Model-Based Reinforcement Learning for Control

1st Sean Campbell
University of Colorado Boulder
sean.campbell-1@colorado.edu

2nd Komal Porwal
University of Colorado Boulder
komal.porwal@colorado.edu

***Abstract*—We implement a model-based reinforcement learning (MBRL) scheme and evaluate its performance across several control tasks in the MuJoCo simulator. We compare our approach to model-free methods like Proximal Policy Optimization (PPO). On a simpler benchmark such as the Inverted Pendulum, our MBRL method attains comparable performance with substantially fewer environment interactions. For a moderately difficult problem such as the Hopper benchmark, we are able to learn a reasonable policy using fewer samples than a model-free baseline. However, for difficult benchmarks such as the Ant locomotion task, our policy struggles to find a good policy. While unimpressive, our MBRL results are better than those reported in [1] which we initially used for inspiration.**

I. Introduction

Model based reinforcement learning (MBRL) promises great things if you can get it to work. Namely, it promises high sample efficiency and rapid adaptation to new tasks. By explicitly learning a dynamics model $\hat{f}_\theta(s_t, a_t) \approx s_{t+1}$, MBRL agents can predict future outcomes and plan accordingly. In theory, if your dynamics model is good enough, you can use it solve new tasks with known reward functions without further interactions with the environment.

In contrast, model-free reinforcement learning methods, such as PPO and SAC, directly optimize policies or value functions from raw experience and typically require large numbers of interactions with the environment to learn good policies.

Unfortunately, the effectiveness of MBRL is constrained by two major forms of uncertainty: epistemic uncertainty due to limited data and aleatoric uncertainty due to the inherent stochasticity in the environment. These forms of uncertainty make it difficult to plan using the dynamics model because you don't know when it's completely wrong.

II. Background and Related Work

Our project is mostly based on the MBRL approach described in Nagabandi et al. [1] though we also attempt the probabilistic ensembling method of Chua et al. [2].

Nagabandi et al. [1] train neural networks to predict state changes:

$$\hat{f}_\theta(s_t, a_t) \approx s_{t+1} - s_t.$$

They integrate this model into Model Predictive Control (MPC) using random shooting for action selection. They evaluate their model on a handful of the easy to moderate MuJoCo environments and find that their approach significantly under performs model-free baselines. Thus, they use a form of imitation learning where they train a network on MPC trajectories and then use it to warm start the policy network in a model free algorithm (TRPO). This combined approach lets them achieve much greater sample efficiency than TRPO alone. However, their purely model based approach showed very poor performance. Based on our own experiments and the numbers reported in their appendix, their MBRL and cloned policies were either worse or comparable to 0 action policies (i.e polices that just emit action vectors of all 0s) for 2 out of 4 of their evaluation benchmarks.

Chua et al. [2] extend this by employing an ensemble of probabilistic dynamics models (i.e. neural networks that predict mean and log variance instead of making point predictions) with the idea that the ensembles reduce epistemic uncertainty and the probabilistic networks reduce aleatoric uncertainty. They use the Cross Entropy Method (CEM) with MPC and report decent results on 4 of the simpler MuJoCo tasks.

In Janner et al. [3], they propose Model-Based Policy Optimization (MBPO) in which they train model free algorithms using short simulated rollouts from their dynamics model. This hybrid approach improves sample efficiency and they are able to solve several of the harder MuJoCo tasks.

In [4], the authors make several observations regarding MPC based MBRL. First, because MPC rollouts lack a value estimate of the final state at the end of the prediction horizon, they necessarily will find locally optimal solutions even with a good dynamics model. Secondly, MBRL learning does not know which dynamics of the system are important for achieving good performance since it may be the case that many state features are irrelevant. Thus, they propose Temporal Difference Model Predictive Control (TD-MPC). In TD-MPC, they learn a latent representation of the state, a Q function, a reward function, and a policy function simultaneously. The reward and Q function are used with MPC to select good action sequences and they alternate between performing MPC to collect more episodes and updating their model parameters. For the MPC step, they augment it by adding a terminal value function (their learned Q function) and use a custom version of Model Predictive Path Integral (MPPI) for action selection. Despite all of this complexity, they show much greater sample efficiency on 92 Deepmind Control tasks when compared to model free baselines. These control tasks include the hardest MuJoCo tasks.

