Model Predictive Control with Learned Dynamics
CS 294-112

Vladimir Feinberg, Samvit Jain, Michael Whittaker

29 November 2017

1 Abstract

TODO one-page abstract

2 Introduction

Deep reinforcement learning has recently made progress in traditionally unsolvable problems for continuous control. Such continuous control problems are characterized by a MDP setting, where an agent interacts through the world, characterized by a set of states $\mathcal{S}$, by taking actions in $\mathcal{A}$ to maximize the expected net present value of a reward given over time. Episodes of a single problem share the same MDP setting, but differ in the initial state. Note we are interested in spaces where the dimension of $\mathcal{A}$ is moderately large, continuous, or both: adversarial and stochastic worst-case or high-probability methods are not applicable.

Model-free algorithms learn a direct mapping from $\mathcal{S}$ to $\mathcal{A}$, so all planning must be performed implicitly and only with state information. For this reason, model-based methods, which construct models for transitions between states, tend to have improved sample complexities relative to model-free algorithms.

On the other hand, constructing a model of the world, even in its restricted form as a simple dynamics prediction problem in continuous control, complicates the existing deep RL architecture. Purely model-based methods presume the model is accurate, and optimize expected reward under this assumption. Understandably, such models are inherently limited by how realistic the transition model is. For instance, one such pure model-based algorithm, iLQG, was outperformed by an algorithm using a learned model, Guided Policy Search [4]. Moreover, generic global models, such as neural networks, have not historically performed as well as simple, local linear models [2]. This is perhaps due to a latent interaction between the an agent's implicit planning and the explicit model-based planning with learned dynamics.

Altogether, pure model-based methods are fundamentally limited by model bias and pure model-free methods are fundamentally limited by policy variance.

Like every other problem in machine learning, we need to find the right trade-off between bias and variance. One way to bridge this gap is to explicitly include model-predicted rewards in the value estimates. In the style of SVG [3], we might accelerate value learning by mixing in a model-based expansion of our value estimates, where the value $V^\pi : \mathcal{S} \to \mathbb{R}$ (the true net present value for the reward at a given state) is approximated with the help of dynamics $f : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ describing transitions deterministically for simplicity and the reward $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ for a policy $\pi : \mathcal{S} \to \mathcal{A}$:

$$V^\pi(s_1) \approx \sum_{t=1}^{H} r(s_t, a_t, s_{t+1}) + \hat{V}(s_{H+1}); \qquad s_{t+1} \triangleq f(s_t, a_t), a_t \triangleq \pi(s_t)$$

While a viable approach, an implementation must take care to account for bias in f, which may skew estimates in V, no matter how good $\hat{V}$ is. Second, this introduces a nuanced coupling of the training of $f, \hat{V}, \pi$ at the same time.

3 Background

We investigate a use of dynamics models less-coupled with policy learning, Model Predictive Control (MPC), and its interaction with learned dynamics. We will revisit the potential for fused model-free and model-based models at the end of the report. MPC builds on top of open-loop planners, which accept a dynamics model and a given horizon H and choose the next H actions to play. MPC uses the planner to create a policy that iteratively queries the planner at each time step, playing only the first action in the plan (Algorithm 1).

Algorithm 1 The MPC algorithm acts as a policy, returning an action for a given state, using an internal planner and a dynamics model.

```
1: procedure MPC(state $s$, horizon $H$, reward $r$, dynamics $f$, planner $P$)
2:     $\{a_t\}_{t=1}^H = P(s, H, r, f)$
3:     return $a_1$
```

Most planners assume f can be treated as the true dynamics (or, if it has a simple probibalistic form, then the assumption corresponds to accuracy of the probabilistic model). Then a planner attempts the following maximization, which is frequently intractable:

$$\begin{aligned} \max_{\{a_t, s_t\}_{t=1}^H} \quad & \sum_{t=1}^H r(s_t, a_t) \\ \text{s.t.} \quad & s_{t+1} = f(s_t, a_t) \end{aligned} \tag{1}$$

A typical way of solving Equation 1 is with the random shooter, which samples the t-th action from $A_t(s_t)$ (Algorithm 2).

Algorithm 2 The RandomShooter algorithm is a planner approximately solving Equation 1, where quality of the solution improves with the number of trials it attempts. Altogether, RandomShooter is a stochastic policy available only in this generative form, returning an action a for a provided state s. The RandomShooter's properties will be highly dependent on the state-conditional time-dependent action sampling distributions A_t.

```
1: procedure RandomShooter(state $s$, horizon $H$, reward $r$, dynamics $f$, samplers $\{A_t\}_{t=1}^H$)
2:     $\left\{s_1^{(i)} \leftarrow s\right\}_{i=1}^K$.
3:     for $i \leftarrow 1, \ldots, K$ do
4:         for $t \leftarrow 1, \ldots, H$ do
5:             sample $a_t^{(i)} \sim A_t(s_t)$
6:             $s_{t+1}^{(i)} \leftarrow f\left(s_t^{(i)}, a_t^{(i)}\right)$
7:         $R_i \leftarrow \sum_{t=1}^H r\left(s_t^{(i)}, a_t^{(i)}, s_{t+1}^{(i)}\right)$
8:     $i_* = \operatorname{argmax}_i R_i$
9:     return $a_1^{(i_*)}$
```

We believe RandomShooter, or more generally any solution to Equation 1, works well when the corresponding reward R_i is a decent estimator for the true H-step reward of the agent. It is important to note that the agent takes $a_1^{(i_*)}$ now and continues with MPC later, so there will necessarily be some mismatch. In fact, one might expect positive bias, since the true reward comes from planning H steps ahead for each of the agent's next H steps, looking ahead $2H$ timesteps, as opposed to their initial estimate of its reward through Algorithm 2, which only looks H steps ahead.

