Model Based Deep Reinforcement Learning with Model Predictive Control

Reinplementing the "Deep Model-Based Reinforcement Learning for Predictive Control of Robotic Systems with Dense and Sparse Rewards" paper

Aaron L. Fernandes
Dept. of Intelligent Systems Engineering
Indiana University Bloomington
Indiana, United States
aalefern@iu.edu

Vishwajeet Ekal
Dept. of Computer Science
Indiana University Bloomigton
Indiana, United States
vekal@iu.edu

***Abstract*—This report outlines the development of a model-based reinforcement learning algorithm aimed at continuous control tasks, drawing inspiration from the Deep Value-and-Predictive-Model Control (DVPMC) framework. The goal of this implementation is to combine system identification, value function approximation, and sampling-based optimization for action selection. We have successfully demonstrated the effectiveness of Model Based Reinforcement on the Dense reward control tast but the sparse reward component has not been successfully implemented. This report also discusses the challenges encountered, including difficulties in accurately modeling the environment and optimizing sample efficiency, and suggests potential avenues for further research and implementation adjustments.**

Keywords—Reinforcement Learning

I. Introduction

Reinforcement learning (RL) in control has been an extensively studied area. However, the practical application of RL in real-world scenarios is still constrained by challenges such as sample efficiency and reward specification. Sample efficiency refers to the ability of an RL method to learn an effective control policy with minimal interactions. While deep reinforcement learning (DRL) has demonstrated superhuman performance in certain tasks, it typically requires millions of interactions with the environment to achieve this level of proficiency. This high demand for data makes the use of complex and computationally expensive simulations necessary before deploying the learned policies in real-world systems. Moreover, real-world systems, such as Unmanned Aerial Vehicles (UAVs), face limitations on the frequency of interactions with the environment, which restricts the data available for learning. For instance, a single mistake in operation could result in a UAV crashing and being damaged. In this context, model-based reinforcement learning (MBRL) algorithms have gained attention for their potential to improve sample efficiency by learning the environment's dynamics, making them a promising approach for real-world applications.

II. Model based methods

A. Background

We as humans can "imagine" a world in which it is possible to predict outcomes of our actions. Similarly, if an agent has the ability to "imagine", they can choose appropriate actions based on this imagination with minimal costs associated with trial and errors. This is what MBRL methods set out to achieve. The "Model" in Model Based methods refers to the model of the environment, or the dynamics model of the environment where the agent is expected to act. The dynamics model, in RL is formulated as a Markov Decision Process or simply referred to as MDP, where each entry is represented by the tuple (S,A,M,R,γ), where S is the state space, A is the action space, Gamma (γ) is the discounting factor, $M:S\times A\rightarrow S$ represents the state transition dynamics and $R:S\times A\rightarrow\mathbb{R}$ represents the reward function. The primary object of MBRLs is to learn the transition dynamics (M) and the reward function (R). In many instances, the reward function is explicitly defined, in these cases, the objective is to learn the transition dynamics (M).

B. Alternatives to MBRL

Model-Free Reinforcement Learning (MFRL) methods are extensively researched and have demonstrated strong performance in various tasks. These methods directly optimize policies or value functions without explicitly modeling the environment, making them conceptually simpler. Prominent approaches include off-policy methods, such as Deep Q-Networks (DQN)[2], which rely on replay buffers to reuse data for improved sample efficiency, and actor-critic algorithms, like Soft Actor-Critic (SAC)[3] and Advantage Actor-Critic (A2C)[4], which use a critic to guide the policy's learning. While MFRL methods have achieved significant successes, they often require millions of interactions with the environment, making them sample-inefficient. Furthermore, the coupling of value function approximation with the sampling policy limits their ability to generalize across diverse policies or simulate alternative scenarios. In contrast, Model-Based Reinforcement Learning (MBRL) leverages learned models of the environment's dynamics to predict

outcomes and evaluate multiple candidate policies, decoupling policy learning from environment interactions.

C. MBRL: Types and MPC

Model-Based Reinforcement Learning (MBRL) approaches can be broadly categorized into three main types: **Dyna-style algorithms**, **shooting algorithms**, and **policy search algorithms**.

