POLICY OPTIMIZATION IN THE FACE OF UNCERTAINTY

Anonymous authors
Paper under double-blind review

ABSTRACT

Model-based reinforcement learning has the potential to be more sample efficient than model-free approaches. However, existing model-based methods are vulnerable to model bias, which leads to poor generalization and asymptotic performance compared to model-free counterparts. In this paper, we propose a novel policy optimization framework using an uncertainty-aware objective function to handle those issues. In this framework, the agent simultaneously learns an uncertainty-aware dynamics model and optimizes the policy according to these learned models. Under this framework, the objective function can represented end-to-end as a single computational graph, which allows seamless policy gradient computation via backpropagation through the models. In addition to being theoretically sound, our approach shows promising results on challenging continuous control benchmarks with competitive asymptotic performance and sample complexity compared to state-of-the-art baselines.

1 INTRODUCTION

Popular reinforcement learning (RL) algorithms are divided into two main paradigms: model-free (MFRL) and model-based (MBRL) types. While achieving good asymtotic performances in many high dimensional problems (Mnih et al., 2015; Silver et al., 2017; Schulman et al., 2017; Hessel et al., 2018; Espeholt et al., 2018), MFRL methods suffer from high sample complexity since they learn state/state-action values only from rewards and do not explicitly exploit the rich information underlying the transition dynamics data. On the contrary, MBRL approaches, by trying model the transition dynamics that are in turn used for planning without having to frequently interacting with real systems, are known to have sample efficiency and thus possess more practicability (Deisenroth et al., 2013; Finn et al., 2016; Ebert et al., 2018; Sutton & Barto, 2018; Kaiser et al., 2019).

Current MBRL methods, however, still have limitations because the accuracy of the learned dynamics model is usually not satisfied, especially in complex environments (Zhang et al., 2018; Lowrey et al., 2018). The model error and its compounding effect when planning, i.e. a small bias in the model can lead to a highly erroneous value function estimate and a strongly-biased suboptimal policy, make MBRL less competitive in terms of asymptotic performance than MFRL for many non-trivial tasks. Numerous attempts have been made to tackle with this model bias problem but none of them have been really successful, such as using Gaussian Process (GP) (Deisenroth & Rasmussen, 2011; Gal & Ghahramani, 2016), Bayesian Neural Networks (Gal et al., 2016; Depeweg et al., 2016a; Kamthe & Deisenroth, 2017), and Emsembling (Kurutach et al., 2018; Clavera et al., 2018).

Another limitation of many existing MBRL methods is that they rely on the model predictive control (MPC) framework (Garcia et al., 1989). While being commonly used, MPC has serveral drawbacks (Atkeson & Schaal, 1997; Thananjeyan et al., 2019). First, each step requires solving a high-dimensional optimization problem and thus is computationally prohibitive for applications requiring either real-time or low-latency reaction such as autonomous driving. Second, the policy is only implicit via solving the mentioned optimization problem. Not being able to explicitly represent the policy makes it hard to transfer the learned policy to other tasks or to initialize agents with an existing better-than-random policy.

Contributions. To address those challenges of MBRL, we propose a new framework called Policy Optimization with Uncertainty-aware Model (POUM) that is able to optimize in the face of uncertainty. Our policy optimization is based on Policy Gradient, which has been widely adopted in MFRL (Lillicrap et al., 2015; Schulman et al., 2017; Haarnoja et al., 2018). However, in POUM, the objective function, a utility function, is formulated around the uncertainty-aware dynamics model. This utility function takes into account both the mean and the variance of the value function estimate. This helps reducing the model bias while effectively approximating true objective, which is the value function of the policy. For experiments, we demonstrate the advantages of POUM over state-of-the-art (SoTA) methods on various RL tasks given training from scratch and all the environments are unaltered, and also investigate on how much risk is tolerable in those tasks. And last, POUM can be represented end-to-end in a single computation graph, which greatly facilitates the training.

2 Related work

Traditional MBRL. Initial successes of MBRL in continuous control achieved promising results by learning control policies trained on models of local dynamics using linear parametric approximators (Abbeel et al., 2007; Levine & Koltun, 2013). Alternative methods such as Deisenroth & Rasmussen (2011); Levine & Koltun (2013) incorporated non-parametric probabilistic GPs to capture model uncertainty during policy planning and evaluation. While these methods enhance data efficiency in low-dimensional tasks, their applications in more challenging domains such as environments involving non-contact dynamics and high-dimensional control remain limited by the inflexibility of their temporally local structure and intractable inference time. Our approach, on the contrary, pushes the uncertainty modeling to the objective function and not anywhere else in the architecture. Plus, the fact that this objective is designed to propagate all the way to the value function makes it versatile in capturing uncertainty. What is more, all core components are constructed by neural networks gives our solution more power in dealing with high-dimensional tasks, thus acquiring asymptotically high performance compared to MFRL methods and, at the same time, retaining data efficiency in those complex domains.