Assuming a good reward estimator R_i, choosing the first action $a_1^{(i_*)}$ that results in the best R_i would be the smartest move one could make. Usually, one specifies $A_t(s) = \text{Uniform}\,\mathcal{A}$ for a rectangle $\mathcal{A}$, independent of t, s (Uniform RS MPC). But a uniform distribution is a poor representation of future MPC steps: it stands to reason that a more intelligent selection of A_t might improve performance.

4 Contributions

We find that one needs to be concerned with both underoptimizing and overoptimizing Equation 1. In particular, because f is learned and partially inaccurate, improvements in Equation 1 do not correspond to improvements in actual reward. For instance, letting $\tilde{R}$ be the true H-step reward, the reward predicted by Algorithm 2 with Uniform RS MPC, $\tilde{R} - R_{i_*}$ becomes increasingly negative as optimization quality is improved. As a function of optimization quality, the reward curve is actually U-shaped: too little optimization, and poor actions are chosen. Too much, and bias prevents or hurts improvement.

Given our findings, we present a new MPC objective instead of Equation 1 and propose several methods to investigate in future work.

5 Method

We consider learning a stationary action distribution conditional on the state, $\pi_\theta(s)$. This policy will be used to regularize the MPC optimization. In particular, if the planner is RandomShooter, then we might select $A_t^{(\pi_\theta)}$ to be a distribution that is some function of π_θ. Overall, we consider a joint training algorithm, Algorithm 3.

Algorithm 3 This assumes we have the testing environment on which we may perform real rollouts to learn from. We consider some fixed class of deterministic functions F to model dynamics, and a parameterization π_θ of generative action distributions conditioned on states, coupled with an associated loss J.

```
1: procedure TrainMPC(iterations $N$, starter transition dataset $\mathcal{D}$, reward $r$)
2:     for $i \leftarrow 1, \dots, N$ do
3:         $f \leftarrow \text{argmin}_F \mathbb{E}\left[\|f(s,a) - s'\|^2\right]$, where $(s,a,s') \sim \text{Uniform}\,\mathcal{D}$
4:         $\theta \leftarrow \text{argmin}_\theta J(\theta, \mathcal{D})$
5:         sample $E$ episodes $\mathcal{D}'$ of $\text{MPC}(\cdot, H, r, f, P_\theta)$, where $P_\theta$ is some planner parameterized by $\theta$.
6:         $\mathcal{D} \leftarrow \mathcal{D}' \cup \mathcal{D}$
7:     return $\text{MPC}(\cdot, H, r, f, P_\theta)$
```

Since the dynamics are learned from historical data, we propose a planner that optimizes Equation 2, where $\mathcal{F}_t$ is a region where we consider our dynamics model to be accurate.

$$\begin{aligned} \max_{\{a_t, s_t\}_{t=1}^H} \quad & \sum_{t=1}^{H} r(s_t, a_t) \\ \text{s.t.} \quad & s_{t+1} = f(s_t, a_t) \\ & (a_t, s_t) \in \mathcal{F}_t \end{aligned} \tag{2}$$

For instance, we might claim that our dynamics model is accurate for actions and states within some ϵ-ball of historical action and state distributions. In other words, let J be the cost for fitting a state-conditional action distribution π_θ on our historical data $\mathcal{D}$. Then we might define $\mathcal{F}_t \triangleq B_{\epsilon_t}(\pi_\theta(s_t)) \times B_{\epsilon'_t}(\bar{s}_t)$, where $\bar{s}_{t+1} = f(\bar{s}_t, \pi_\theta(\bar{s}_t))$: actions must be near a historical mean of actions taken in the corresponding state and states must be near the states that would come from a historical trajectory. $B_r(z)$ indicates a ball of radius r around z.

An approximate way of optimizing the above objective for $\epsilon_1 = \infty$ and $\epsilon_t, \epsilon'_t = 0$ otherwise would be to only allow RandomShooter to sample randomly on the first action, and otherwise stick to a policy representative of past actions (Equation 3). In general, setting the support of the sampling distributions for RandomShooter results in Monte Carlo optimization of the constrained problem.

$$A_t(s) = \begin{cases} \text{Uniform}\,(\mathcal{A}) & t = 1 \\ \pi_\theta(s) \triangleq \delta_{g_\theta(s)} & \text{otherwise} \end{cases} \tag{3}$$
$$J(\theta, \mathcal{D}) = \mathbb{E}_{(s,a)\sim\mathcal{D}} \left\|a - A\right\|^2 = \mathbb{E}_{(s,a)\sim\mathcal{D}} \left\|a - g_\theta(s)\right\|^2 ;\ A \sim \delta_{g_\theta(s)}$$

Above, g is some function parameterized by θ, such as a neural network.

