Model-Based Policy Optimization for Legged Locomotion

Javier Garcia-Barcos
I3A,
Universidad de Zaragoza
Zaragoza, Spain
jgbarcos@unizar.es

Ruben Martinez-Cantin
I3A,
Universidad de Zaragoza
Zaragoza, Spain
rmcantin@unizar.es

Luis Montesano
I3A,
Universidad de Zaragoza
Zaragoza, Spain
montesano@unizar.es

***Abstract*—In this paper, we explore model-based reinforcement learning (MBRL) for legged locomotion in robotics, employing state-of-the-art techniques to evaluate the performance and stability of the learning process under different models and learning strategies. Our algorithm utilizes policy search methods, specifically leveraging deep ensembles to construct a dynamic model and integrating the soft actor-critic (SAC) framework for policy learning. Deep ensembles are traditionally employed to address model uncertainties and improve the robustness of the learning process, however, this relationship is barely evaluated in practice. We conduct a comprehensive comparison of different architectural designs and ensemble strategies within our dynamic model on several simulated legged robot platforms. Our evaluation includes benchmarking the performance of different strategies of state-of-the-art algorithms, providing a detailed analysis of learning speed, stability, and overall effectiveness. The results highlight the importance of certain design criteria which are typically assumed as granted. This study provides valuable insights into the application of deep ensembles and SAC in MBRL for legged locomotion and complex robotic control systems.**

***Index Terms*—model-based rl, exploration, bayesian nn**

I. Introduction

Legged locomotion is fundamental for robots to accomplish tasks in human environments as wheeled locomotion is limited in stairs, ladders, certain terrains, etc. Legged locomotion spans from one-leg hoppers to spider-like hexapods, including human-like robots or horse-like quadrupeds [1]. However, legged locomotion control is much more complex and convoluted than other types of locomotion. Even static balancing can be problematic for certain robots.

Reinforcement learning (RL) in general, and deep reinforcement learning in particular, are promising technologies to deal with the hard control problems of legged locomotion, even in rough or unexpected terrain with robots from all kinds of shapes and number of legs [2]–[4]. By leveraging the predictive properties of neural networks, previous works have some outstanding results in learning complex walking patterns using reinforcement learning [5], [6]. However, most reinforcement learning algorithms use a model-free strategy, enabling great generalizability capabilities. That is, the same algorithm and model architectures can be applied in many contexts and configurations, and remove many assumptions about the dynamics of the robot or the environment [7], [8].

This work was supported by DGA T45_23R and MCIN/AEI/ERDF/NextGenerationEU/PRTR projects PID2021-125209OB-I00, TED2021-129410B-I00 and TED2021-131150BI00.

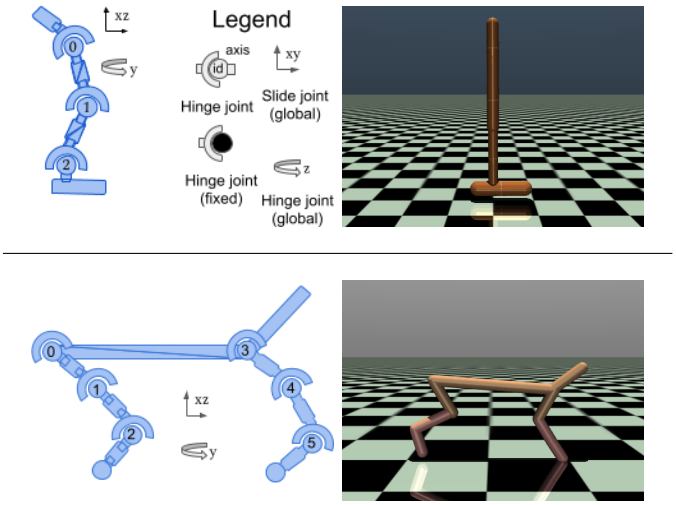


Fig. 1: Locomotion environments used in this work. The top plot corresponds to a planar **hopper** robot, which has to learn to walk forward, without falling, as fast as possible. It consists of 4 links and 3 joint actuators: leg, knee and foot. The bottom part of the figure is a planar cat-like robot called **halfcheetah**. It consists of 9 links and 8 joints, including the two paws. In this case, we have replaced the standard task of running, for a *flipping* task where the robot has to learn to do a backflip in place. Diagram figures from Gymnasium [9].