Deep Neural Networks (DNNs). Recently, there has been a revived interest in using DNNs to learn predictive models of environments from data, drawing inspiration from ideas in the early literature on this MBRL field, mainly because the large representational capacity enables them as suitable function approximators for complex environments, especially that involve images or videos (Ebert et al., 2018; Kaiser et al., 2019). However, additional care has to be usually taken to avoid model bias, a situation where the DNNs overfit in the early stages of learning, resulting in inaccurate models. For example, Depeweg et al. (2016b) modeled a Bayesian type of DNNs to capture uncertainty in transition dynamics. In another approach, Nagabandi et al. (2017) combined a learned dynamics network with MPC to initialize the policy network to accelerate learning in model-free deep RL. Chua et al. (2018) extended this idea by introducing a bootstrapped ensemble of probabilistic DNNs to model predictive uncertainty of the learned networks and demonstrating that a pure model-based approach can attain the asymptotic performance of MFRL counterparts. However, the use of MPC to define a policy leads to poor run-time execution and hard to transfer policy across tasks. On the contrary, our framework is much simpler in that we do not employ any extra method to model the dynamics uncertainty into DNNs that are already complicated itself with numerous architectures and hyperparameters, but instead formulate a single, new uncertainty-aware objective for end-to-end optimization.

Ensemble. Another group of work leveraged the learned ensemble of dynamics models to train a policy network. Kurutach et al. (2018) learned a stochastic policy via trust-region policy optimization, and Clavera et al. (2018) casted the policy gradient as a meta-learning adaptation step with respect to each member of the ensemble. Buckman et al. (2018) proposed an algorithm to learn a weighted combination of roll-outs of different horizon lengths, which dynamically interpolates between model-based and model-free learning based on the uncertainty in the model predictions. To our knowledge, this is the closest work in aside from ours, which learns a reward function in addition to the dynamics function. But none of the aforementioned work propagates the uncertainty all the way to the value function and uses the concept of utility function to balance risk and return as in our model.

Finally, ensemble of DNNs also provide a straightforward technique to obtain reliable estimates of predictive uncertainty (Lakshminarayanan et al., 2017) and has been integrated with bootstrap to guide exploration in MFRL (Osband et al., 2016; Janner et al., 2019). While many of the approaches mentioned in this section employ bootstrap to train an ensemble of models, we note that their implementations comprise of reconstructing bootstrap datasets at every training iteration, which effectively trains every single data sample and thus diminishes the advantage on uncertainty quantification achieved through bootstrapping. Except for a novel objective formulation, our model is different in that, to maintain online bootstrapped datasets across ensembles, it adds each incoming data sample to a dataset according to a Poisson probability distribution (Park et al., 2007; Qin et al., 2013), thereby guaranteeing asymptotically consistent those datasets.

3 Uncertainty-aware Model-based Policy Optimization

3.1 Background

Consider a discrete-time Markov Decision Process (MDP) defined by a tuple $M = \{S, A, f, r, \gamma\}$, in which S is a state space, A is an action space, $f : S \times A \to S$ is a deterministic (or probabilistic) transition function, $r : S \times A \to \mathbb{R}$ is a deterministic reward function, and $\gamma \in (0, 1)$ is a discount factor. We define the return as sum of the rewards $r\left(s_t, a_t\right) = r\left(s_t, \pi(s_t)\right)$ for $t = 0, \ldots, T$ for the whole trajectory $(s_0, a_0, ..., s_T, a_T)$ induced by a policy $\pi : S \to A$ and discounted by γ. Here $T \in \mathbb{Z}_+$ is a task horizon, which may take a value of ∞ for non-episodic environments. The goal of RL is to find an optimal policy $\pi^\star$ to maximize the expected return

$$J(\pi) = \mathop{\mathbb{E}}_{s_0 \sim S}[V^\pi(s_0)] \tag{1}$$

where the value function is defined as $V^\pi(s_0) = \sum_{t=0}^{T-1} \gamma^t \, r(s_t, \pi(s_t))$, and the state transition is $s_{t+1} = f(s_t, \pi(s_t))$, with s_0 being randomly chosen from the distribution of $s \in S$. Then if the dynamics function f and the reward function r are given, solving Equation 1 can be done using the Calculus of Variations (Young, 2000) or using Policy Gradient (Sutton et al., 2000) when the control function is parameterized or is finite dimensional.

