizon, which might be shortsighted and result in a suboptimal trajectory sequence. The top e highest scoring actions sequences, also called elites, are selected and used to refine the sampling distribution from which the action sequences are sampled from.

$$\begin{aligned} A_i = \{a_0^i, a_1^i, .., a_H^i\}, A_i &\sim \mathcal{N}(\mu^m, \Sigma^m) \, \forall i \in N \\ A_{\text{elites}} &= \text{sort}(A_i)[-e:] \\ \mu^{m+1} &= \alpha * \text{mean}(A_{\text{elites}}) + (1-\alpha)\mu^m \\ \Sigma^{m+1} &= \alpha * \text{var}(A_{\text{elites}}) + (1-\alpha)\Sigma^m \end{aligned} \tag{4}$$

After M iterations of refinement, we take the mean of the resulting action distribution as the final output. Following MPC, only the first action of the sequence is executed. For each subsequent timestep, replanning is performed. The transitions will be collected into the replay buffer, which will be used to train the models and the Q-function.

3.4. Actor-guided Trajectory Optimization (CEM-AG)

Combining trajectory optimization and off-policy learning as described above might suffer from the issue of "actor divergence": There is a mismatch between the state-action distribution induced by the model-free actor and the state-action distribution of the MPC-based behavior policy that collects data. As discussed in Fujimoto et al. (2018a), the mismatch will lead to extrapolation errors in Q-learning and cause overestimation bias that deteriorates performance. In our experiments, we observe that naively using the method in section 3.3 sometimes results in worse performance than the original off-policy method due to this issue.

To mitigate the issue of actor divergence, we propose Actor-guided Trajectory Optimization. At each timestep, we use the model-free actor to propose a sequence of trajectories by rolling out the dynamics model. These trajectories will be included in the batch of samples in each CEM iteration. When these trajectories are selected as elites in CEM, they will guide the optimization and bring the solution closer to the actor distribution. In our experiments, we observe that CEM-AG effectively decreases actor divergence and improves the performance of LOOP.

One alternative to solve the issue of actor divergence is to incorporate the divergence as an additional cost in CEM using Kullback-Leibler (KL) divergence or L_2 distance. In practice, we found this to be overly conservative and limits the performance benefits from trajectory optimization.

4. Experimental Results

We benchmark the proposed method LOOP on four OpenAI Gym MuJoCo environments: HalfCheetah, Hopper, Walker, and InvertedPendulum. First, we evaluate the sample efficiency and performance of LOOP comparing to the algorithms that it is built upon. We further analyze the effect of Actor-guided CEM in reducing actor divergence.

4.1. Implementation Details

We use a horizon length of 5 for all the environments except Walker (uses horizon=3). We use the author's implementation of TD3 with the original hyperparameters. The dynamics model ensemble has 5 neural networks each consisting of 4 hidden layers, 200 hidden units each. We list all the hyperparameters in Appendix 7.1. The CEM optimizer uses 200 particles, sampled from a multivariate Gaussian distribution. Each action sequence is passed through each of the dynamics models and the average return is used as the maximization objective in CEM. The MPC policy will be used for the evaluation of LOOP. For CEM-AG we use 1 trajectory from π_ϕ for every 20 trajectories from sampling distribution. We use 5 random seeds to account for variability in training.

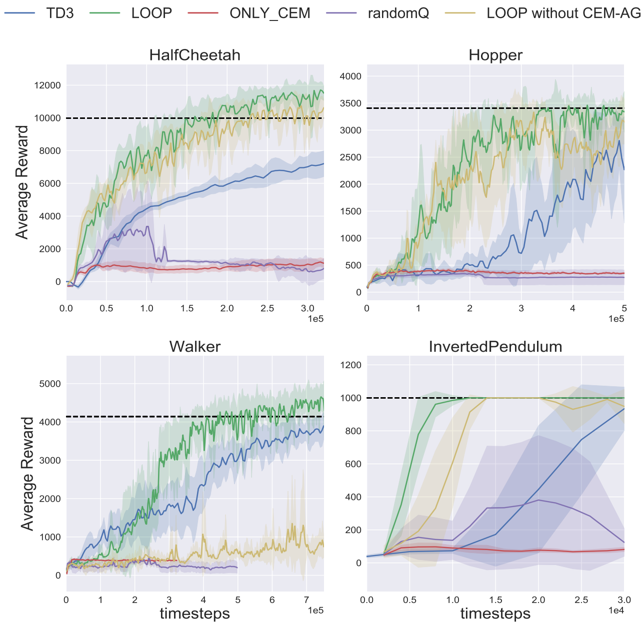

Figure 2. Training Performance of LOOP and its baselines on MuJoCo tasks. Dashed line indicates the performance of TD3 at 1e6 timesteps.

4.2. Improvement over the underlying model-based and model-free algorithm

As shown in Figure 2, LOOP (green curves) has significant performance gains over TD3 in all of the four environments due to better exploration. The red curves show the performance of CEM with the same planning horizon as LOOP but without the terminal Q-function, similar to Nagabandi et al. (2019). We observe that fixed-horizon CEM performs poorly in MuJoCo tasks due to the short planning horizon. In Appendix 7.6, we further show that LOOP has comparable performance to SOTA model-based methods.