In practice, many robotics applications are not compatible with large datasets. Collecting data is expensive in many ways: energy, time, wear, money, resources, etc. Large robotic datasets are very limited and difficult to scale. Alternatively, there are methods to adapt simulator data for reinforcement learning and close the simulation to a real gap. However, this still requires a complex simulation with an accurate physics engine, precise robot and environment models, realistic sensors, etc. Similarly, this is again expensive computationally and in terms of resources. Therefore, there is a necessity for data-efficient reinforcement learning algorithms.

Model-based reinforcement learning (MBRL) algorithm is

a branch of RL algorithms that learns a dynamic model of the robot and the environment to predict future states and rewards [10]–[12]. Unlike model-free approaches, which rely solely on interactions with the environment to learn a policy, MBRL uses the learned model to simulate these interactions. This allows for more sample-efficient learning, as the agent can evaluate multiple possible actions and trajectories using the model before executing them in the real environment. We can assume that MBRL generated *imaginary interactions*. Also, because MBRL uses computational models, such as Gaussian processes or neural networks, they are much faster and more efficient to generate than physics engines and complex simulators. Thus, MBRL is particularly useful for our scenario, where data collection is expensive or time-consuming, offering the potential for faster and more efficient policy optimization.

Our contributions are an analysis of MBRL algorithms and, in particular, of model-based policy optimization (MBPO) using deep ensembles as a dynamic model, which is a method that is capable of solving different legged locomotion scenarios such as those in Figure 1. Based on the literature, the size of the neural network, and more importantly, the size of the deep ensemble model should translate into an increased quality of the model uncertainty as described in Section II-A. In this paper, we analyze the performance of the learning algorithms in terms of model size and uncertainty quality. Furthermore, we found that many implementations use a method that has not been reported in the literature that we call *elites selection*. It reduces the number of ensembles used for prediction, which can help improve the predictive throughput, but seems counter-intuitive in theory with proper uncertainty calibration. We analyze the influence of the elite's trick in the performance of the results.

II. Model-based Reinforcement Learning

We can frame the problem of legged locomotion in RL under the formal mathematical framework of the Markov Decision Process (MDP). It assumes that state transitions satisfy the Markov property, that is future states of the process only depend on the present state, or mathematically $p(s_{t+1}|s_t) = p(s_{t+1}|s_1, \cdots, s_t)$. An MDP is defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, r, \gamma, p_0)$. $\mathcal{A}$ is the action space, $\mathcal{S}$ is the state space, $\gamma \in (0, 1)$ is the discount factor. The possible transition dynamics of the system are defined by the function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to p(\mathcal{S})$, initial state is sampled from distribution $p_0(s)$ and a reward function $r(s, a) : \mathcal{S} \times \mathcal{A} \to \mathcal{R}$. In RL, the goal is to find the optimal policy π^* that maximizes the expected sum of rewards:

$$\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{t=0} \gamma^t r(s_t, a_t)\right] \tag{1}$$

In practice, the dynamics of the transition function $\mathcal{T} = p(s'|s, a)$ and, in some cases, the reward function $r(s, a)$ can be assumed to be unknown. As commented before, model-free methods try to solve Equation (1) directly without explicitly modeling $\mathcal{T}$. In contrast, model-based reinforcement learning, on top of learning the optimal policy π^*, also learns a probabilistic model of the dynamics of the system $\tilde{p}(s'|s, a) \approx \mathcal{T}$. This model is leveraged to help with learning the optimal agent policy π^* more efficiently in terms of interactions with the real environment. To simplify the notation, for the remainder of the paper, we use $p(s'|s, a)$ as the approximate distribution resulting from the dynamic model.