Dyna-style algorithms leverage a learned model of the environment to generate synthetic data, which is then used to train a model-free agent. This hybrid approach combines the strengths of both model-free and model-based paradigms, improving sample efficiency by augmenting real experience with simulated data. Some notable examples include **Dreamer [5]** and **MBMF[6].**

Policy search algorithms learn a continuously differentiable dynamics model and value function. By utilizing the differentiable nature of the dynamics model, these methods analytically compute the gradient of the reinforcement learning objective with respect to the policy parameters, bypassing the need for sampling-based gradient estimation typical in deep reinforcement learning (DRL) algorithms. Important ones include **DreamerV2[7]** and **TRPO-MPC[8]**

Shooting algorithms use a learned dynamics model, a known reward function, and sampling-based optimization to directly plan actions for solving tasks. These algorithms are notable for their exceptional sample efficiency. They operate by learning a model of the environment, sampling action trajectories up to a prediction horizon, evaluating the trajectories using a reward function, and then executing the first action from the trajectory with the highest reward. The approach is conceptually similar to Model Predictive Control (MPC). However, most shooting methods assume access to the reward function and use their learned models autoregressively, leading to compounding prediction errors over time. This limitation makes them less effective for sparse reward problems, which often require evaluating long-horizon trajectories to determine whether an action sequence leads to a meaningful outcome.

Noteworthy examples of shooting-based MBRL algorithms include **PETS (Probabilistic Ensembles with Trajectory Sampling) [9]** and **PlaNet (Planning in Latent Space)[10]**. PETS uses an ensemble of probabilistic models to explicitly account for uncertainty in dynamics, improving robustness. PlaNet, on the other hand, maps inputs to a compact latent space and performs sampling-based trajectory optimization within this latent representation, enabling efficient long-horizon planning.

D. Model Predicitve Control

Model Predictive Control (MPC) is an optimization-based control strategy that plans actions by predicting future system behavior using a model of the environment. At each timestep, MPC solves an optimization problem over a finite prediction horizon (H) to determine a sequence of actions that maximize a defined objective, such as minimizing cost or maximizing reward. Only the first action of the optimized sequence is executed, and the process is repeated at the next timestep with updated state information, enabling it to adapt to changes in the environment in real time. MPC is particularly powerful for systems with complex dynamics, constraints on states or actions, and long-term dependencies. However, its reliance on accurate models and the computational cost of solving optimization problems at every timestep can limit its applicability, especially in high-dimensional or fast-changing environments. MPC framework has been employed many recent publications such as in Yang et al [12], where the RL-MPC was used for energy management in hybrid electric vehicles.

III. CONTIRBUTIONS

This project is mainly concerned about exploring the Model Predictive Control based approach outlined in the original paper. Rather than focusing on the "usefulness" of this particular method in a particular problem, the main focus of this project is to understand the fundamentals of modern MBRL systems. This report will also go on to address the challenges encountered and suggest potential remedies.

IV. The Method

The method proposed by the original paper combines model learning, value function approximation and a sampling based optimizer in a configuration analogous to MPC

a. System Identification

A Fully Connected Neural Network with Relu activations and batchnormalization layers is used to learn the forward-time dynamics of the system. The choice of architecture would depend on the problem; the type of state data. Recurrent NNs for temporal and CNNs for spacial. I chose a basic 3 layer FCNN because of the initial stage of the research. The model is trained on the data collected from the environment in the form of (s_t,a,s') to minimize the state difference s_t-s' rather than learning the state itself. We use the MSE loss function and the Adam optimizer.

For prediction horizons larger than a single time step, the output of the previous timestep would have to be fed back into the model to predict the future outputs. The model would then be autoregressive and hence the error would grow with the length of the prediction horizon. Hence a prediction horizon of one time step is used as in the original paper.

b. Value Function Approximation

Value Function Approximation is used to minimize the *H* for controlling a system. A mapping of the state to predicted discount returns is learnt under the current policy. The network is trained to minimize the Temporal Difference (TD) error. To improve sample efficiency, the experiences are stored in a Replay Buffer and every time step, a minibatch is uniformly sampled from this "Experience Replay Buffer" to train the value function network. In our implementation, the Value network is a simple 4 layer FCNN, for now.