There are several other papers that explore the core idea of merging model free and model based methods by learning latent dynamics while also learning some sort of value function. For example, in DreamerV3 [5], the authors learn a latent dynamics model but do away with MPC and replace it with an actor critic formulation. They show strong results on a number of difficult RL benchmarks.

Finally, while not as closely related to this project, there are a number of papers that look at how to derive or learn a terminal value function for MPC in order to mitigate the problem with using a limited horizon in MPC. For example, [6] and [7].

A. Problem Formulation

Our objective is to solve control tasks using pure MBRL. We aim to learn dynamics models of the form

$$s_{t+1} = s_t + \hat{f}_\theta(s_t, a_t) \tag{1}$$

(where $\hat{f}_\theta(s_t, a_t)$ is a neural network) by collecting sequences of (s_t, a_t, r_t, s_{t+1}) from MuJoCo environments. Following the standard RL formulation, we wish to maximize the expected sum of discounted rewards. Using the learned model, we perform MPC by solving

$$(a_t, \ldots, a_{t+H-1}) = \arg \max_{a_{t:t+H-1}} \sum_{k=0}^{H-1} \gamma^k r(\hat{s}_{t+k}, a_{t+k}), \tag{2}$$

and execute only the first action before repeating the process. We assume the reward function is known and fixed.

III. Solution Approach: MBRL and MPC

We begin with random environment interactions to train a dynamics model using Eq. 1. This network is optimized via gradient descent. We then use MPC to select actions.

Like Chua et al. [2], we compared random shooting and CEM and found CEM to be better. However, both methods required large sample sizes to find better than random actions for the more difficult benchmarks. Thus, they were exceptionally slow to use and we decided to use gradient-based action optimization, which was both faster and more effective in our experiments. Consequently, all experiments use gradient descent to solve Eq. 2 and assume a differentiable reward function. In MuJoCo, the reward functions are usually not differentiable and so we approximate them with differentiable alternatives (details in Section V).

We evaluate both single neural networks and ensembles of neural networks for modeling dynamics. We explore several architectural and hyperparameter variants including fairly standard 2-3 layer feedforward networks as well as probabilistic networks.

We assess model quality via multi-step rollout error and use an iterative training procedure: starting with random trajectories, we train a dynamics model, use MPC to collect improved data, and retrain the model. This is summarized in Algorithm 1 and is nearly identical to algorithms presented in [1] and [2]. The main difference between our approach and theirs is our MPC algorithm.

Algorithm 1 Iterative Model Learning with MPC

```
Initialize dataset $\mathcal{D} \leftarrow \{\}$
Collect random trajectories and add to $\mathcal{D}$
for iteration $= 1$ to $N$ do
    Train model $\hat{f}_\theta$ on $\mathcal{D}$
    Use MPC to generate new data
    Add new transitions to $\mathcal{D}$
end for
```

IV. Solution Approach: Imitation Learning

After training a dynamics model and using MPC to collect trajectories, we train a policy network via behavior cloning. Specifically, we use the dataset of state-action pairs (s_t, a_t) collected from the MPC controller and optimize the policy parameters to minimize the mean squared error between the policy's predicted action and the MPC action for each state:

$$\min_\theta \sum_{(s_t, a_t) \in \mathcal{D}} \|\pi_\theta(s_t) - a_t\|^2 \tag{3}$$

where π_θ is the policy network parameterized by θ.

Once the imitation policy is trained, we use its weights to initialize ("warm start") the policy network of a model-free RL agent (e.g., PPO). We then continue training this agent in the environment.

V. Results

A. A Brief Comparison of MPC Methods

While CEM and random shooting worked fine for the pendulum problem, we found that gradient based optimization was greatly superior to both for the Hopper problem. The graphs below summarize our results for a typical dynamics model we trained. Random shooting even with 1000 samples over a horizon 10 was not better than just randomly selecting actions. CEM was much better but still terrible. While it is possible that we would have gotten better results for random shooting and CEM had we vastly increased the number of samples used, our implementations were not particularly fast and we did not have the patience to wait longer.