MBRL methods typically follow a general recipe: 1) an agent policy π is used to act on the real environment and collect real data D_{env}, 2) then this data is used to learn the dynamics model $p(s'|s, a)$, 3) which then is used to generate *model rollouts* following a policy π and collect model data D_{model}, and, 4) finally, the data is used to learn the agent policy π. Different methods differ in how each of the components are used, such as: how the data is used (both D_{env} and D_{model}), the amount of model and policy updates, training frequency, model rollout generations.

a) ***Dynamics model****:* The dynamics model is a supervised learning problem that uses data acquired from real environment interactions $D_{\text{env}} = [d_1, \ldots, d_N]$ where $d_i = (s_i, a_i, s'_i, r_i)$ are one-step transition data. Note that classical dynamics models only require data in the form $d_i = (s_i, a_i, s'_i)$ however, we consider the general case where the reward function is unknown. In this case, you can use the same model with multiple heads or outputs to predict both the next state s' and current reward $r(s, a)$. This model can be non-parametric, such as Gaussian processes [10] or parametric, such as deep neural networks [11], [12]. Given that we are dealing with a probabilistic model $p(s'|s, a)$, the output is a distribution, such as the parameters of a Gaussian distribution (μ, σ). In the case of the Gaussian process, this distribution comes directly from the prediction. In the case of deep neural networks, we also need multiple heads to predict both the mean and variance of the distribution. Then, we can say that the neural network is learning both the prediction $\mu(s, a)$ and a *heteroscedastic* noise $\sigma(s, a)$. Model parameters θ are learned by optimizing loss functions such as *maximum likelihood estimation (MLE)* or *maximum a posteriori (MAP)* of the model weights:

$$\begin{aligned} \theta_{MLE} &= \arg\max_{\theta} \log p(\mathcal{D}|\theta) \\ \theta_{MAP} &= \arg\max_{\theta} \log p(\mathcal{D}|\theta) p(\theta) \end{aligned} \tag{2}$$

b) ***Model usage****:* The learned dynamics model $p(s'|s, a)$ can be used to perform *model rollouts* and acquire *model data*, reducing the need for real world interactions. This can be seen as solving an approximate MDP where the transition function $\mathcal{T}$ is replaced by its approximation $p(s'|s, a)$. Alternatively, it can also be understood as if the learned model is used as a computationally cheap simulator which also provides probability distributions of each prediction during a transition.

A. Bayesian model-based RL

According to the literature, one crucial aspect of learning the dynamics model is capturing both the *aleatoric uncertainty* and *epistemic uncertainty* [11]–[15]. Aleatoric uncertainty is uncertainty associated with the data or data noise. In many

models, a constant noise level or *homoscedastic* noise is assumed. Alternatively, we can use a multi-headed neural network to predict a noise level for each input or *heteroscedastic* noise. The *epistemic uncertainty* or model uncertainty, corresponds to the uncertainty due to the lack of knowledge or model mismatching. The Bayesian approach to epistemic uncertainty corresponds to replace Equation (2) by finding the whole posterior distribution $p(\theta|D)$ using Bayes rule:

$$p(\theta|D) = \frac{p(\mathcal{D}|\theta)p(\theta)}{p(D)} = \frac{p(\mathcal{D}|\theta)p(\theta)}{\int p(\mathcal{D}|\theta)p(\theta)d\theta}. \tag{3}$$

Unfortunately, the integral in the denominator is intractable for neural networks or nonlinear problems. Therefore, approximate Bayesian inference is needed. Furthermore, in order to use the posterior distribution in our predictions, we also need to compute the predictive posterior distribution by marginalizing the weights. That is:

$$p(s'|s,a) = \int_\theta p(s'|s,a,\theta)p(\theta|D)d\theta \tag{4}$$

Again, this integral is intractable and we also need to rely on approximate Bayesian methods to estimate it. Some methods use a Gaussian approximation of the posterior and predictive posterior distribution using either the Laplace approximation [14], [16] or variational inference [17]. However, the most popular method is to use a sample representation of the network weights $\{\theta_i\}_{i=1}^M$, as in Monte Carlo where $\theta_i \sim p(\theta|D)$ is a sample of the posterior distribution, and replace the integrals with the corresponding sample approximation, that is we have M different networks each one with different weights θ_i and then we can compute:

$$p(s'|s,a) \approx \sum_{i=1}^{M} p(s'|s,a,\theta_i) \tag{5}$$

where $p(s'|s,a,\theta_i)$ is the prediction of the i-th network. For computing a sample representation, there are methods based on Monte Carlo Markov chain methods, with specific proposals but which are difficult to work in practice. For scalable approaches, the most popular methods are Monte Carlo dropout [16], [18] and deep ensembles.