We refer to TrainMPC with RandomShooter using the distributions of Equation 3 as δ RS MPC. We also consider a simple, parameter-less regularization $g_\theta(s) = \mathbf{0}$, called 0 RS MPC.

6 Results

6.1 Easy Cheetah

For preliminary validation of our ideas, we consider the environment provided in the course's Homework 4. In particular, we have the `HalfCheetah-v1` setting. Note that the reward function here includes additional penalties for undesirable behavior, making this an easy baseline to test on. [1]

The performance of the various MPC policies is shown in Figure 1. The underlying π_θ policy (henceforth the "learner") is poor (Figure 2), so the improvement it induces in MPC is likely coming from its regularizing properties rather than any smarter selection of actions that may be induced by the learner.

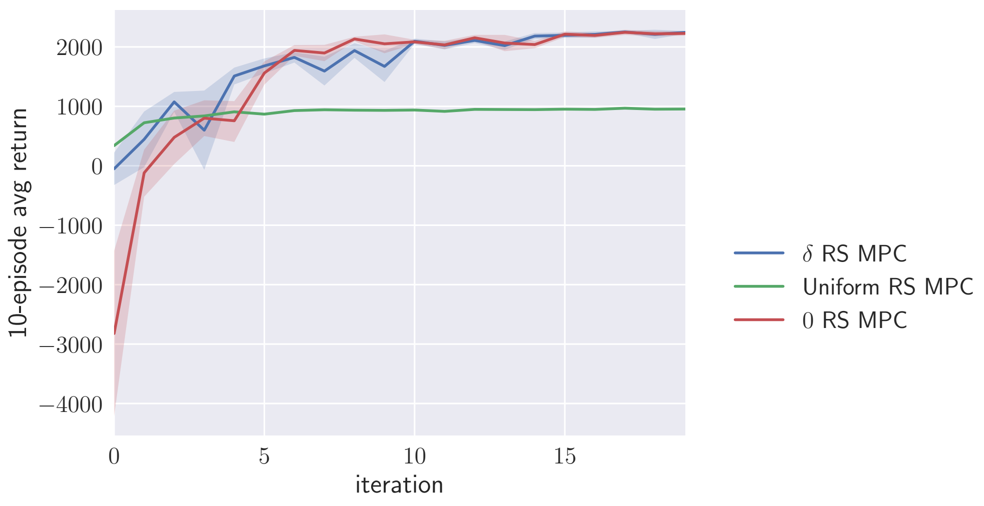

Figure 1: Performance during the course of optimization in the easy cheetah setting. Note both regularized versions significantly outperform Uniform MPC.

Notice both regularized versions reduce the bias of the predicted reward from the MPC optimization, $\tilde{R} - R_i$ (Figure 3). This is not due to catastrophic cancellation either, as the MSE $(\tilde{R} - R_i)^2$ figures demonstrate (Figure 4).

[1]Here our dynamics model is represented by a neural network of depth 2 and width 500. With an MPC horizon $H = 15$ and $K = 1000$ simulation paths, we evaluate the performance of the proposed methods over several iterations of the training procedure (Algorithm 3). Every iteration takes 10 sample episodes, and trains the dynamics for 60 epochs with a 10^{-3} learning rate. The policy optimization step, if present, does 100 epochs with 10^{-3} learning rate on a neural network g_θ of depth 5 and width 32. The universal batch size was 512. Reported results are averages over 4 seeds.

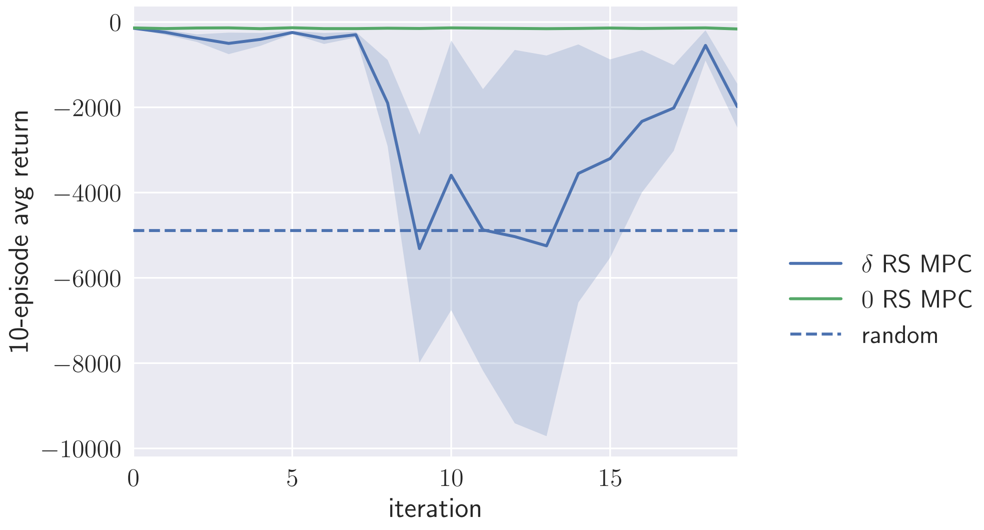

Figure 2: Performance of the underlying learner π_θ in the easy cheetah setting, during the course of optimization. We compare to the performance of an agent taking random actions uniformly on the action space.