In RL, however, f and r are often unknown and hence Equation 1 becomes a blackbox optimization problem with an unknown objective function. Following the Bayesian approach commonly used in the blackbox optimization literature (Shahriari et al., 2015), we propose to solve this problem by iteratively learning a probabilistic estimate $\widehat{V}$ of V from data and optimizing the policy according to this approximate model, as detailed in the next section.

3.2 Formulation of Uncertainty-Aware Optimization Objective

It is worth noting that any unbiased method would model $\widehat{V}(\pi)$ as a probabilistic estimate, i.e. $\widehat{V}(\pi)$ would be a distribution (as opposed to a point estimate) for a given policy π. Optimizing a stochastic objective is, however, not well-defined. Our solution is to transform $\widehat{V}$ into a deterministic utility function that reflects a subjective measure balancing the risk and return. Following Markowitz (1952); Sato et al. (2001); Garcıa & Fernández (2015), we propose a risk-sensitive objective criterion using a linear combination of the mean and the standard deviation of $\widehat{V}(\pi)$. Formally stated, our objective criterion, which we also call the utility function, now becomes

$$U(\pi)(s_0) = \mathop{\mathbb{E}}_{s_0 \sim S}\left[\mu\left(\widehat{V}(\pi)(s_0)\right) + c \times \sigma\left(\widehat{V}(\pi)(s_0)\right)\right], \tag{2}$$

where μ and σ are respectively the mean and the standard deviation of $\widehat{V}(\pi)(s_0)$, and c is a constant that represents the subjective risk preference of the learning agent. A positive risk preference infers that the agent is adventurous while a negative risk preference indicates that the agent has a safe exploration strategy. To our best knowledge, this uncertainty-aware model-based objective function has not been used in the RL literature.

3.3 Empirical Estimate of Value Function

Section 3.2 provides a general framework for policy optimization under uncertainty, assuming the availability of the estimation model $\widehat{V}(\pi)$ of the true value function $V(\pi)$. In this section, we describe how to estimate $\widehat{V}(\pi)$ with a model-based approach. The main idea is to approximate the functions $\{f, r\}$ with probabilistic parametric models $\{\widehat{f}, \widehat{r}\}$ and fully propagate the estimated uncertainty when planning under each policy π from an initial state s_0. The value function estimate $\widehat{V}$ can be formulated as

$$\widehat{V}(\pi)(s_0) = \sum_{t=0}^{T-1} \gamma^t \widehat{r}\left(\hat{s}_t, \pi(\hat{s}_t)\right), \tag{3}$$

where $\hat{s}_0 = s_0$ and $\hat{s}_{t+1} = \widehat{f}(\hat{s}_t, \pi(\hat{s}_t))$ for $t = 0, \dots, T-1$. Next, we describe how to efficiently model $\{f, r\}$ with well-calibrated uncertainty and a rollout technique that allows the uncertainty to be faithfully propagated into $\widehat{V}(\pi)$.

3.3.1 Bootstrap Setup for Model Learning

Following the traditional bootstrap methodology, the empirical model function $\widehat{f}$ is represented as $\{\widehat{f}_{\phi_k}(s_t, a_t) \rightarrow s_{t+1}\}_{k=1}^{B}$. For simplicity of implementation, we model each bootstrap replica as deterministic and rely on the ensemble as the sole mechanism for quantifying and propagating uncertainty. Each bootstrapped model $\widehat{f}_{\phi_k}$, which is parameterized by ϕ_k, learns to minimize the $L2$ one-step prediction loss over the respective bootstrapped dataset $\mathbb{D}_k$:

$$\min_{\phi_{k:=1\mapsto B}} \mathbb{E}_{(s_t, a_t, s_{t+1})\sim\mathbb{D}_k} \|s_{t+1} - \widehat{f}_{\phi_k}(s_t, a_t)\|_2^2. \tag{4}$$

The training dataset $\mathbf{D}$, from which the bootstrapped datasets $\{\mathbb{D}_k\}_{k=1}^{B}$ are sampled, stores the transitions on which the agent has experienced. Since each model observes its own subset of the real data samples, the predictions across the ensemble remain sufficiently diverse in the early stages of the learning and will then converge to their true values as the error of the individual networks decreases.

In addition to model estimation and unlike many other model-based approaches, we also learn the reward function along the same design of classical MBRL algorithms Sutton (1991). But in POUM, we use a deterministic model (also parameterized by a DNN) for the reward function to simplify the policy evaluation.

3.3.2 Bootstrap Rollout

In this section, we describe how to propagate the estimates with uncertainty from the dynamics model to evaluate a policy π. We represent our policy $\pi_\theta : S \rightarrow A$ as a neural network parameterized by θ . Note that we choose to represent our policy as *deterministic*. We argue that while all estimation models, including that of the dynamics and of the value function, need to be stochastic (i.e. uncertainty-aware), the policy does not need to be. The policy is not an estimator and deterministic policy simply means that the agent is consistent when taking an action, no matter how uncertain it may know about the world.

