Evaluating Zeroth and First-Order MPC Methods with a World Model

Sai Prasanna Diego Fernandez Clausen Akshay L Chandra
Computer Science, University of Freiburg
Freiburg im Breisgau, Germany
{*ramans, fernandi, lagandua*}*@cs.uni-freiburg.de*

***Abstract*—It is readily accepted by the robotics community that agents trained with reinforcement learning (RL) methods struggle to embody explicit environmental assumptions and obey constraints, while model predictive control (MPC) methods thrive in such constrained situations. MPC methods assume an existing accurate model of the environment and focus on planning, while RL methods rely on agent interactions with the environment to produce useful policies. In this project, we flip the table and study MPC methods' performances when dealing with a learnt model of the forward dynamics and the reward function, i.e. a world model. Specifically, a neural network-based world model is initially uncertain and inaccurate but evolves through a careful interplay between policy and model learning. To that end, we compare MPC using the Cross-Entropy Method (CEM), its modern variants, and a learnt actor-critic approach to obtain the policy. Through our experiments, we found that CEM can perform reasonably well when given an uncertain and evolving world model. While we did find competing evidence when evaluating first-order CEM variants, we suspect it is entirely due to our hyperparameter choices and that this contrasting evidence does not represent the true potential of these methods. The actor-critic approach performs the best.**

***Index Terms*—reinforcement learning, model predictive control, dreamer, planet, cross-entropy method**

I. Introduction

Researchers have long strived to solve long-horizon problems, especially in robotics. In this setting, the agent or controller must consider not only the immediate reward but also the possible future transitions into differing states since actions taken at early stages could substantially impact the long-term future. Model Predictive Control (MPC) tries to solve this problem. The general idea of MPC is to predict the future behaviour of the controlled system over a finite time horizon, given a set of constraints, and compute an optimal control input that minimizes a specific cost function. MPC is widely applied to real-world dynamics systems, chemical plants, oil refineries and safe robotics, which work in complex scenarios close to humans. [1] MPC is often used when an accurate model is provided due to its straightforward formulation, explicit handling of constraints and well-understood tuning parameters. Another long-horizon method is Reinforcement Learning (RL). This is a set of algorithmic techniques to solve long-horizon problems. In these settings, you have an agent and an environment. The agent acts and receives rewards from the environment. The goal of the agent is to maximize the cumulative sum of rewards. This is usually defined as a Markov decision process, a 4-tuple: $\{S, A, p, R\}$ where S is the states, A is the actions, p is the model or transition dynamics and R is a finite set of scalar rewards.

We can then define the expected reward:

$$r(s, a) = E[R_{t+1}|S_t = s, A_t = a] \tag{1}$$

The agent learns a policy π(s) that maps states to actions The state S contains is all the information the agent needs to know how to act in the environment. This is referred to as a Markov property. Which states that the future is independent of the past, given the present.

Unlike traditional control, RL generally does not rely on an *a priori* dynamic model and can be directly applied to uncertain dynamics. However, RL doesn't naturally allow the agents to model and obey explicit assumptions and constraints on the environment, limiting their applicability to safe control. While this limitation of RL is well-understood by the community, through this study, we want to understand how some MPC methods can perform when given uncertain dynamics models that only grow certain over time.

II. Related Work

A. Model-Based Reinforcement Learning

Broadly speaking, there are two types of RL algorithms: Model-free and Model-based. In model-free RL, we do not have the transition dynamics of the environment *a priori*, so the agent can only learn through direct interaction with the environment, making these methods model-independent. However, they tend to be sample inefficient, and it would usually require a lot of agent exploration in the environment to learn meaningful policies. On the other hand, model-based RL methods assume a dynamics model of the environment. This makes planning possible and makes the system considerably sample-efficient. MPC and Model-based RL are quite similar in many ways. The difference is that model-based RL can explode in compute as the state dimension increases since RL tries to learn the system's dynamics on the go. At the same time, MPC often assumes access to an existing accurate model. Nonetheless, both solve long-horizon problems using a transition model of the environment.

In model-based RL we can potentially learn a forward dynamics model directly on the observations. But then the model becomes costly to use when the observation dimension increases. For example, let's define our observation as a small

Initialize $\mu \in \mathbb{R}^d, \sigma \in \mathbb{R}^d$
for iteration $= 1, 2, \ldots$ **do**
  Collect n samples of $\theta_i \sim N(\mu, \text{diag}(\sigma))$
  Perform a noisy evaluation $R_i \sim \theta_i$
  Select the top ρ% of samples (e.g. $p = 20$), which we'll call the **elite set**
  Fit a Gaussian distribution, with diagonal covariance, to the elite set, obtaining a new μ, σ.
end for
Return the final μ.