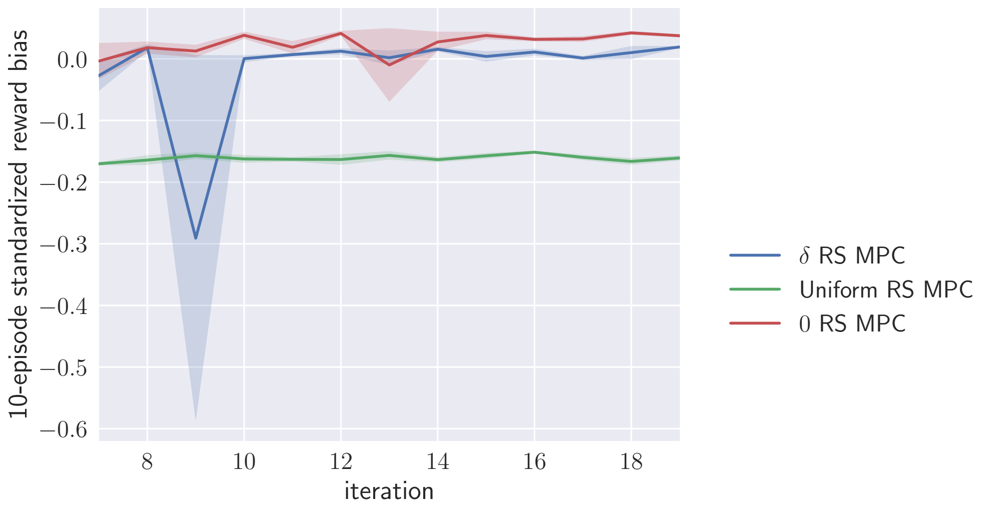

Figure 3: Easy cheetah MPC predicted reward bias is calculated as the true H-step reward minus the MPC's planner objective value H steps ago. A very negative value indicates overoptimism in the expected reward during MPC planning. We will refer to this as reward bias from this point.

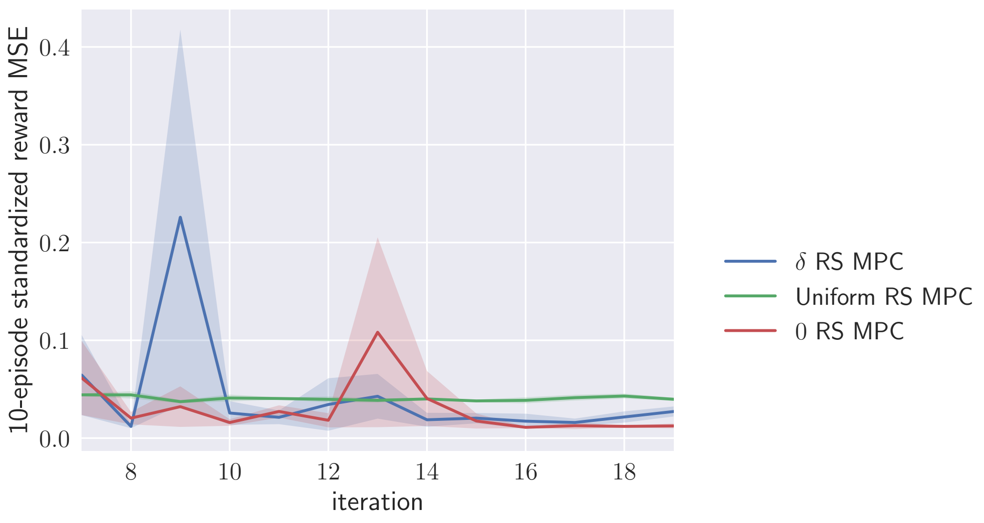

Figure 4: Easy cheetah MPC predicted reward MSE. This is the average bias of Figure 3, squared.