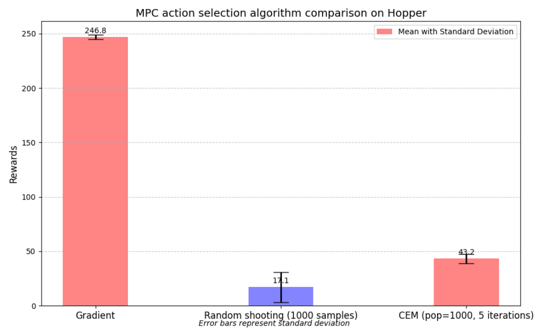

Fig. 1.

B. Pendulum

This benchmark requires balancing an inverted pendulum for 1000 steps. Each step that the pendulum remains within 0.2 radians of vertical yields +1 reward. The action space is one-dimensional, and the observation space includes positions and velocities. To enable gradient-based planning, we approximate the reward as $r(t) = -\theta^2$, where θ is the pendulum angle.

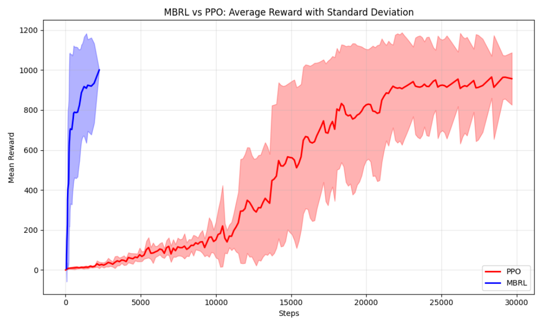

Fig. 2. Learning curve for Pendulum: MBRL vs PPO.

Our MBRL approach solves this task significantly faster than PPO with default settings. Using a single network and planning horizon $H = 1$, we achieved similar rates of learning with fewer samples. Longer horizons (up to $H = 10$) yielded comparable performance.

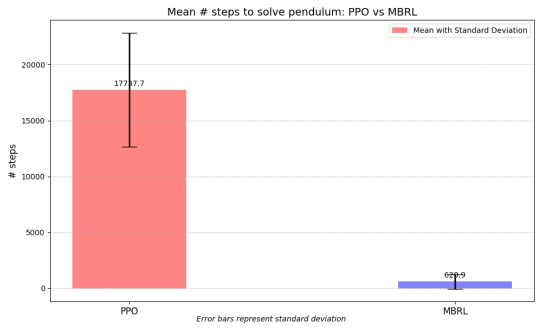

Fig. 3. Average steps to reach 1000 reward over ten runs. PPO takes roughly 30x more steps than MBRL

C. Hopper

The Hopper task involves a one-legged robot with three continuous actions and seven observations. The goal is to move forward without falling. The true reward is

$$r_t = r_{\text{healthy}} + \frac{x_{t+1} - x_t}{\Delta t} - w_{\text{control}} \|a_t\|^2, \tag{4}$$

where $w_{\text{control}} = 0.001$. For gradient-based planning, we approximate this reward with:

$$r_z = \tanh\big(k(z - z_{\min})\big), \tag{5}$$

$$r_\theta = \tanh\big(k(\theta_{\max} - |\theta|)\big), \tag{6}$$

$$r_{\text{forward}} = v_x \cdot r_z \cdot r_\theta, \tag{7}$$

$$r_t = r_{\text{forward}} + r_z r_\theta. \tag{8}$$

where:

- k is a constant that we manually set. We found 20 worked fine.
- z is the torso height.
- θ is the torso angle.
- v_x is the forward velocity.
- The first two terms penalize unhealthiness based on soft thresholds: low height or large torso tilt where θ_{max} is the max torso angle before becoming unhealthy and z_{min} is the minimum torso height before becoming unhealthy.
- The third term rewards forward velocity, scaled by a soft health indicator.