Given a deterministic policy π_θ and an initial state $s_0 \in \mathbb{D}$, we can estimate the distribution of $V(\pi)(s_0)$ by simulating π_θ through each each bootstrapped dynamics model. And since each bootstrap model is an independent approximator of the dynamics function, by expanding the value function via these dynamics approximators, we eventually obtain independent estimates of that value function. Finally, those separate and independent trajectories collectively form an ensemble estimator of V.

In practice, we sample these trajectories with a finite horizon $H < T$. It is still a challenge to expand the value function estimation for a very long horizon due to a few reasons. First, DNNs training becomes harder when the depth increases. Second, despite our best effort to control the uncertainty, we still do not have a guarantee that our uncertainty modeling is perfectly calibrated, which in turn may be problematic if the planning horizon is too large. Finally, policy learning time is proportional to the rollout horizon.

3.4 Policy Gradient

Based on Equation 2, the optimization target to optimize based on policy gradient method is:

$$\arg\max_{\theta} J(\theta) = \mathbb{E}_{s\sim S}\left[U_\theta(s)\right], \tag{5}$$

where $U_\theta(s) = \mu(\widehat{V}_\theta(s)) + c \times \sigma(\widehat{V}_\theta(s))$. Using the ensemble method and the rollout technique described above, we can naturally compute $\mu(\widehat{V}_\theta(s))$ and $\sigma(\widehat{V}_\theta(s))$ for a given policy π_θ and for a given state s. Therefore, the policy π_θ can be updated using the SGD or a variance of it.

Importantly, in terms of implementation, it is worth noting that the aforementioned rollout method also allows for easily expressing $U(\theta)$ in Equation 5 as a single computational graph of θ. This makes it straightforward to compute the policy gradient $\nabla_\theta U_\theta(s)$ using automatic differentiation, a feature provided by default in most popular deep learning toolkits.

4 Algorithm Summary

Algorithm 1 Policy Optimization with Uncertainty-aware Model (POUM)

```
Initialize a training dataset $\mathbf{D}$, bootstrapped datasets $\{\mathbb{D}_i\}_{i=1}^B$, parameterized bootstrapped models $\{\widehat{f}_i\}_{i=1}^B$, a parameterized reward model $\hat{r}_\phi$, and a parameterized deterministic policy $\pi_\theta$.
while not done do
    • Step in the environment, collect new data point $(s, a, s', r)$ and push into $\mathbf{D}$,
    • Sample from $\mathbf{D}$ and push data into the bootstrapped replay buffers: for each member $i^{th}$ in the ensemble, add $z_i \sim Poisson(1)$ copies of that data point to $\mathbb{D}_i$,
    • Update $\{\widehat{f}_i\}_{i=1}^B$ on $\mathbb{D}_i$ and $\hat{r}_\phi$ on $\mathbf{D}$ using SGD,
    • Evaluate $\widehat{V}_\theta(s)$ and $U_\theta(s)$ by simulating through the learned models $\{\widehat{f}_i\}_{i=1}^B$ and $\widehat{r}_\phi$,
    • Update $\pi_\theta$ using SGD with the policy gradient being backpropagated on $\mathbb{E}_s\left[U_\theta(s)\right]$ through the learned models.
end while
```

We summarize our framework POUM in Algorithm 1 and later in this section, we will also highlight some important details in our implementation.

4.1 Dynamics Model Learning with Online Bootstrap

As discussed in Section 1, there are several prior attempts to learn uncertainty-aware dynamics models such as GPs, Bayesian neural networks (NNs), dropout NNs and ensemble of NNs. In this work, however, we employ an ensemble of bootstrapped DNNs. Bootstrap is a generic, principled and statistical approach for uncertainty quantification. Furthermore, as will be also later explained in Section 3.4, this ensembling approach also gives rise to easy gradient computation.

4.1.1 Online bootstrap for training data

Bootstrap learning is often studied in the context of batch learning. However, since our agent updates its empirical model $\widehat{F}$ after each physical step for the best possible sample efficiency, we follow an online bootstrapping method by sampling from Poisson distribution (Oza, 2005; Qin et al., 2013). This is a very effective online approximation to batch bootstrapping, and can be easily done by this simple rule: bootstrapping a dataset $\mathbb{D}$ with n examples means sampling n examples from $\mathbb{D}$ with replacement. In detail, each example i will appear z_i times in the bootstrapped sample where z_i is a random variable whose distribution is $Binom(n, 1/n)$ because during resampling, the i^{th} example will have n chances to be picked, each with probability $1/n$. This $Binom(n, 1/n)$ distribution converges to $Poisson(1)$ when $n \to \infty$. Therefore, for each new data point, this method adds z_k copies of that data point to the bootstrapped dataset $\mathbb{D}_k$, where z_k is sampled from a $Poisson(1)$.