7 Reward Bias

We now investigate more carefully that it is over-optimization that occurs in MPC that optimizes for the unconstrained Equation 1. We consider the usual `HalfCheetah-v1` environment.[2] In particular, we improve the quality of the RandomShooter optimization of Equation 1 in the Uniform RS MPC procedure by increasing K. We note that as K increases, so does the over-optimism in the expected reward from the MPC objective (Figure 5, Figure 6). Note that the predicted reward is a function of the states projected by the MPC planner $\{s_t\}_{t=1}^H$. Then, a mismatch in the reward function implies a mismatch in predicted states, because the state space topology is necessary finer than the pushforward topology on the state space induced by the (continuous) reward function. This may explain why increases in K do not result in improvements in episode reward, and start to hamper performance for large K (Figure 7).[3]

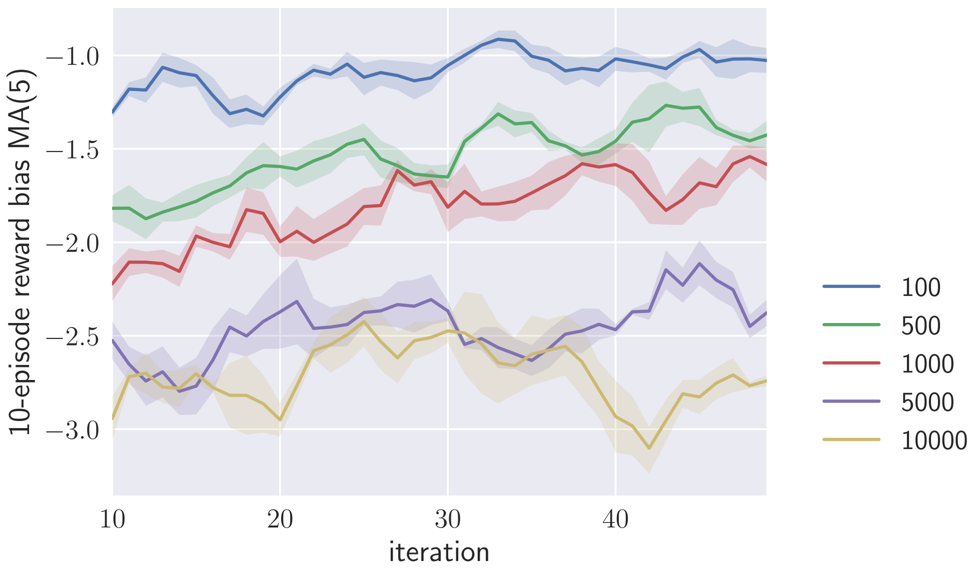

Figure 5: Half Cheetah predicted reward bias for Uniform RS MPC on a variety of K, smoothed over the 5 previous iterations.

[2]We use a minor modification to the observations in this environment to include additional position information in the state so that the usual Half Cheetah reward function is a function of the transitions. Otherwise MPC optimization would be impossible because usually the Half Cheetah enviroment determines the reward based on a latent coordinate position.

[3]Hyperparameters are as in Section 6.1, except for K which is modified as indicated.

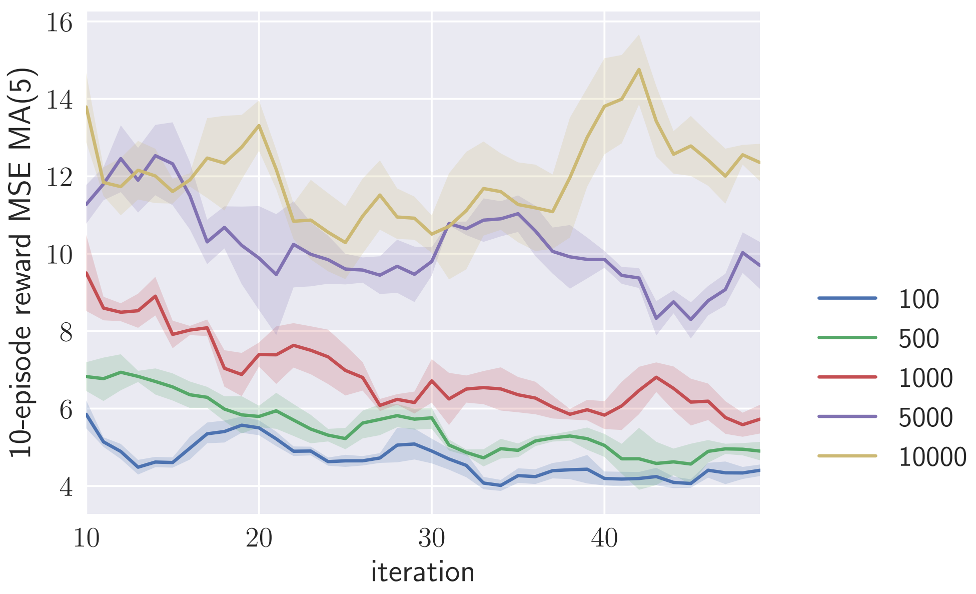

Figure 6: Half Cheetah predicted reward MSE for Uniform RS MPC on a variety of K, smoothed.