Despite reaching average rewards of 200-300 (depending on model variant) in under 1000 steps, our method plateaus and we see no improvement after another 50k steps while PPO eventually exceeds rewards of 1000 in 500k steps (though PPO takes 30k steps to reach a reward of 250). Nagabandi et al. [1] report rewards of 100 with their MPC MBRL. Random actions yield 16, and a zero-action policy scores 150 (i.e. [1]'s MBRL approach was not better than just slowly falling over). We also experimented with different horizons and found that horizons of 5-10 performed somewhat better than shorter horizons. Additionally, we saw no benefit from using probabilistic neural networks (not shown) and only a minor improvement from using an ensemble of networks. These results are summarized in the table below.

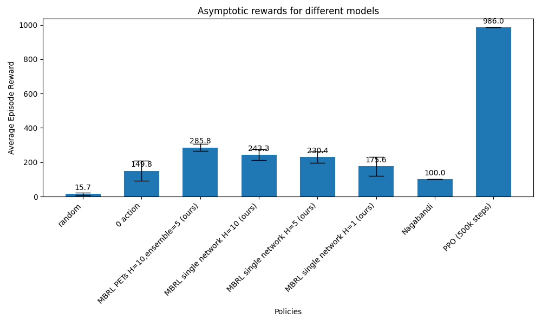

Fig. 4. We compare our models across different horizons (H). We also try the ensembling method in [2] with probabilistic networks (PETS). Nagabandi are the MPC rewards reported in [1]. PPO is the mean reward achieved after 500k steps using the PPO algorithm. If run for longer, PPO continues to improve.

We investigated why our model failed to learn better and believe it is because the dynamics function fails to learn dynamics far away from its usual starting position. We examined typical trajectories and it appears that our model simply learns to do one or two hops in the x direction and then promptly falls over. Our error analysis shows accurate one-step predictions but much larger error over longer horizons. Below are some typical results for an episode where we plot values for the 3 physical quantities that impact episode reward: torso height, torso angle, and x velocity. Note that these are absolute values. The relative values tell the same story but are larger and noiser.

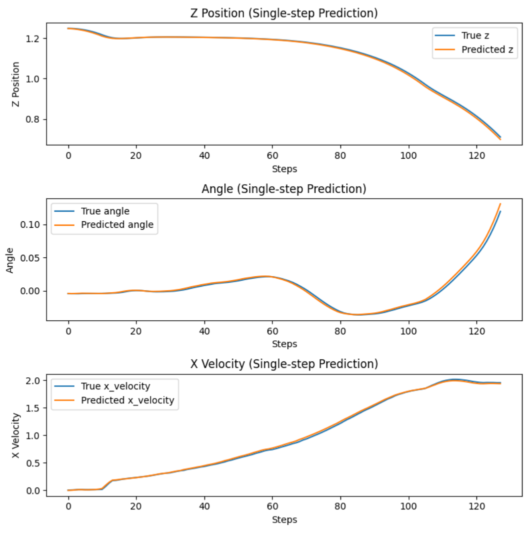

Fig. 5. One-step model predictions vs. ground truth.

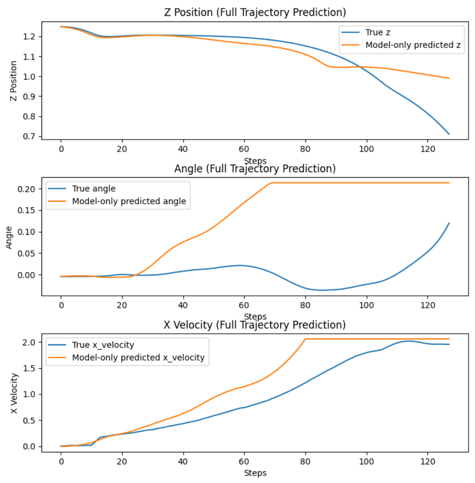

Fig. 6. Predictions from initial state with increasing horizon length. The predictions within about 20 steps are reasonably accurate which suggests that the model learns to model the initial conditions well.

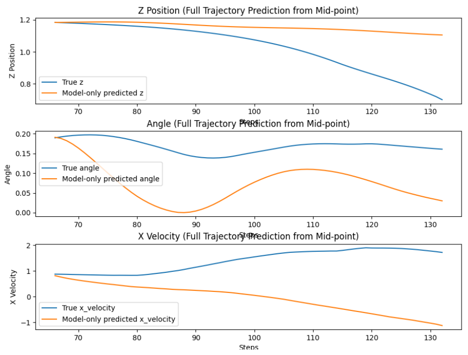

Fig. 7. Predictions from step 65 of an episode with increasing horizon length. Here we see that the predictions are much less accurate (essentially useless) and suggests that the model struggles to learn dynamics farther from the episode's beginning.