Fig. 1: CEM Algorithm. Credits: John Schulman [7]

$64 \times 64 \times 3$ image. This has 12,288 different values. Generating these vectors will take a deconvolutional network which is computationally expensive. A better alternative is to learn the dynamics model on compressed state representations. Recent works like PlaNet and Dreamer have adopted this approach.

PlaNet [4]: avoids the problem of video prediction, where we learn to predict high-dimensional observations given a sequence of previous observations and actions. Instead, it learns to encode the observations to latent states and learn reward and forward dynamics predictors as a function of the latent states. The world model comprises of a recurrent state space model (RSSM), a forward dynamics model and a reward model. The RSSM consists of a Convolutional Neural network (CNN) and a Recurrent neural network. The CNN encodes the image at a given timestep into an observation vector. The RNN is applied on all the CNN-encoded observation vectors to derive the latent state. These latent state representations are then used to train the forward dynamics model and the reward model. The forward dynamics model and the reward model predict the next state and reward for a given latent state and action pair. The RSSM has the potential to compress the observations into latent factors such as object positions and velocities. In PlaNet, the policy is derived by optimizing the MPC problem on the learnt dynamics and reward model.

Dreamer [3]: leverages the PlaNet world model for generating artificial trajectories in the latent state space. These latent trajectories are used to train an Actor-Critic agent. This is sample efficient as the agent avoids costly interaction with the real environment. It can rather learn from past learned experiences. The actor-critic network has the benefit of avoiding slow online planning during inference. Since the learnt world model is a differentiable neural network, the parameters of the actor-network can be trained using the gradients of the sum of rewards directly. Bootstrapping with a value network allows for obtaining better reward estimates outside the imagined trajectory length. This approach avoids high variance policy gradient estimates for the actor's parameters. We refer [4] and [3] for more details on PlaNet and Dreamer, respectively.

B. Cross-Entropy Method with Gradient Descent Planner

The cross-entropy method (CEM) was introduced by Rubinstein and Davidson [5] in the 1990s as a stochastic, derivative-free, global optimization technique. This was proposed as an importance sampling procedure for the calculation of low-probability events that leverage the cross-entropy measure. CEM was later utilized as a trajectory optimization technique in model-based RL. This method uses an evolution strategy that minimizes a cost function $f(x)$ by finding the best candidate x. CEM samples from a Gaussian with mean μ and diagonal covariance matrix diag(σ^2), where μ, $\sigma \in \mathrm{R}^{\mathrm{n}}$. After sampling, the candidates are evaluated with the cost function $f(x)$ and only the best K samples are used to fit the Gaussian distribution in the next iteration. Figure 1 concisely shows CEM as an algorithm.

Another common planning technique is Gradient Descent (GD) Planner. GD naively uses gradient descent on the actions in the direction that increases the rewards in the planned horizon. First-order methods like GD are often used effectively on high-dimensional optimization problems [4]. In model-based RL, if we have a differentiable dynamics model and objective function, we can directly optimize the action sequences following the gradient direction of the reward function with respect to actions. It can be more sample efficient during planning than the zeroth-order approaches. However, they tend to get stuck in poor local minima, especially if the surface of the learnt forward dynamics function is highly non-linear and non-smooth, which is the case with deep neural networks.

Zeroth-order optimizers like CEM have the benefit of not requiring to compute the derivatives, thus being able to work with non-differentiable functions. However, as the dimensionality of the input grows, CEM becomes too expensive to use. On the other hand, first-order methods can work well in high-dimensionality optimization problems. Cross-Entropy Method with Gradient Descent (CEM-GD) [6] proposes to combine the zeroth-order CEM and the first-order GD planner to enjoy the benefits of both simultaneously.

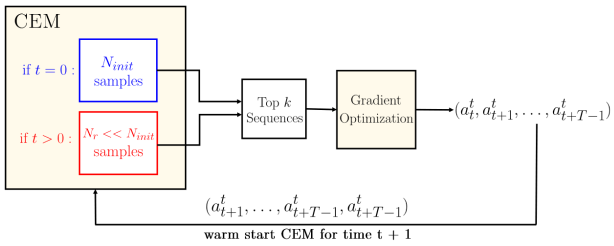

Fig. 2: Overview of CEM. Image is directly taken from [6].

As shown in Figure 2, CEM-GD first uses samples generated randomly by CEM. It then selects the top k sequences of actions and performs gradient optimization. Finally, the highest reward-achieving actions are selected as a warm start for the next iteration. CEM-GD uses PETS as its end-to-end reinforcement learning algorithm. This planning method performs better than CEM, with 100x fewer samples per time step. Furthermore, as the dimensionality of the planning problem increases, it maintains desirable performance with a constant small number of samples. In CEM-GD [6], the authors use PETS as their

III. Our Experiments

We study how MPC methods and one RL method perform when given a world model learnt concurrently with the said methods. Specifically, we evaluate Dreamer (Actor-Critic + RSSM), PlaNet (CEM + RSSM), Gradient Descent Planner (GD + RSSM) and Cross-Entropy with Gradient Descent Planner (CEM-GD + RSSM) on Cartpole Swingup task. For accuracy and consistency, we reuse the main algorithm code from the official code provided by the authors of CEM-GD [1]. This potentially guarantees that all results we report are contingent solely on our design and hyperparameter choices.