a) ***Deep Ensembles****:* Deep ensembles [19] is a method to sample neural networks, which is designed for its simplicity and scalability, being able to be applied to any architecture and learning method. It is based on the idea of *sample-then-optimize*, where M different networks are trained independently from random initial weights. The idea is that by optimizing Equation (2) using gradient descent from different random initializations will end up in each network optimizing a different local optimum, capturing the multimodality of the posterior distribution. Although this method of sampling does not properly represent the posterior distribution, given the dimensionality of the weight space in deep neural networks and the reduced number of samples required (remember that each sample is a different network) the performance is superior to other Bayesian methods for neural networks [19], [20] and it is the gold standard in reinforcement learning literature [11], [12]. In the RL literature [11], [12], some authors propose to combine ensembles with bootstrapping, in what is called *probabilistic ensembles* where each network is trained with a different random subset of the dataset, to give more variability to the samples. However, in other fields, such as computer vision, it has been shown to reduce performance [19].

b) ***Elite selection of ensembles****:* We found that some authors use a subset of the deep ensembles in their official implementations of model-based RL methods [11], [12], [21]. The idea is to train a larger set of deep ensembles M and use inference only with a subset $B < M$ of the ensembles with the best final loss. To the authors' knowledge, this has never been reported officially. Thus, we have decided to call it *elites selections*. More concretely:

$$\{\theta_j^{EL}\}_{j=1}^{B} = \arg\max_\theta \log p(\mathcal{D}|\theta_i^{DE})p(\theta_i^{DE}) \quad \forall i = 1 \dots M \tag{6}$$

where θ^{EL} are the elite samples and θ^{DE} are the original samples obtained with deep ensembles. This seems counter-intuitive as this reduces the calibration of the epistemic uncertainty, resulting in an overconfident estimation of the uncertainty. On one hand, this reduces the computational cost and memory footprint during evaluation, as it requires less number of network passes during prediction and average performance should be higher. On the other hand, the largest computation cost of deep ensembles is training, which remains constant. Thus, we will evaluate the performance of this method in our experiments.

III. Model-Based Policy Optimization (MBPO)

In this paper, we will focus on model-based policy optimization (MBPO) [12]. The goal remains the same as other RL methods, finding the optimal policy π^* as defined in equation (1). It uses a policy optimization approach, that is, a policy is optimized by performing gradient updates but relies on

Algorithm 1 Model-Based Policy Optimization

Require: Agent policy π, predictive model p, environment dataset D_{env}, model dataset D_{model}

for N epochs **do**
  Learn dynamics $p(s'|s,a)$ from D_{env} via MLE
  for E steps **do**
    Take action in the environment with π
    Add observed transition to D_{env}
    for N model rollouts **do**
      Sample s_t uniformly from D_{env}
      Perform k-step model rollout from s_t with π
      Add model rollouts to D_{model}
    end for
    for G gradient updates **do**
      Update agent policy π using D_{model}
    end for
  end for
end for

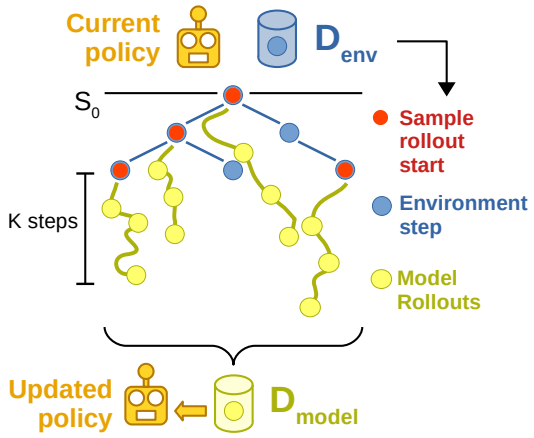


Fig. 2: Overview of *branching rollouts* procedure in MBPO. From a current policy (orange) and previously acquired data in the real environment D_{env} (blue), a set of initial states are sampled (red) from D_{env} uniformly which are then used as starting states for model rollouts (yellow). Model rollout uses the current dynamics model to *simulate* imaginary transitions up to k-steps. All the data acquired from model rollouts D_{model} is used to update the policy.

learning a dynamics model and model rollouts to improve the data efficiency. Algorithm 1 shows an overview of the method.