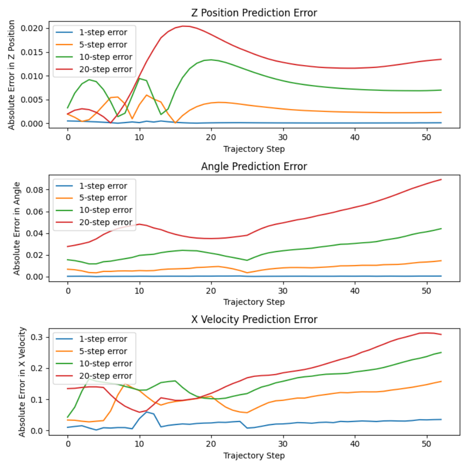

Fig. 8. Absolute prediction error over time for different horizons. As the episode progresses, the prediction errors get worse.

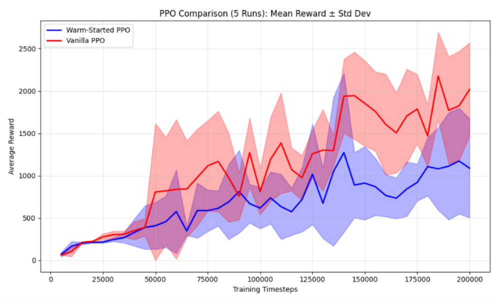

Fig. 9. Warm started PPO (ours) initially performs better than vanilla PPO but fails to match vanilla PPO's long term performance.

Finally, we tried imitation learning in an attempt to improve upon the above results. We collected MPC trajectories, trained an initial policy network, and warm started the policy network in PPO. This initial policy network scores about 180 which is worse than our best MPC runs but better than the imitation policy reported in [1]. In Figure 9, we compare the performance of Vanilla PPO and our Warm-Started PPO variant across five runs, evaluating mean reward and standard deviation over training timesteps. Initially, the Warm-Started PPO outperforms Vanilla PPO due the learned MPC policy network. However, this initial advantage appears to bias the learning process toward suboptimal policy regions, resulting in stagnation and poorer long-term performance. In contrast, the Vanilla PPO, though slower to improve, achieves higher final rewards. One potential explanation could be that the MPC-based warm start constrained the exploration space too early, anchoring the policy in a narrow, possibly suboptimal solution space. Additionally, differences in the underlying network architectures or initialization strategies between the two implementations could have contributed to the divergent learning trajectories. It is worth noting that this result is in contrast to what was reported in [1] where they saw improved sample efficiency due to warm starting. However, they did use a more complex, iterative imitation learning strategy (DAgger), a probabilistic neural network architecture, and TRPO instead of PPO which in aggregate might change results (though we are skeptical).

VI. Ant

Given our poor results on Hopper, we decided to spend less time on this task since it is in many ways more difficult (e.g. higher dimensionality). The same issues we had with Hopper likely apply here as well. The Ant task involves a four-legged robot with 8 continuous actions and 27 continuous observations. The goal is to move forward without falling. Like the Hopper benchmark, the true reward is a weighted combination of a forward motion reward, a control penalty, and a +1 for staying healthy. Unlike the hopper, it is much easier to stay balanced. Thus, even random actions sometimes will lead to the ant staying upright for an entire episode. The real challenge is to learn an efficient policy to move the ant in the positive x direction. We approximate the true reward with a differentiable approximation in nearly the exact same way as for the hopper but appropriately adjusted to the task specifications. We trained several variations of our model but found that our gradient based action selection method got stuck finding the locally optimal solution which is to minimize action outputs, jitter slightly in the x direction, and just not fall over. We were unable to mitigate this with changes to the dynamics network architecture, horizon length, or using curriculum learning approaches (e.g. slowly increasing episode length). Once again, we are able to do 10x "better" than the results reported in [1] though that is not exactly inspiring. As our model free baseline, we used SAC with default parameters since PPO struggled to learn a good policy after a million steps.

