Bootstrapped MPC

CS 294-112

Vladimir Feinberg, Samvit Jain, Michael Whittaker

10 October 2017

1 Introduction

We changed our research direction from the encoded states research to exploring model predictive control (MPC), namely a method we call bootstrapped MPC (BMPC). BMPC is a model-based algorithm that can be viewed as instantiation of the meta-MPC algorithm (Algorithm 1), which also explains the motivation for bootstrapped MPC (Algorithm 2).

Note we assume continuous domains for actions and states, named $\mathcal{A}, \mathcal{S}$, respectively. The continuous action space $\mathcal{A}$ is a meaningful assumption: MetaMPC does not take advantage of $\mathcal{A}$'s size. We also restrict discussion to deterministic learned dynamics and functionally specified rewards for simplicity (though extensions here to non-stationary or stochastic dynamics and rewards would not require much modification).

Algorithm 1 The MetaMPC algorithm is a template for MPC-based algorithms. As hyperparameters, it accepts a number of simulations to perform K and the simulation horizon H. The critical hyperparameter of interest is the possibly time-dependent action-sampling distribution A_t. Altogether, MetaMPC is a stochastic policy available only in this generative form, returning an action a for a provided state s.

```
1: procedure MetaMPC(state $s$, dynamics $f : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$, reward $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$)
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 MetaMPC works well when R_i is a decent estimator for the true H-step reward of the agent under the MetaMPC policy, supposing the agent takes $a_1^{(i)}$ now and continues with MetaMPC later. Then, choosing the first action $a_1^{(i_*)}$ that results in the best R_i would be the smartest move one could make. Regular MPC specifies $A_t(s) = \text{Uniform}\,\mathcal{A}$ for a rectangle $\mathcal{A}$, independent of t, s. But a uniform distribution is a poor representation of future MetaMPC steps: it stands to reason that a more intelligent selection of A_t might improve performance.

In particular, we consider learning a (mostly) stationary action distribution conditional on the state, $\pi_\theta(s)$, which is trained to mimic the result of the MPC algorithm. In particular, every iteration, we roll out MetaMPC with $A_t = \pi_\theta^{(t)}$ (with θ held fixed during both the real and simulated rollouts). After collecting some rollouts, resulting in a large dataset of state-action pairs $(s, a, s') \in \mathcal{D}$, we train $\pi_\theta^{(t)}$ to replicate MetaMPC's behavior with the expectation that the reward estimates improve (Algorithm 2).

Algorithm 2 The BMPC algorithm, with the same hyperparameters as Algorithm 1. 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 distribution loss J.