Similarly to other model-free and model-based RL methods, to populate the initial data in D_{env}, MBPO uses a randomized policy to initialize D_{env}. After that, the current policy π is used to acquire new environment data.

Then, following the MBRL general recipe, a dynamics model is learned from previously acquired environment data D_{env}. This model is used to perform model rollouts following the current policy π and generate model data D_{model}. Since model dynamics don't change drastically between environment steps and the training is costly as shown in Section II-A, training frequency is episodic.

For model rollouts, it uses a *branching rollout* strategy. Figure 2 shows a diagram of the approach which, instead of doing rollouts from the real environment's initial or current state up until the full task horizon, it favors smaller k-step rollouts branching from states uniformly sampled from D_{env}.

Finally, taking advantage of the model-based approach, we use the generated model data D_{model} to update the current policy π. Since the *branching rollouts* have shorter horizons, for updating the policy, MBPO relies on smaller but more frequent updates of the policy, updated once per real environment step.

IV. Experiments

The experiments are performed in a simulated environment with different legged robots from the Gymnasium benchmarks [9] as shown in Figure 1, using Mujoco [22] as the physics simulator. We have used the Hopper and HalfCheetah robots. However, for the HalfCheetah robot, we have replaced the default running task with a flipping task where the robot has to perform a backflip to provide a broader variability of scenarios.

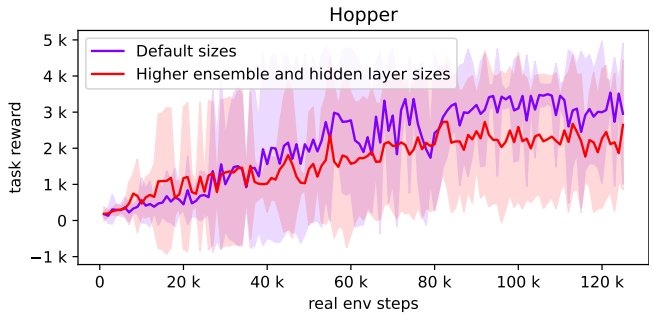


Fig. 3: Execution of MBPO in the Hopper environment comparing different model sizes. It shows slightly worse performance using the bigger model.

All the environments have continuous state and action spaces. More detail:

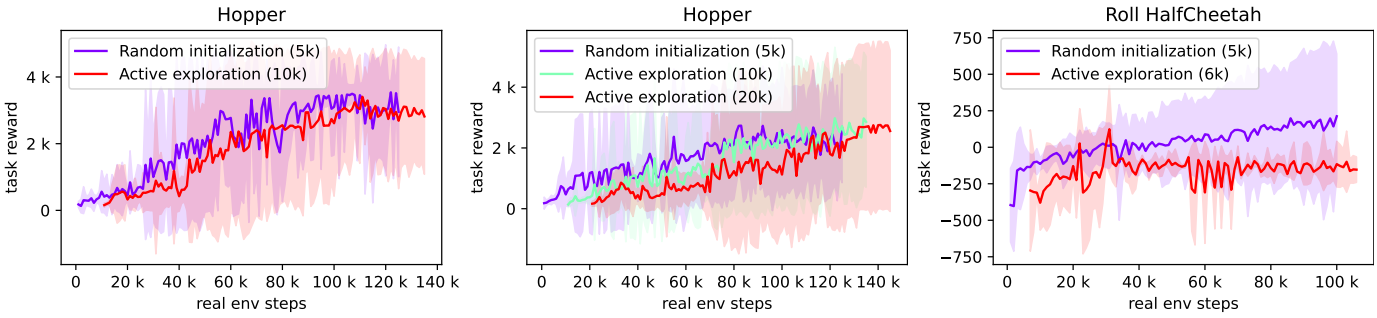
a) Hopper: It is a simulated one-legged robot that moves in a 2D space [23]. It consists of 3 joints: thigh, leg and foot; and a torque can be applied to each of them. Figure 1 shows an image of the robot. The reward is the distance traveled in the positive x-axis. For this environment, $\mathcal{A} \in \mathbb{R}^3$ and $\mathcal{S} \in \mathbb{R}^{11}$.