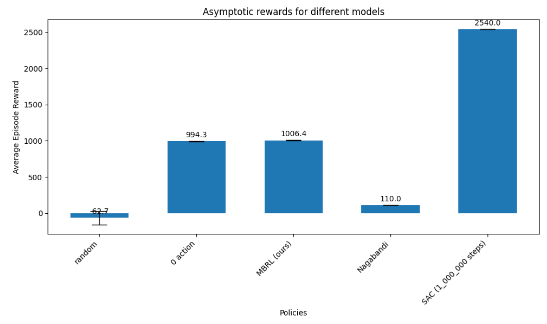

Fig. 10. We compare a single simple feedforward model with a horizon of 10 against SAC, 0 action, random action, and the results reported in [1]

Like for Hopper, we also tried imitation learning and found that the warm starting did not improve PPO. In fact, PPO itself struggled to learn a good policy. This is not particularly surprising given that our MPC trajectories were not very good. However, [1] reports a large improvement in sample efficiency despite a low scoring imitation learned policy. We are unsure of the root cause of this; however there are many hyperparameters that we could have potentially tuned more optimally.

VII. Conclusion

Basic MBRL is difficult. Based on our experiments, it does not achieve good performance out of the box. Perhaps there's some magic set of hyperparameters or curriculum learning or other tweak that would have made things work but we tried the most obvious changes and were unable to make substantial improvements. We believe that the results in [1] and [2] might be overly optimistic and find it telling that they do not report results on the more difficult control tasks, suggesting that their methods may not generalize easily. On the whole, our poor results seem to be mostly due to the difficulty of learning a dynamics model. That said, our implementation outperforms the MBRL models reported in Nagabandi et al. [1] on the Ant and Hopper benchmarks.

In contrast, model free RL algorithms produced high quality policies using default parameters for all of the MuJoCo benchmarks we tried, including the hardest ones like humanoid (not described here). In future work, it would be interesting to explore approaches that combine model free and model based learning as in TD-MPC [4] or DreamerV3 [5].

Finally, we tried several different variations on MBRL not described here including learning terminal value functions but nothing performed well. We are quite disappointed and would not recommend the pure MBRL approach to others.

Contributions

While we both worked on all aspects of the paper including research and solution methods, Sean focused on MBRL for Ant, Pendulum, and Hopper benchmarks. Komal focused on getting imitation learning and warm starting for Hopper/Ant working.

Permissions

The authors grant permission for this report to be posted publicly.

Algorithm implementation

We implemented all of the MBRL algorithms from scratch. SAC and PPO were taken from the Stable-Baselines3 Python package [8].

References

[1] A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine, "Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning," arXiv preprint arXiv:1708.02596, 2017.

[2] K. Chua, R. Calandra, R. McAllister, and S. Levine, "Deep reinforcement learning in a handful of trials using probabilistic dynamics models," Advances in Neural Information Processing Systems, vol. 31, 2018.

[3] M. Janner, J. Fu, M. Zhang, and S. Levine, "When to trust your model: Model-based policy optimization," Advances in Neural Information Processing Systems, vol. 32, 2019.

[4] N. Hansen, X. Wang, and H. Su, "Temporal Difference Learning for Model Predictive Control," Proceedings of the 39th International Conference on Machine Learning, vol. 162, pp. 8266-8276, 2022.

[5] D. Hafner, T. Lillicrap, J. Ba, and M. Norouzi, "Mastering Diverse Domains through World Models," Proceedings of the 11th International Conference on Learning Representations, 2023.

[6] N. Hatch and B. Boots, "The Value of Planning for Infinite-Horizon Model Predictive Control," arXiv preprint arXiv:2103.12069, 2021.

[7] L. Beckenbach, P. Osinenko and S. Streif, ”Model predictive control with stage cost shaping inspired by reinforcement learning,” 2019 IEEE 58th Conference on Decision and Control (CDC), Nice, France, 2019

[8] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, ”Stable-Baselines3: Reliable reinforcement learning implementations,” GitHub repository, 2021. [Online]. Available: https://github.com/DLR-RM/stable-baselines3