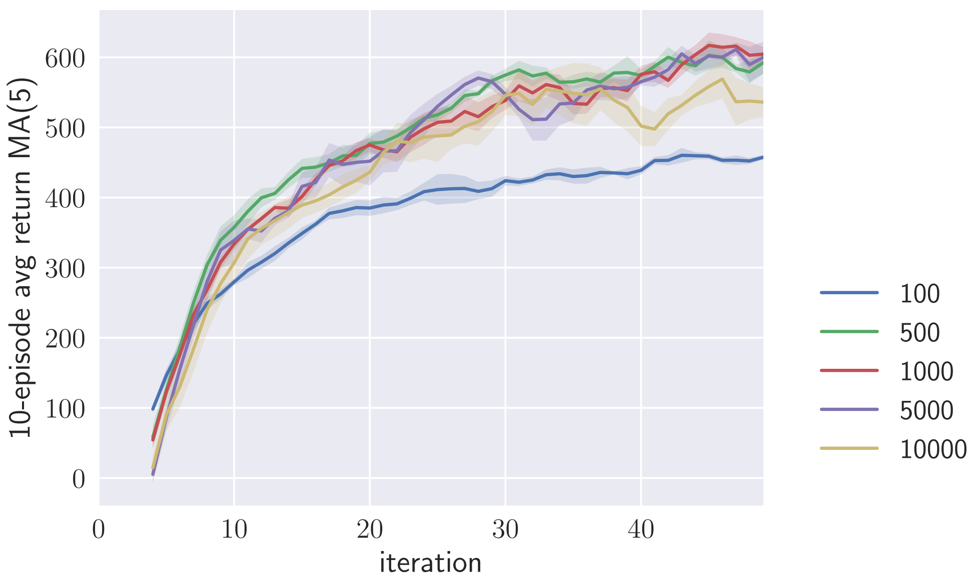

Figure 7: Half Cheetah reward for Uniform RS MPC on a variety of K, smoothed. Note that both under-optimizing and overoptimizing reduces performance.

8 Validation on Other Environments

We validate that regularizing MPC to solve Equation 2 with a Monte Carlo RandomShooter method (but supports of sampling distributions appropriately constrained) improves MPC performance across a variety of continuous control environments. We test on the `HalfCheetah-v1, Ant-v1, Walker2d-v1` environments, with observations slightly expanded to make the reward a function of the transition.[4]

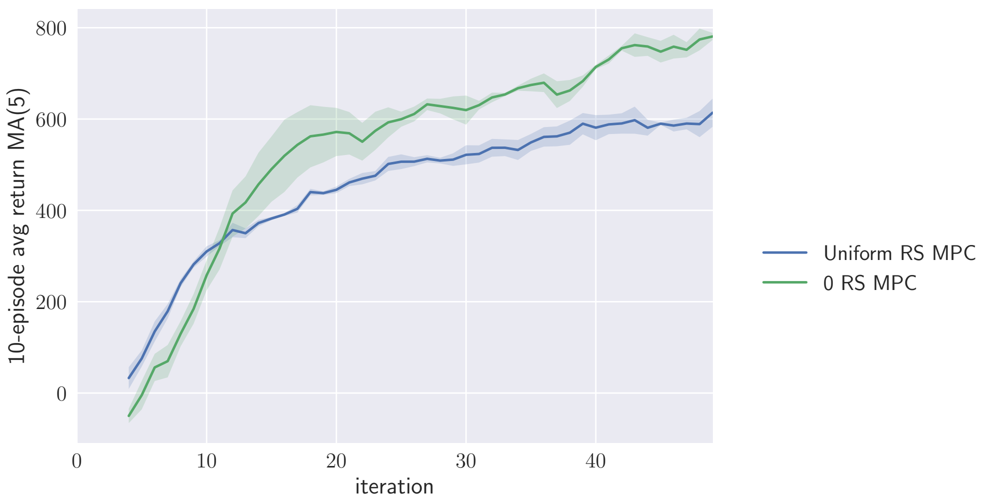

Figure 8: Half Cheetah reward for Uniform RS MPC vs 0 RS MPC with $K = 1000$ and $H = 15$.

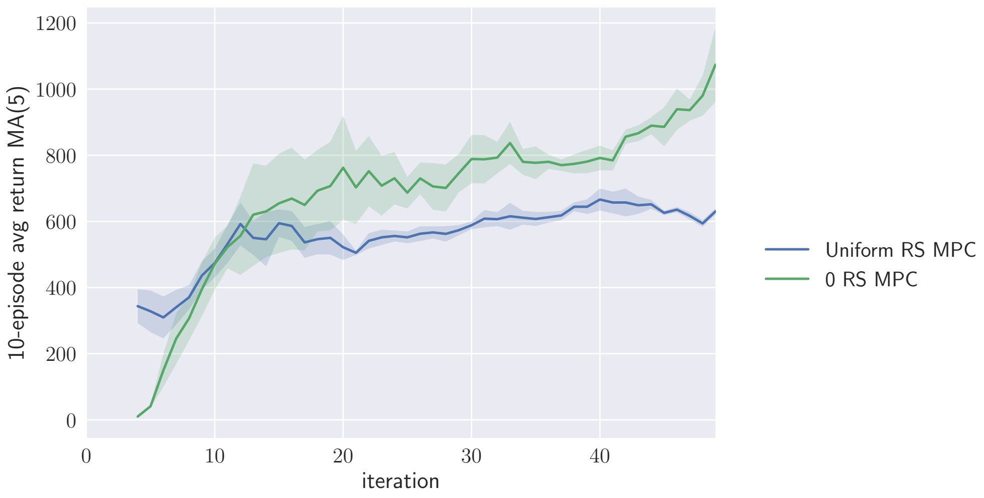

Figure 9: Half Cheetah reward for Uniform RS MPC vs 0 RS MPC with $K = 3000$ and $H = 30$.