c. Trajectory sampling:

The Cross-Entropy Method (CEM), an optimization technique used trajectory planning was used. It iteratively refines a distribution over possible solutions by sampling and selecting the best-performing candidates, based on a predefined objective or cost function. CEM works by maintaining a parameterized distribution (typically Gaussian) over the decision variables, and after each iteration, it updates the distribution parameters (mean and variance) based on the elite solutions (top performers) from the sampled trajectories. This process continues until convergence, enabling efficient optimization even in high-dimensional spaces. Other methods considered were Model Predictive Path Integral (MPPI) and sample-efficient CEM.

d. The Algorithm: DVMPC

Algorithm 1 DVPMC

Given:

- Number of episodes EP
- Environment env
- Flag g that is true if the reward is sparse and goal-based and false otherwise

Initialize replay buffer E
Initialize dataset $\mathcal{D}$
Initialize predictive model $\hat{f}^{\phi}$ parameterized by ϕ
Initialize value model $\hat{V}^{\theta}$ parameterized by θ
for `episode in EP` **do**
  Observe initial state $\mathbf{s}_t$ from env
  $t \leftarrow 0$
  $done_t \leftarrow False$
  while not $done_t$ **do**
    $\mathbf{u}_t \leftarrow CEM(\mathbf{s}_t, \hat{f}^{\phi}, \hat{V}^{\theta})$
    $\mathbf{s}_{t+1}, r_t, done_t \leftarrow env.step(\mathbf{u}_t)$
    Store $(\mathbf{s}_t, \mathbf{u}_t, \Delta\mathbf{s}_{t+1})$ in $\mathcal{D}$
    Store $(\mathbf{s}_t, \mathbf{u}_t, r_t, \mathbf{s}_{t+1})$ in E
    Sample a minibatch of N transitions from E
    Update $\hat{V}^{\theta}$ to minimize $L(\theta)$
    if $done_t$ **then**
      Update $\hat{f}^{\phi}$ to minimize
      $MSE = \frac{1}{N}\sum_{i=0}^{N}(\Delta\hat{\mathbf{s}}_{t+1}^{i} - \Delta\mathbf{s}_{t+1}^{i})^2 \forall \mathcal{D}$
    end if
    $\mathbf{s}_t \leftarrow \mathbf{s}_{t+1}$
  end while
  if g **then**
    for `t in T` **do**
      Relabel $(\mathbf{s}_t, \mathbf{u}_t, r_t, \mathbf{s}_{t+1}, g_t)$ with new goal g_t^a
      and r_t^a
      Store relabelled $(\mathbf{s}_t, \mathbf{u}_t, r_t, \mathbf{s}_{t+1}, g_t^r)$ in E
    end for
  end if
end for

This method is as explained in the original paper. The things to note here are:

1) Bootstrap target is only used when the next state is not terminal, to enforce zero value constraints on terminal states

2) The value function is trained on terminal states using a target of 0

V. Experiment

a. Setup

We used the Mujoco environments from Gymnasium: https://gymnasium.farama.org/environments/mujoco/

Specifically we considered the InvertedPendulumV5 environment.

1) Inverted Pendulum:

TABLE I. INVERTED PENDULUM

Action Space	**Box(-3.0, 3.0, (1,), float32)**
Observation Space	**Box(-inf, inf, (4,), float64)**
import	**gymnasium.make("InvertedPendulum-v5")**

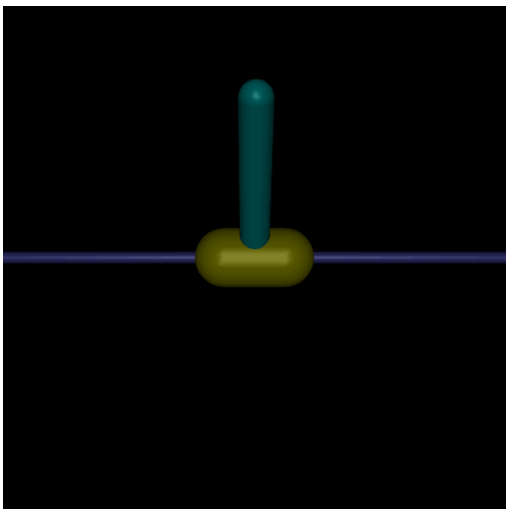

Fig. 1. A render of the Inverted Pendulum

This environment is a Mujoco-powered version of the classic Cartpole environment. While maintaining the fundamental setup of the classic environment, the use of the Mujoco physics simulator enables more intricate experiments, such as modifying gravity's effects. The setup involves a cart that moves linearly, with a pole attached at one end and free at the other. By applying forces to push the cart left or right, the goal is to balance the pole upright on the cart.