b) Roll HalfCheetah: Simulates a two-legged robot that moves in a 2D space [24]. Each leg consists of 3 joints in which torque can be applied. As opposed to the classic HalfCheetah environment reward [9] which rewards distance traveled on the positive x-axis, here we perform the flip task which rewards rotation of the root in a counter-clockwise direction. For this environment, $\mathcal{A} \in \mathbb{R}^6$ and $\mathcal{S} \in \mathbb{R}^{17}$.

Our implementation of MBPO is based on the MBRL-Lib library [21]. The performance will be measured in terms of task reward at test time, executing the current policy for 100 episodes and reporting average values. Progress will be measured in the number of real environment interactions. In the experiments, we analyze the influence of the network size and ensemble size. We also consider initializing the dynamic model using actively explored data of the environment [15], which in theory, provides a better calibration of the model by using more informative data. Finally, we evaluate the influence of the elites selection discussed in Section II-A.

A. Influence of model size

In previous works [12], this type of scenario (Hopper) is solved with a relatively small multi-layer perceptron (MLP) for the dynamic model (4 layers of 200 hidden neurons per layer) and especially for the deep ensembles (7 networks, choosing 5 elites). In our first experiment, we tried to increase the model size (4 layers and 512 hidden neurons per layer) and, especially, the size of NN for the deep ensembles (16 networks, choosing 5 elites). As seen in Figure 3, contrary to intuition, the performance is quite similar with bigger models having a slightly reduced performance. Furthermore, the model seems to stagnate, meaning that including more data would not improve the performance of the larger model. However, we hypothesize that a larger model might benefit from more informative data. Thus, in the next section, we tried initializing



(a) (b) (c)

Fig. 4: Comparison of different initialization procedures with MBPO, comparing random policy against a model-based active exploration (MAX). Initialization steps are indicated in parenthesis and data is shifted to account for environment interactions used by MAX. Results from a) Hopper, b) Hopper with a bigger model and c) roll HalfCheetah. Results show no significant statistical difference except for the complex dynamics of HalfCheetah, in which MAX hinders performance.

the model with an information-based active exploration [15] to improve the initial data quality.

B. Initialization with Active Exploration

In Section III we explained that MBPO requires initialization data before it can begin updating the policy of a SAC agent and proceed with the MBPO procedure. For that, MBPO uses an untrained SAC, which in practice, behaves like a random policy. Random policy is a typical initialization in both model-free and model-based RL approaches as it provides enough coverage and it is trivial to implement. However, there are alternative approaches that use active learning to carefully select the most informative data points for this initialization phase. Model-based Active Exploration (MAX) [15] is an active exploration method for model-based RL that aims to efficiently explore the state and action spaces to learn the most informative dynamic model. It learns an exploratory policy by introducing an information gain utility based on the dynamic model uncertainty as a reward function.

Next, we replace the initial 5k steps of random initialization of MBPO with an execution of the MAX algorithm for 10k steps in the real environment. Note that all the initialization steps are used to learn the dynamic model, but not to learn the policy. Policy learning is based on model rollouts, being the same algorithm despite the initialization. Ideally, a more informative model should generate better rollouts which, at the same time, should improve the learning performance.