For all tasks, we set the planning horizon (imagination horizon in the case of Dreamer) to 15 steps, but all other hyperparameters are directly taken from the original CEM-GD implementation and can be found in Section 3 of [6]. We ran each experiment twice since a single run of CEM and its first-order variants took approximately 24 hours on a single RTX 2080 GPU to reach up to 200K steps. To know about all the hyperparameters and model architectures used for training Dreamer's Actor-Critic model, refer to the *config.yaml* file in our code repository[2].

IV. Results and Discussion

Figure 3 shows the results of our experiments. As shown, Dreamer's performance keeps increasing over time. This is expected as the original Dreamer's actor-critic implementation was proven to work with an RSSM-based world model. In the case of MPC methods, CEM reaches a respectable average reward range of 350-400 pretty early into the training but is stuck there throughout the end of training in both the seeds, showing signs of convergence. Perhaps longer experiments would add more evidence to this. This shows that a zeroth-order MPC method, CEM, can deal reasonably well when given an uncertain and evolving latent dynamics model.

On the other hand, GD and CEM-GD do not seem to learn or do anything meaningful. They converge to 0 reward very early into the training. We suspect this behaviour is due to one of our design choices and might not reflect the true potential of the methods [6]. Despite tuning some interpretable hyperparameters like the planning horizon, the learning rate, etc., we couldn't find the problem with these methods. While we are highly motivated to seek an explanation for this result, we are severely limited by computational resources.

V. Conclusion

We evaluated three MPC approaches and one RL (actor-critic) approach for optimal control using a world model. Among them, the zeroth order CEM approach achieves good performance initially but stops improving quickly. The CEM-GD and GD approaches incorporating first-order gradients do not obtain good returns. Learning an actor-critic network using the experience generated by the world model has the best performance. This could be because of better reward estimates via bootstrapping with the value network. The world model on the image observations could lead to a highly non-convex optimization problem, which could explain the poor performance of gradient-based approaches (GD and CEM-GD). More experiments with different environments and hyperparameter settings might be necessary to strengthen this hypothesis.

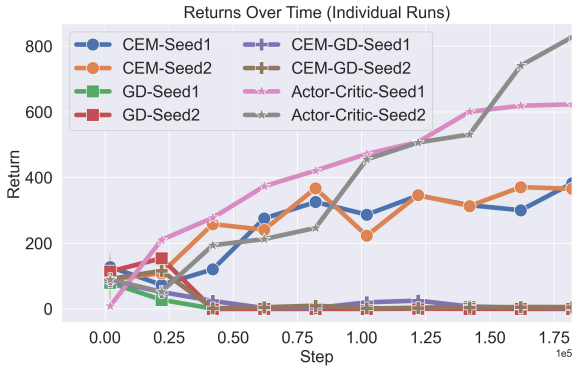

Fig. 3: Our Main Results.

References

[1] L. Brunke, M. Greeff, A. W. Hall, Z. Yuan, S. Zhou, J. Panerati, and A. P. Schoellig, "Safe learning in robotics: From learning-based control to safe reinforcement learning," Annual Review of Control, Robotics, and Autonomous Systems, vol. 5, no. 1, pp. 411–444, 2022.

[2] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In Advances in Neural Information Processing Systems 31, pages 4759–4770, 2018. URL https://proceedings.neurips.cc/paper/2018/hash/3de568f8597b94bda53149c7d7f5958c-Abstract.html.

[3] D Hafner, T Lillicrap, J Ba, M Norouzi. Dream to Control: Learning Behaviors by Latent Imagination. ArXiv Preprint ArXiv:1912.01603, 2019.

[4] Danijar Hafner, Timothy P. Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In Proceedings of the 36th International Conference on Machine Learning, pages 2555–2565, 2019. URL http://proceedings.mlr.press/v97/hafner19a.html.

[5] R. Rubinstein and W. Davidson. The cross-entropy method for combinatorial and continuous optimization. Methodology and Computing in Applied Probability, 1999.

[6] K. Huang, S, Lale, U. Rosolia, Y Shi, A. Anandkumar. CEM-GD: Cross-Entropy Method with Gradient Descent Planner for Model-Based reinforcement Learning.ArXiv Preprint ArXiv:2112.07746, 2021.

[7] J. Schulman. Deep Reinforcement Learning. MLSS May 2016, Cadiz. https://www.youtube.com/watch?v=aUrX-rP_ss4.

[1] https://github.com/KevinHuang8/CEM-GD

[2] https://github.com/sai-prasanna/mpc_and_rl/