a) Action Space: The action space is a continuous (action) in [-3,3], where action is the numerical force applied to the cart.

b) Observation Space: The observation space consists of the following parts (in order):

i) qpos (2 element): Position values of the robot's cart and pole.

ii) qvel (2 elements): The velocities of cart and pole (their derivatives).

The observation space is a Box(-Inf, Inf, (4,), float64) where the elements are as follows:

Num	Observation	Min	Max	Joint	Type(Unit)
0	position of the cart along the linear surface	-inf	inf	slide	position(m)
1	vertical angle on the cart	-inf	inf	hinge	angle(rad)
2	linear velocity of the cart	-inf	inf	slide	velocity (m/s)
3	angular velocity of the pole on the cart	-inf	inf	hinge	angular velocity(rad/s)

c) Reward: The goal is to keep the pendulum upright (within a certain angle limit). +1 reward for every time step.

II. Experimental results:

The experiment was run for until the the stabilizing policy was reached. It took around 1000 episodes to reach a stable policy.

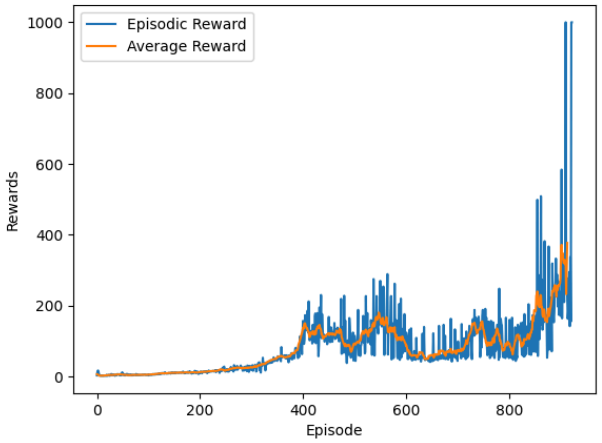

Fig. 2. Episodic and Average rewards

As we can see the rewards for the episodes, reaches 1000 after approximately 1000 epochs, that would indicate that the stabilizing policy was reached as the max time step was 1000 and the Inverted Pendulum environment gives +1 reward for every timestamp successfully balanced.