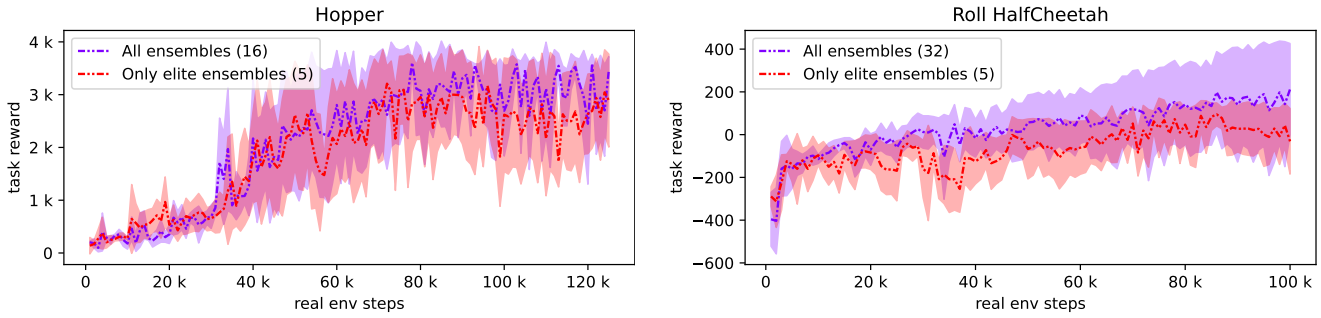
First, we evaluate the influence of informative data in the small model from the previous experiment. Figure 4a shows results in the Hopper environment and shows that it does not produce any meaningful improvement. If we take into account the 10k interactions of MAX before starting MBPO, it shows that data efficiency slightly deteriorates as this extra data barely produces any difference. Contrary to our initial hypothesis, the performance is also similar for the larger models and ensemble size, as shown in Figure 4b.

We also try the flip task in HalfCheetah, which has more complex dynamics, that we learn with a larger number of ensembles (32). In this scenario, previous results in the literature have shown that MAX produces more informative samples, which should help the performance [15]. However, figure 4c shows the results on roll HalfCheetah are still underperforming and it saturates before. On the other hand, the variability is much lower, which might be the result of more consistent or robust training. We conjecture that the exploratory data, being off-policy might generate biased or unlikely rollouts, which undermines the training performance overall, but at the same time improves the robustness.

C. Ensemble elites pruning

In Section II-A, the concept of *elites selection* was introduced as a pruning method where predictions with the dynamics model are performed only with the top N performing ensembles based on predictive performance. As mentioned before, this is counter-intuitive to the idea of proper uncertainty quantification. Theoretically, training multiple models allows the ensemble to quantify the epistemic uncertainty better but, by ignoring some of them for predictions, we are producing overconfident estimates of the epistemic uncertainty.

Here we study if there is a practical impact of introducing the pruning with elites for model predictions. For that, we performed experiments with Hopper and the roll HalfCheetah, which uses 16 and 32 ensembles respectively, compared to using only the top $N = 5$ ensembles. Results in Figure 5 show that average performance deteriorates when introducing elites selection and using the full ensemble for predictions. Results align with the hypothesis that better model uncertainty calibration translates into improved data efficiency. However, there are still practical reasons to use elites. To maximize the predictive throughput and achieve faster model rollouts, it is important to fit all ensembles into memory and elites selection reduces the number of ensembles required for predictions. Additionally, for simple models or with a more conservative number of elites, the negative impact of elites could be reduced.



(a) (b)

Fig. 5: Comparison of MBPO using all the ensembles and using the top $N = 5$ *elites selection* comparing: a) Hopper and b) roll HalfCheetah. Results show that using elites deteriorates performance, especially in complex tasks.

V. Conclusions

This study explores how design choices in model-based reinforcement learning (MBRL) affect data efficiency. MBRL relies on cheap model rollouts to reduce real world interactions to learn a policy. Specifically, we examine model-based policy optimization, a deep ensemble method capable of solving legged locomotion tasks efficiently.

We explore the influence of design decisions such as model complexity or improved data quality. Contrary to theoretical expectations, results showed that smaller neural network and ensemble sizes simplify learning the dynamics and improve performance. Similarly, we show that initialization with random policy is better than active exploration, since the improved data quality comes at the cost of a highly off-policy dataset that hinders policy learning. Finally, we studied elites selection, an undocumented implementation trick that improves predictive throughput with a subset of ensembles but, as we show, can deteriorate the performance with complex models due to poor uncertainty calibration. Although results may seem negative, such design decisions are common practices and working hypotheses in the literature and, as we see, it highlights the importance of empirical testing to validate theoretical assumptions and uncover potential limitations.