Online off-policy learning. Except for the initialization step (we may initialize the models with batch training from off-policy data), our model learning is an online learning process. For each time step, the learning cost stays constant and does not grow over time, which is required for lifelong learning. Despite being online, the learning is off-policy because we maintain a bootstrapped replay buffer for each model in the ensemble. For each model update, we sample a minibatch of training data from the respective replay buffer. In addition, as mentioned, the models can also be initialized from existing data even before the policy optimization starts.

4.1.2 Linearly-weighted sampling from bootstrapped training data

Since our replay buffers are accumulated online, a naive uniformly sampling strategy would lead to early data being sampled more frequently than the later ones. We thus propose a linearly weighted random sampling scheme to mitigate this early-data bias issue. In this sampling scheme, example i^{th} is randomly sampled with weight i, i.e. higher weights for the fresher examples in each online update step. Despite its simplicity, this scheme plays an important role in data bias removal, as shown in Appendix A.1.

5 Experiment

Our experiments are designed to help 1) compare our POUM framework with other SoTA approaches and 2) investigate the impact of the risk factor in our utility function on guiding agents.

5.1 Comparison to baseline algorithms

Experimental Design. We evaluate the performance of our POUM algorithm on four continuous control tasks including: one classic control task (Pendulum-v0) and three other tasks in the MuJoCo simulator (Todorov et al., 2012) from OpenAI Gym (Brockman et al., 2016). It is important to note that, we keep the default configurations prodived by OpenAI Gym (See Appendix A.2.1) and also does not assume access to the reward function as some recent works in model-based reinforcement learning (Chua et al., 2018; Clavera et al., 2018; Kurutach et al., 2018).

For the baselines, we compare POUM to the following SoTA algorithms designed for continuous control: MBPO (Janner et al., 2019), DDPG (Lillicrap et al., 2015), SAC (Haarnoja et al., 2018), STEVE (Buckman et al., 2018). For each one of them, we evaluate the learned policy after every episode. The evaluation is done by running the current policy on 20 random episodes and then computing the average return over them.

Results. Figure 5.1 shows that POUM has a sample efficiency compared to the baseline algorithms across a wide range of environments. Furthermore, it also has the asymptotic performance competitive to or even better than that of the model-free counterparts. Note that, there are horizontal parts at the beginning of evaluation curves in some algorithms and environments, that because these algorithms take random exploration at the beginning of training (as their default configuration) to initialize dynamics. For simple environments: Pendulum-v0, Reacher-v2, Push-v2, our POUM can get a good performance without initialized dynamics[1].

However, Figure 5.1 also shows that the performance of POUM in more complex environment like HalfCheetah-v2 is sensitive to random seeds. We hypothesize that this is due to the impact of risk-preference value on policy optimization framework and our strategy of aggressive online learning and linearly weighted batch sampling. The ablation study below validates our current analysis on these hypotheses.

5.2 Ablation study

To obtain a better understanding about the role of the subjective risk preference in the utility function, we conduct an ablation study on the parameter c that controls this risk factor (Equation 2) and make the following observations.

As illustrated in Figure 5.2, as the first observation, complete zero risk is not a good choice. This behavior is expected because no risk means no uncertainty is quantified properly, leading to agents

[1] MBPO failed to attain a good performance for Reacher-v2 and Pusher-v2, regardless our best effort to produce the results based on the authors' official repository.