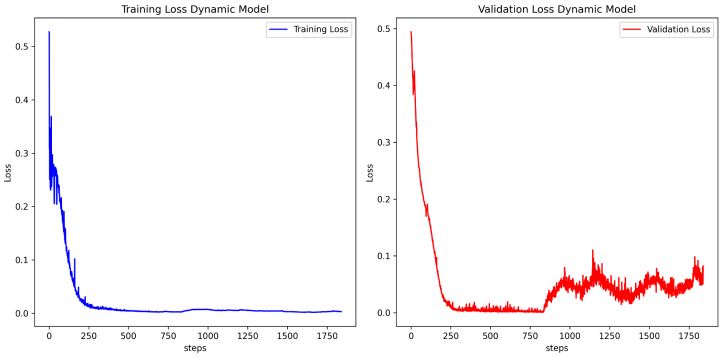

Fig. 3. Train and Validation plots for the Dynamics Model

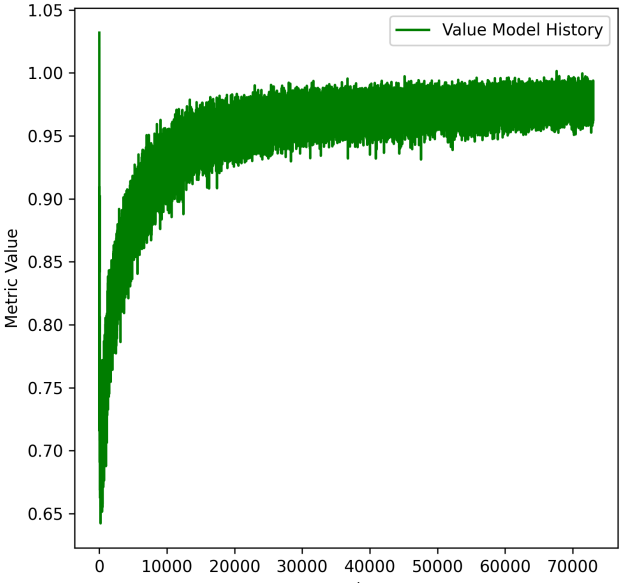

Fig. 4. Value Model Loss

Although the value model loss seems to be increasing, the model still seem to be learning a good policy, this might mean that the gradients of the value model are still sufficient for the learning of the policy and the agent seems to be exploration well.

CONCLUSION

This project managed to highlight the effectiveness of MBRL in policy learning. Another interesting thing about MBRL is that the model learnt can be used for other policies also.

Now to the things that need improvement:

1) Optimizer: MPPI and iCEM are “better” than CEM, they can be employed
2) Learning a reward Function: The Experience Replay buffer is used our method, but this is a off-policy utility, hence there might be some inconsistencies, even though the original paper says it doesn’t affect

the output. We can get rid of it by learning a reward function

3) Quadratic Neural Networks for the dynamics model

References

[1] Antonyshyn, L., Givigi, S. Deep Model-Based Reinforcement Learning for Predictive Control of Robotic Systems with Dense and Sparse Rewards. J Intell Robot Syst 110, 100 (2024). https://doi.org/10.1007/s10846-024-02118-y

[2] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., et al. (2013). *Playing Atari with Deep Reinforcement Learning*. arXiv preprint arXiv:1312.5602.

[3] Haarnoja, T., Zhou, A., Abbeel, P., & Levine, S. (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv.* https://doi.org/10.48550/arXiv.1801.01290

[4] Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., & Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. *International Conference on Machine Learning*. https://doi.org/10.48550/arXiv.1602.01783

[5] Hafner, D., Lillicrap, T., Ba, J., & Norouzi, M. (2020). Dream to control: Learning behaviors by latent imagination. *International Conference on Learning Representations.* https://doi.org/10.48550/arXiv.1912.01603

[6] Zhou, W., Chen, Z., Ramsundar, B., Liu, L., Ma, T., & Ermon, S. (2020). Model-based reinforcement learning via meta-policy optimization. *Advances in Neural Information Processing Systems.* https://doi.org/10.48550/arXiv.1906.08253

[7] Hafner, D., Lillicrap, T., Norouzi, M., & Ba, J. (2021). Mastering Atari with discrete world models. *International Conference on Learning Representations*. https://doi.org/10.48550/arXiv.2010.02193

[8] Tian, Y., Arora, S., Chen, Y., & Finn, C. (2018). Model predictive control with neural network dynamics. *International Conference on Learning Representations*. https://doi.org/10.48550/arXiv.1708.04102

[9] Pinneri, C., Sawant, S., Blaes, S., Achterhold, J., Stueckler, J., Rolinek, M., & Martius, G. (2020). *Sample-efficient cross-entropy method for real-time planning*. In *Proceedings of the Conference on Robot Learning 2020*. Retrieved from https://corlconf.github.io/corl2020/paper_217/

[10] Calandra, R., Lesort, T., et al. (2017). Model Predictive Path Integral Control. *NeurIPS 2017*. https://doi.org/10.48550/arXiv.1703.08420

[11] Hafner, D., Lillicrap, T., Ba, J., et al. (2020). PLaNet: Learning to Plan with Model-based Reinforcement Learning. *ICML 2020*. https://doi.org/10.48550/arXiv.2009.12644

[12] Yang, N., Ruan, S., Han, L., Liu, H., Guo, L., & Xiang, C. (2023). Reinforcement learning-based real-time intelligent energy management for hybrid electric vehicles in a model predictive control framework. *Journal Name*, *Volume*(Issue), page range. https://doi.org/xxx