References

[1] Jawaad Bhatti, AR Plummer, Pejman Iravani, and Beichen Ding. A survey of dynamic robot legged locomotion. In *2015 International Conference on Fluid Power and Mechatronics (FPM)*, pages 770–775. IEEE, 2015.

[2] Roberto Calandra, André Seyfarth, Jan Peters, and Marc Peter Deisenroth. Bayesian optimization for learning gaits under uncertainty: An experimental comparison on a dynamic bipedal walker. *Annals of Mathematics and Artificial Intelligence*, 76:5–23, 2016.

[3] Antoine Cully, Jeff Clune, Danesh Tarapore, and Jean-Baptiste Mouret. Robots that can adapt like animals. *Nature*, 521(7553):503–507, 2015.

[4] J Peters. Reinforcement learning for humanoid robots-policy gradients and beyond. In *3rd IEEE International Conference on Humanoid Robotics, 2003*, 2003.

[5] Tuomas Haarnoja, Sehoon Ha, Aurick Zhou, Jie Tan, George Tucker, and Sergey Levine. Learning to walk via deep reinforcement learning. In *Robotics: Science and Systems XV*. Robotics: Science and Systems Foundation, 2019.

[6] Nicolas Heess, Dhruva Tb, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, SM Eslami, et al. Emergence of locomotion behaviours in rich environments. *preprint arXiv:1707.02286*, 2017.

[7] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT Press, 2018.

[8] Marc Peter Deisenroth, Gerhard Neumann, Jan Peters, et al. A survey on policy search for robotics. *Foundations and Trends® in Robotics*, 2(1–2):1–142, 2013.

[9] Mark Towers, Jordan K. Terry, Ariel Kwiatkowski, John U. Balis, Gianluca de Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Arjun KG, Markus Krimmel, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Andrew Tan Jin Shen, and Omar G. Younis. Gymnasium, March 2023.

[10] Marc Deisenroth and Carl E Rasmussen. PILCO: A model-based and data-efficient approach to policy search. In *ICML*, 2011.

[11] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *NeurIPS*, 31, 2018.

[12] Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to trust your model: Model-based policy optimization. *NeurIPS*, 32, 2019.

[13] Alex Kendall, Vijay Badrinarayanan, and Roberto Cipolla. Bayesian segnet: Model uncertainty in deep convolutional encoder-decoder architectures for scene understanding. *preprint arXiv:1511.02680*, 2015.

[14] David JC MacKay. Bayesian interpolation. *Neural computation*, 4(3):415–447, 1992.

[15] Pranav Shyam, Wojciech Jaśkowski, and Faustino Gomez. Model-based active exploration. In *ICML*, pages 5779–5788. PMLR, 2019.

[16] Carlos Plou, Ana C. Murillo, and Ruben Martinez-Cantin. Active exploration in bayesian model-based reinforcement learning for robot manipulation, 2024.

[17] David M Blei, Alp Kucukelbir, and Jon D McAuliffe. Variational inference: A review for statisticians. *Journal of the American statistical Association*, 112(518):859–877, 2017.

[18] Yarin Gal and Zoubin Ghahramani. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In *ICML*, pages 1050–1059. PMLR, 2016.

[19] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. *NeurIPS*, 30, 2017.

[20] Javier Rodríguez-Puigvert, Rubén Martínez-Cantín, and Javier Civera. Bayesian deep neural networks for supervised learning of single-view depth. *RA-L*, 7(2):2565–2572, 2022.

[21] Luis Pineda, Brandon Amos, Amy Zhang, Nathan O. Lambert, and Roberto Calandra. Mbrl-lib: A modular library for model-based reinforcement learning. *Arxiv*, 2021.

[22] Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. In *IEEE/RSJ IROS*, 2012.

[23] Tom Erez, Yuval Tassa, and Emanuel Todorov. Infinite-horizon model predictive control for periodic tasks with contacts. In *Robotics: Science and Systems*, 2011.

[24] Paweł Wawrzyński. A cat-like robot real-time learning to run. In Mikko Kolehmainen, Pekka Toivanen, and Bartlomiej Beliczynski, editors, *Adaptive and Natural Computing Algorithms*, pages 380–390. Springer Berlin Heidelberg, 2009.