[4]Hyperparameters are as in Section 6.1, except we only run for three random seeds and modify the simulation horizon and number of simulation paths as indicated.

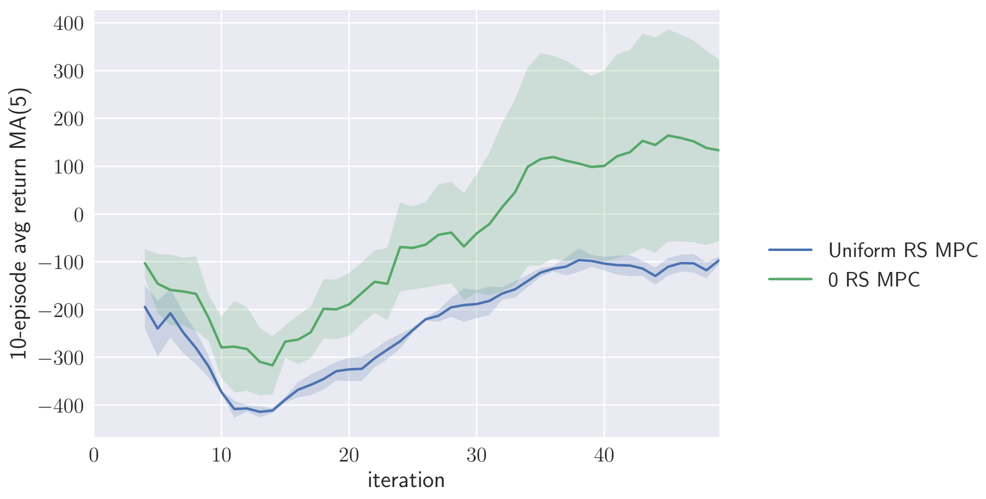

Figure 10: Ant reward for Uniform RS MPC vs 0 RS MPC with $K = 1000$ and $H = 15$.

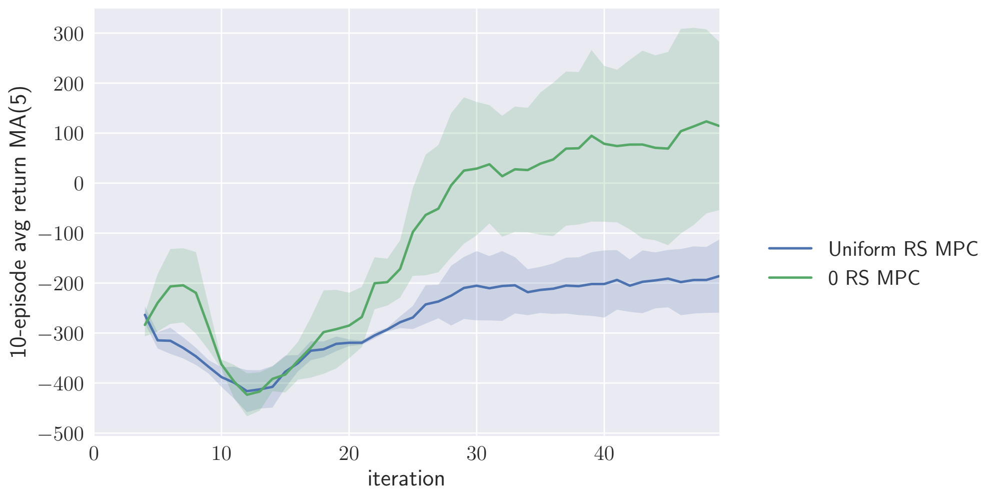

Figure 11: Ant reward for Uniform RS MPC vs 0 RS MPC with $K = 3000$ and $H = 30$.

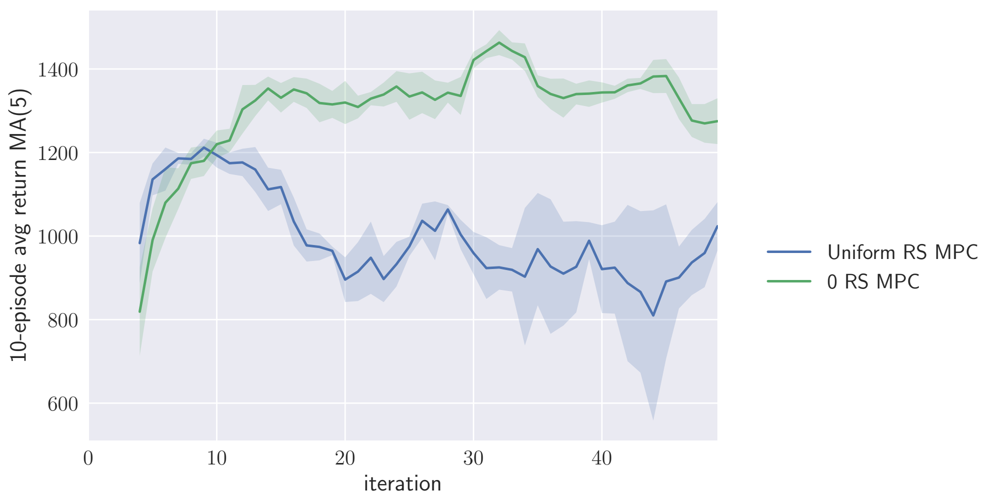

Figure 12: Walker2d reward for Uniform RS MPC vs 0 RS MPC with $K = 1000$ and $H = 15$.