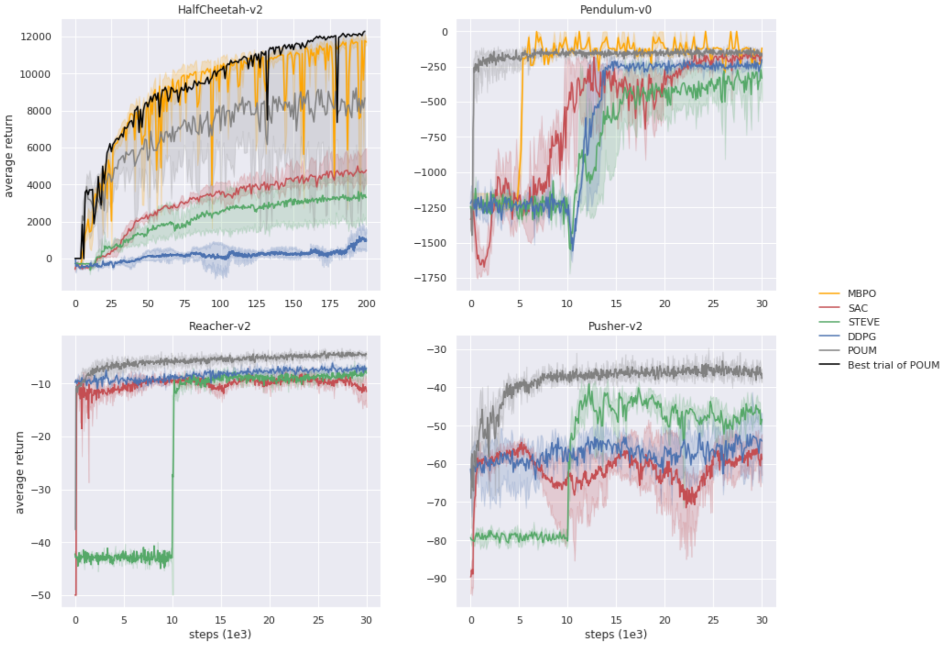

Figure 1: Average return of POUM model over 3 different randomly selected random seeds compared with SoTA appoaches. Solid lines indicate the mean and shaded areas indicate one standard deviation. POUM beats all other solutions on environments tested, except for HalfCheetah-v2 where it has a competitive performances compared to MBPO.

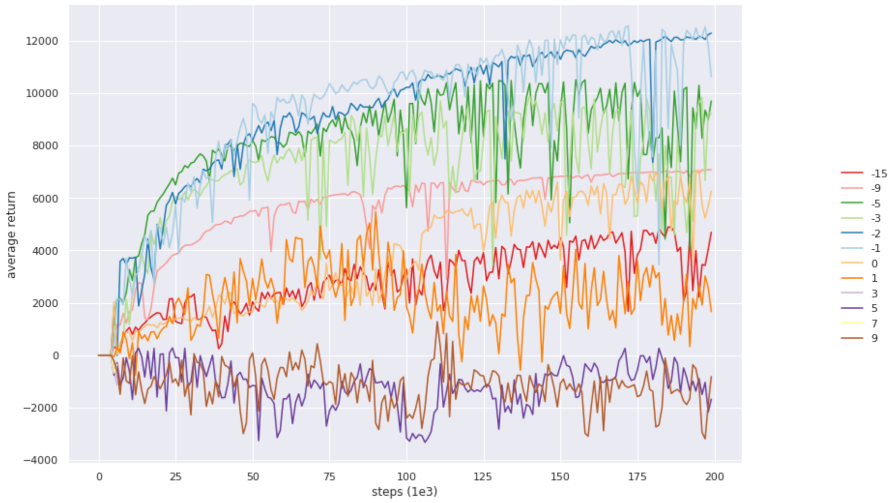

Figure 2: POUM with different subjective risk preference values on HalfCheetah-v2 environment. The moderate risk yields the best return while too much risk (either too high or too low) will harm the agent in getting good results. We use the same settings of HalfCheetah-v2 environment as presented in Appendix A.2.3 excluding risk-preference values

could not learn well to model the dynamics, as well as optimize an efficient policy. In another observation, POUM performs best with the risks $c = -1$, follows by $c = -2$ and $c = -3$, while it gets worse and worse at both directions, the risk factor goes either higher or lower. This phenomenon is because the risk factor controls the scale of standard deviation, and hence the variance. Too low or too high risks, consequently, imply too much variance which is not favorable in many cases because it indudes agents to explore more aggressively and hence suffer more potential failures, while not exploiting current, safer experiences. Finally and interestingly, as the scale of the risk changes, both directions are not behaving the same. In particular, POUM gets bad results with positive risk-preference value, and even can not learn with high positive value. That is because in current work, we use fixed the subjective risk preference value and at the beginning of learning process, the dynamics models are unstable and high variance, with high positive risk-preference values, policy learning strange decisions. In contrast, with negative risk-preference, our utility function work as lower confidence bound that keep policy in a safe region. The figure indicates that with lower negative risk-preference value, the learning curve is more stable. However, lower risk-preference value means that less exploration, and results in lower final reward.

6 Discussion and Conclusion

In summary, this paper proposed a new approach in MBRL in which we developed a novel objective function that balances the mean and variance in the estimation of the value function, which is induced by the model. Our experiments suggest that our POUM algorithm not only can achieve the asymptotic performance of model-free methods in challenging continuous control tasks and compared to other SoTA approaches, it does so in much fewer samples. We further demonstrate that the model bias issue in model-based RL can be dealt with effectively with principled and careful uncertainty quantification, by guiding agents with a subjective risk factor. Unlike other methods, quantifying and controlling the uncertainty with a novel uncertainty-aware objective function, and without any complex designs for other components is an advantage, of being simple yet efficient, compared with others.