4.3. Ablation Studies

To better understand the importance of different components, we do ablation studies on the actor-guided CEM and the parameterized actor. Using a simple CEM instead of the Actor-guided CEM (CEM-AG), Walker completely fails. The performance of other environments also drops and sometimes becomes unstable. There are two hypotheses of how actor-guided CEM helps. First, the trajectories proposed by the parameterized actor might sometimes provide a better solution to this optimization problem than CEM alone. However, we observe that actor-guided CEM usually achieves similar imagined reward as original CEM (Appendix 7.5). Second, actor-guided CEM biases the solutions of CEM towards the parameterized actor, and thus reduces extrapolation issue for off-policy learning. In Figure 3, we compute the L_2 distance between the action proposed by the MPC policy (final output from CEM) and the TD3 policy. We observe that actor-guided CEM in LOOP is indeed able to reduce the actor-divergence compared to normal CEM. In Appendix 7.3, we plot the "actor usage" in CEM by measuring the fraction of time when the trajectories suggested by the parameterized actor are selected to be the elites in CEM. It further demonstrates that the actor-proposed trajectories are biasing the sampling distribution of CEM. In addition, we show that using a Q-function trained with a random policy performs poorly in LOOP framework indicating that the parameterized actor provides meaningful information for MPC policy to reason beyond the planning horizon.

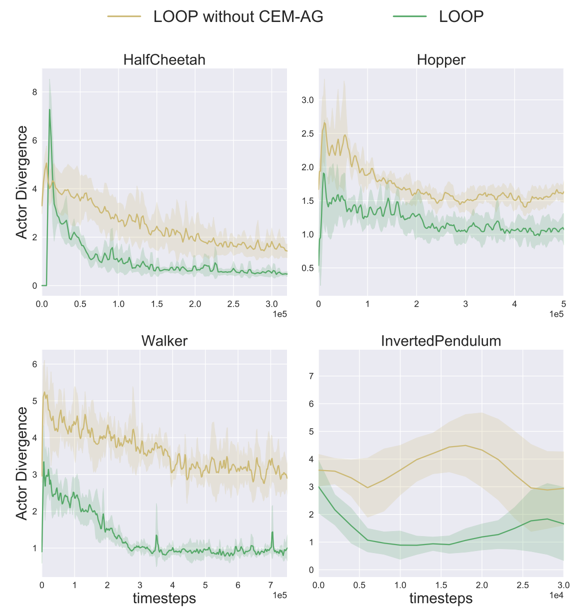

Figure 3. Actor-guided CEM reduces actor-divergence between the MPC policy and the model-free actor.

5. Conclusion

In this work, we present a novel method that combines model-free and model-based reinforcement learning. It allows model-based online planning to reason about long-horizon cumulative returns. From another perspective, it improves upon the model-free algorithm by using a more efficient behavior policy. We highlight the issues present in applying a terminal Q-function to the online planning methods and present the actor-guided solution. From the experiments, we demonstrate that LOOP improves the performance and sample-efficiency over the underlying model-based and model-free method.

6. Future Work

As discussed above, we observe that the parameterized actor in our method has poor performance. We present our attempted solution in Appendix 7.4. We look forward to improving the performance of parameterized actor to further improve LOOP. Another interesting direction is to use Offline-RL methods (Levine et al., 2020) to reduce the actor-divergence issue more systematically.

References

Buckman, J., Hafner, D., Tucker, G., Brevdo, E., and Lee, H. Sample-efficient reinforcement learning with stochastic ensemble value expansion. In *Advances in Neural Information Processing Systems*, pp. 8224–8234, 2018.

Chua, K., Calandra, R., McAllister, R., and Levine, S. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems*, pp. 4754–4765, 2018.

Feinberg, V., Wan, A., Stoica, I., Jordan, M. I., Gonzalez, J. E., and Levine, S. Model-based value estimation for efficient model-free reinforcement learning. *arXiv preprint arXiv:1803.00101*, 2018.

Fujimoto, S., Meger, D., and Precup, D. Off-policy deep reinforcement learning without exploration. *arXiv preprint arXiv:1812.02900*, 2018a.

Fujimoto, S., Van Hoof, H., and Meger, D. Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477*, 2018b.

Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Pfaff, T., Weber, T., Buesing, L., and Battaglia, P. W. Combining q-learning and search with amortized value estimates. *arXiv preprint arXiv:1912.02807*, 2019.

Janner, M., Fu, J., Zhang, M., and Levine, S. When to trust your model: Model-based policy optimization, 2019.

Kalashnikov, D., Irpan, A., Pastor, P., Ibarz, J., Herzog, A., Jang, E., Quillen, D., Holly, E., Kalakrishnan, M., Vanhoucke, V., et al. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *arXiv preprint arXiv:1806.10293*, 2018.

Levine, S., Kumar, A., Tucker, G., and Fu, J. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning, 2015.

Lowrey, K., Rajeswaran, A., Kakade, S., Todorov, E., and Mordatch, I. Plan online, learn offline: Efficient learning and exploration via model-based control. *arXiv preprint arXiv:1811.01848*, 2018.

Nagabandi, A., Konoglie, K., Levine, S., and Kumar, V. Deep dynamics models for learning dexterous manipulation. *arXiv preprint arXiv:1909.11652*, 2019.

Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., et al. Mastering atari, go, chess and shogi by planning with a learned model. *arXiv preprint arXiv:1911.08265*, 2019.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. Trust region policy optimization. In *International conference on machine learning*, pp. 1889–1897, 2015.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Wang, T. and Ba, J. Exploring model-based planning with policy networks. *arXiv preprint arXiv:1906.08649*, 2019.