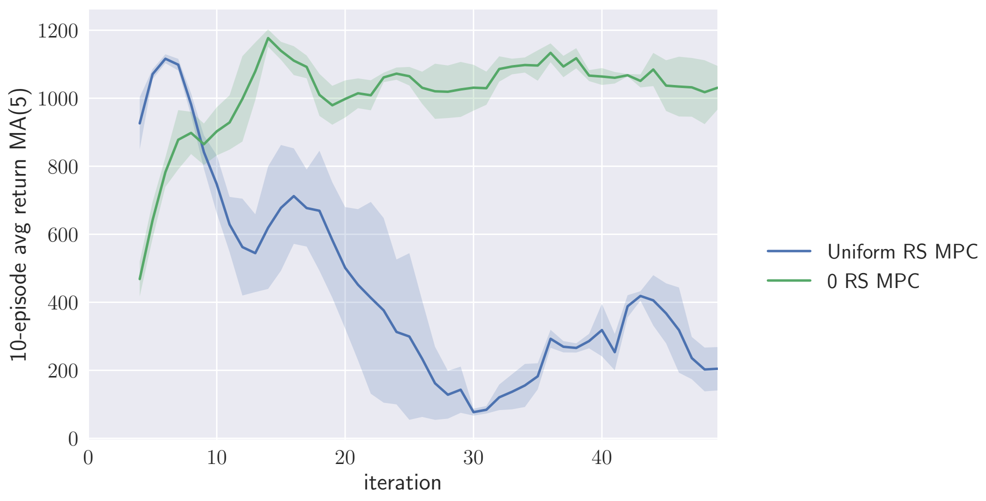

Figure 13: Walker2d reward for Uniform RS MPC vs 0 RS MPC with $K = 3000$ and $H = 30$.

9 Conclusion

We have provided evidence for the hypothesis that one needs to use constrained MPC per Equation 2 when using learned dynamics. The explicit relationship between over-optimization and over-optimism has been shown for one environment, and a generic but crude regularization strategy has shown promise in improving MPC performance.

9.1 Future Work

We believe that our work provides sufficient evidence to explore several promising directions.

First, we propose fusing the model-based with model-free approach, by virtue of using an off-policy model-free method to train π_θ with an appropriate choice of loss J in Algorithm 3. Centering MPC around π_θ may lead to performance improvements over the model-free method. However, a more valuable contribution would be to accelerate learning. In particular, by generating more performant episodes thanks to MPC, the off-policy algorithm may exhibit more stable or accelerated convergence properties. Early experiments show that this may be viable, but ϵ_t, ϵ_t' from Equation 2 may require additional nuance over the course of training. Note using a model-free approach here requires that the resulting trajectories are not too distant from those of the off-policy method: if the model-free method's value functions are not accurate on the support of the MPC agent's trajectories, then it will not learn from the off-policy data.

Second, we should move from using the RandomShooter for optimization to an explicit colocation method directly solving Equation 2 with dual gradient ascent. This should reduce variance in the MPC runs by consistently finding a good estimator. Note that the MPC solution from a previous timestep can be re-used to warm-start the next MPC solution, making the method tractable.

Third, we need to have a more nuanced set of constraints than proximity to the mean. The phenomenon we are interesting in capturing is dynamics accuracy, so one approach might be to directly retrieve dynamics variance. Then we can solve the an MPC objective while constraining ourselves to likely trajectories Equation 4.

$$\begin{aligned} \max_{\{a_t, s_t\}_{t=1}^H} \quad & \sum_{t=1}^H r(s_t, a_t) \\ \text{s.t.} \quad & s_{t+1} = \mathbb{E}\, f(s_t, a_t) \\ & \sum_t \operatorname{var} f(s_t, a_t) \le \epsilon \end{aligned} \tag{4}$$

We can approximately solve a soft version of the above by fixing Lagrange multipliers and treating the dynamics neural network model f as an approximate variational Gaussian Process, so that approximate variance can be computed using dropout samples [1].

References

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

[2] Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. Continuous deep q-learning with model-based acceleration. In *International Conference on Machine Learning*, pages 2829–2838, 2016.

[3] Nicolas Heess, Gregory Wayne, David Silver, Tim Lillicrap, Tom Erez, and Yuval Tassa. Learning continuous control policies by stochastic value gradients. In *Advances in Neural Information Processing Systems*, pages 2944–2952, 2015.

[4] Sergey Levine and Pieter Abbeel. Learning neural network policies with guided policy search under unknown dynamics. In *Advances in Neural Information Processing Systems*, pages 1071–1079, 2014.