Nonetheless, we acknowledge that our current implementation for POUM still has several limitations, such as high variance in the empirical performance, which still depends on many hyper-parameters (plan horizon, risk sensitivity, and all hyper-parameters associated with neural networks training techniques) and even depends on random seeds. It is, however, worth noting that these traits are not unique to our methods. In spite of this limitation, the results indicate that if implemented properly, MBRL methods can be both sample efficient and have better asymptotic performances than the MFRL counterparts on challenging tasks. In addition, by explicitly representing both the dynamics model and the policy, POUM enables transfer learning, not just for the world (dynamics) model but also for the policy.

To sum up, we identify that sample efficiency, off-policy learning, and transferability are three necessary, albeit not sufficient, properties for real-world reinforcement learning. We claim that our method meets these criteria and hence is a step towards real-world reinforcement learning.

References

Pieter Abbeel, Adam Coates, Morgan Quigley, and Andrew Y Ng. An application of reinforcement learning to aerobatic helicopter flight. In *Advances in neural information processing systems*, pp. 1–8, 2007.

Christopher G Atkeson and Stefan Schaal. Learning tasks from a single demonstration. In *Proceedings of International Conference on Robotics and Automation*, volume 2, pp. 1706–1712. IEEE, 1997.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Jacob Buckman, Danijar Hafner, George Tucker, Eugene Brevdo, and Honglak Lee. Sample-efficient reinforcement learning with stochastic ensemble value expansion. In *Advances in Neural Information Processing Systems*, pp. 8224–8234, 2018.

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*, pp. 4754–4765, 2018.

Ignasi Clavera, Jonas Rothfuss, John Schulman, Yasuhiro Fujita, Tamim Asfour, and Pieter Abbeel. Model-based reinforcement learning via meta-policy optimization. *arXiv preprint arXiv:1809.05214*, 2018.

Marc Deisenroth and Carl E Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pp. 465–472, 2011.

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.

Stefan Depeweg, José Miguel Hernández-Lobato, Finale Doshi-Velez, and Steffen Udluft. Learning and policy search in stochastic dynamical systems with bayesian neural networks. *arXiv preprint arXiv:1605.07127*, 2016a.

Stefan Depeweg, José Miguel Hernández-Lobato, Finale Doshi-Velez, and Steffen Udluft. Learning and policy search in stochastic dynamical systems with bayesian neural networks. *arXiv preprint arXiv:1605.07127*, 2016b.

Frederik Ebert, Chelsea Finn, Sudeep Dasari, Annie Xie, Alex Lee, and Sergey Levine. Visual foresight: Model-based deep reinforcement learning for vision-based robotic control. *arXiv preprint arXiv:1812.00568*, 2018.

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymir Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. *arXiv preprint arXiv:1802.01561*, 2018.

Chelsea Finn, Xin Yu Tan, Yan Duan, Trevor Darrell, Sergey Levine, and Pieter Abbeel. Deep spatial autoencoders for visuomotor learning. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 512–519. IEEE, 2016.

Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pp. 1050–1059, 2016.

Yarin Gal, Rowan McAllister, and Carl Edward Rasmussen. Improving pilco with bayesian neural network dynamics models. In *Data-Efficient Machine Learning workshop, ICML*, volume 4, 2016.

Carlos E Garcia, David M Prett, and Manfred Morari. Model predictive control: theory and practice—a survey. *Automatica*, 25(3):335–348, 1989.

Javier Garcıa and Fernando Fernández. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(1):1437–1480, 2015.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.

Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to trust your model: Model-based policy optimization. *arXiv preprint arXiv:1906.08253*, 2019.

Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, et al. Model-based reinforcement learning for atari. *arXiv preprint arXiv:1903.00374*, 2019.

Sanket Kamthe and Marc Peter Deisenroth. Data-efficient reinforcement learning with probabilistic model predictive control. *arXiv preprint arXiv:1706.06491*, 2017.

Thanard Kurutach, Ignasi Clavera, Yan Duan, Aviv Tamar, and Pieter Abbeel. Model-ensemble trust-region policy optimization. *arXiv preprint arXiv:1802.10592*, 2018.

Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in Neural Information Processing Systems*, pp. 6402–6413, 2017.

Sergey Levine and Vladlen Koltun. Guided policy search. In *International Conference on Machine Learning*, pp. 1–9, 2013.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Kendall Lowrey, Aravind Rajeswaran, Sham Kakade, Emanuel Todorov, and Igor Mordatch. Plan online, learn offline: Efficient learning and exploration via model-based control. *arXiv preprint arXiv:1811.01848*, 2018.

Harry Markowitz. Portfolio selection. *The journal of finance*, 7(1):77–91, 1952.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

Anusha Nagabandi, Gregory Kahn, Ronald S. Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. *CoRR*, abs/1708.02596, 2017. URL `http://arxiv.org/abs/1708.02596`.

Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped dqn. In *Advances in neural information processing systems*, pp. 4026–4034, 2016.

Nikunj C Oza. Online bagging and boosting. In *2005 IEEE international conference on systems, man and cybernetics*, volume 3, pp. 2340–2345. Ieee, 2005.

Byung-Hoon Park, George Ostrouchov, and Nagiza F Samatova. Sampling streaming data with replacement. *Computational Statistics & Data Analysis*, 52(2):750–762, 2007.

Zhen Qin, Vaclav Petricek, Nikos Karampatziakis, Lihong Li, and John Langford. Efficient online bootstrapping for large scale learning. *arXiv preprint arXiv:1312.5021*, 2013.

Makoto Sato, Hajime Kimura, and Shibenobu Kobayashi. Td algorithm for the variance of return and mean-variance reinforcement learning. *Transactions of the Japanese Society for Artificial Intelligence*, 16(3):353–362, 2001.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando De Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1): 148–175, 2015.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

Richard S Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *ACM SIGART Bulletin*, 2(4):160–163, 1991.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pp. 1057–1063, 2000.

Brijen Thananjeyan, Ashwin Balakrishna, Ugo Rosolia, Felix Li, Rowan McAllister, Joseph E Gonzalez, Sergey Levine, Francesco Borrelli, and Ken Goldberg. Extending deep model predictive control with safety augmented value estimation from demonstrations. *arXiv preprint arXiv:1905.13402*, 2019.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033. IEEE, 2012.

Laurence Chisholm Young. *Lecture on the calculus of variations and optimal control theory*, volume 304. American Mathematical Soc., 2000.

Marvin Zhang, Sharad Vikram, Laura Smith, Pieter Abbeel, Matthew J Johnson, and Sergey Levine. Solar: Deep structured latent representations for model-based reinforcement learning. *arXiv preprint arXiv:1808.09105*, 2018.

A Appendix

A.1 Why linearly weighted random sampling is a fairer sampling scheme

Consider the following online learning process: for each time step, we need to randomly sample an example from the accumulating dataset. Suppose that at time t, each example i^{th} is randomly sampled with weight $w(t, i)$. Note that at each time t, we have a total of t examples in the dataset. Then the probability of that example being sampled is

$$\frac{w(t,i)}{\sum_{k=1}^{t} w(t,k)}.$$

If we use uniformly random sampling then the expected number of times an example i^{th} gets selected until time t is

$$C_i^t = \sum_{k=i}^{t} \frac{1}{k}.$$

Hence, for all t, for $i > j$, C_i^t is larger than C_j^t by $\sum_{k=i}^{j} \frac{1}{k}$. Now, if we use a linearly weighted random sampling scheme, in which $w(t, i) = i$, then the expected number of times an example i^{th} gets selected until time t is

$$C_i^t = \sum_{k=i}^{t} \frac{2i}{k(k+1)} = 2\sum_{k=i}^{t} \left(\frac{i}{k} - \frac{i}{k+1}\right) = 2 - 2\frac{i}{t}.$$

We can see that at time t, C_i^t is still larger than C_j^t for $i < j$ but by weighting recent examples more in each online update step, we reduce the overall early-data bias.

A.2 Experimental settings

A.2.1 Environments

Table 1: Description of the environment used for testing

Environment	State dimension	Action dimension	Task horizon
Reacher-v2	11	2	50
Pusher-v2	23	27	100
Pendulum-v0	3	1	200
HalfCheetah-v2	23	6	1000

A.2.2 Network architecture

A.2.3 Hyper-parameter settings

Table 2: Network layer configurations

Environment	**bootstrap model:** $\widehat{f}_i$	**Reward model:** $\hat{r}_\phi$	**Policy:** π_θ
Reacher-v2	$[13, 256, 128, 11]$	$[13, 256, 128, 1]$	$[11, 128, 128, 2]$
Pusher-v2	$[30, 1024, 512, 256, 23]$	$[30, 1024, 512, 256, 1]$	$[23, 512, 256, 7]$
Pendulum-v0	$[4, 64, 64, 3]$	$[4, 64, 64, 1]$	$[3, 8, 1]$
HalfCheetah-v2	$[23, 1024, 512, 256, 17]$	$[23, 1024, 512, 256, 1]$	$[17, 512, 256, 6]$
Network format:	[input size, hidden layers size, output size]		

Table 3: Hyper-parameter settings

Hype-parameter	**Pendulum-v0**	**Reacher-v2**	**Pusher-v2**	**HalfCheetah-v2**
Bootstrap size	16	8	16	16
Plan horizon	20	30	35	20
Risk preference	-0.25	-0.25	0	-1
Policy updates (per environment step)	8	4	16	2
Dynamics update (per environment step)	8	6	6	6