```
1: procedure BMPC(iterations $N$, starter transition dataset $\mathcal{D}$, reward $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$)
2:     for $i \leftarrow 1, \ldots, N$ do
3:         $f \leftarrow \operatorname{argmin}_F \mathbb{E}\left[\|f(s,a) - s'\|^2\right]$, where $(s,a,s') \sim \text{Uniform}\,\mathcal{D}$
4:         $\theta \leftarrow \operatorname{argmin}_\theta J(\theta, \mathcal{D}, \ell)$
5:         sample rollouts $\mathcal{D}'$ of MetaMPC$(\cdot, f, r)$ with $A_t$ specified by $\pi_\theta$
6:         $\mathcal{D} \leftarrow \mathcal{D}' \cup \mathcal{D}$
7:     return the agent MetaMPC$(\cdot, f, r)$ with $A_t$ specified by $\pi_\theta$.
```

In Algorithm 2, the loss J will usually be defined by some measure of distributional distance ℓ between the joint state-action distribution (Equation 1).

$$\begin{aligned} p_{\pi_\theta}^{(\mathcal{D})}(s,a) &= p_S(s)\pi_\theta(a|s) \qquad S \sim \text{Uniform}\,\{s \,|\, (s,-,=) \in \mathcal{D}\} \\ p_{\mathcal{D}}(s,a) &= p(S,A) \qquad S, A \sim \text{Uniform}\,\{(s,a) \,|\, (s,a,-) \in \mathcal{D}\} \\ J(\theta, \mathcal{D}, \ell) &= \ell\left(p_{\pi_\theta}^{(\mathcal{D})}, p_{\mathcal{D}}\right) \end{aligned} \tag{1}$$

We can motivate the BMPC approach mainly with the claim that the learner policy π used to take actions during MetaMPC simulations is closer to the true actions the BMPC agent would take at the simulated state. Then, BMPC might be bootstrapping performance because as π_θ gets more accurate in its representation of the corresponding MetaMPC policy, the MetaMPC procedure is able to take better actions, which are then learned by π, resulting in a virtuous cycle. Also, learner policy π might be taking actions which are better modelled by the dynamics f, since f is trained on BMPC transitions and is more accurate at predicting transitions by policies similar to BMPC agents.

2 Related Work

To some degree, this approach is similar to other continuous-action methods that seek to optimize the reward function directly, since in effect we are proposing a method for stochastic optimization of the rollout reward. One difference than, say, CMA-ES, would be that we explicitly model the reinforcement learning setting by modelling the BMPC agent as a stationary policy.

To some degree this approach is similar to Alpha-Go style sampling, where instead of replicating agent behavior the goal is to have intelligent MCTS (Monte Carlo Tree Search) pruning for a UCT (Upper Confidence Tree). If UCT pruning is guided by a neural network, then one might view UCT as a tree version of this BMPC approach (where the space for exploration, being discrete, is much more tractable to explore in the fairly exhaustive tree manner when using pruning). We might imagine this continuous case as a dual to pruning, where the learner π_θ proposes samples instead.

3 Easy Examples

We consider learning the dynamics for the `HalfCheetah-v1` environment and using those learned dynamics for BMPC planning. We note two important difference from the regular `gym` environment: the full observation is provided so the dynamics and rewards are true functions of the previous state (the usual observation excerpts the first coordinate of the agent).

Unless otherwise specified, all parameters are their defaults: 3 different seeds for each setup, 60 epochs of dynamics training on the full dataset per on-policy iteration, a dynamics batch size of 512, maximum episode length of 1000, $K = 1000$ simulated paths in the MPC controller (each of which has a simulated horizon $H = 15$), and 10 paths sampled by the initial random agent and then by the MPC controller that are aggregated every policy iteration. The learning rate was the default 10^{-3}. Each such policy iteration contains a round of training the learner policy π_θ in BMPC. All policies π_θ contain neural networks g_θ with $\dim \mathcal{A}$ outputs and $\dim \mathcal{S}$ inputs, all depth 5, width 32, trained with 100 epochs after each policy iteration with 512-size minibatches and a 10^{-3} learning rate. We didn't tune these parameters.

We consider a couple variants of BMPC, which are all defined by the conditional action distribution A_t. While our conditional distributions are mostly stationary, we consider exploring at the first step $t = 1$. Equation 2 describes the δ-BMPC, with a point mass describing the deterministic action in the MetaMPC simulation taken by A_t when $t > 1$. δ-BMPC is trained on minimizing the expected loss ℓ, which is the MSE from the agent's previous actions. Recall the distributions of S, A from Equation 1.

$$A_t(s) = \begin{cases} \text{Uniform}\left(\mathcal{A}\right) & t = 1 \\ \pi_\theta(s) \triangleq \delta_{g_\theta(s)} & \text{otherwise} \end{cases} \qquad \ell\left(p_{\pi_\theta}^{(\mathcal{D})}, p_{\mathcal{D}}\right) = \mathbb{E}\left[\left\|g_\theta(S) - A\right\|^2\right] \tag{2}$$

Gaussian BMPC relaxes the determinism of δ-BMPC, fitting a conditional diagonal Gaussian to the MetaMPC actions, as described in Equation 3.

$$A_t(s) = \begin{cases} \text{Uniform}\left(\mathcal{A}\right) & t = 1 \\ \pi_{\theta,\boldsymbol{\sigma}}(s) \triangleq N\left(g_\theta(s), \operatorname{diag}\boldsymbol{\sigma}^2\right) & \text{otherwise} \end{cases} \qquad \ell\left(p_{\pi_\theta}^{(\mathcal{D})}, p_{\mathcal{D}}\right) = \mathbb{E}\left[\log \pi_{\theta,\boldsymbol{\sigma}}(A|S)\right] \tag{3}$$

No-explore BMPC reduces the exploration by keeping the policy purely conditional diagonal Gaussian and stationary, with no initial exploration, as described in Equation 4.

$$A_t(s) = \pi_\theta(s) = N\left(g_\theta(s), \operatorname{diag}\boldsymbol{\sigma}^2\right) \qquad \ell\left(p_{\pi_\theta}^{(\mathcal{D})}, p_{\mathcal{D}}\right) = \mathbb{E}\left[\log \pi_{\theta,\boldsymbol{\sigma}}(A|S)\right] \tag{4}$$

We evaluate these examples with an easy, supervised reward for `HalfCheetah-v1` which was provided in HW4 (Figure 1). Note that this is different than the classical `gym` reward.

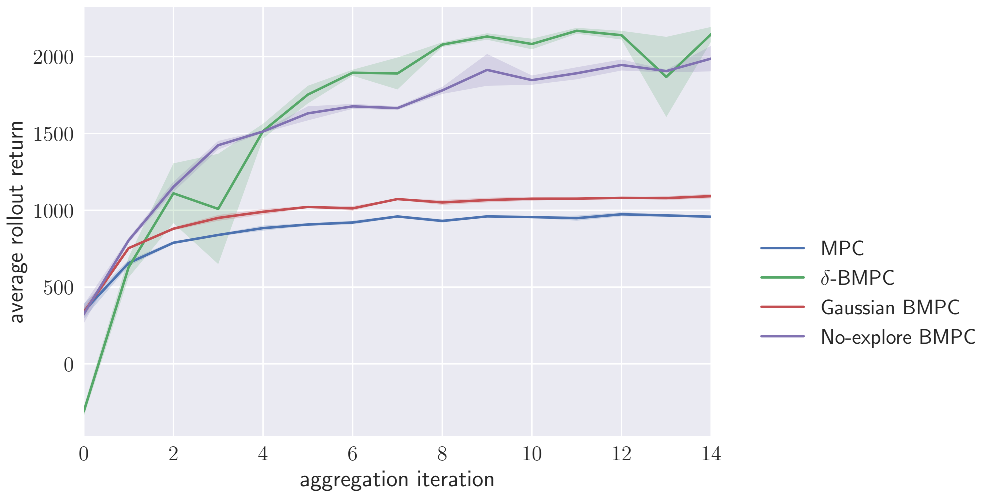

Figure 1: Average return over policy aggregation iterations for the learning-based sampling MPC agent, compared to the standard uniform sampling MPC agent.

In Figure 1, we see a couple interesting things going on already. Amazingly, the improvement above happens even though the learners themselves are not near the MPC performance at any iteration (Figure 2).

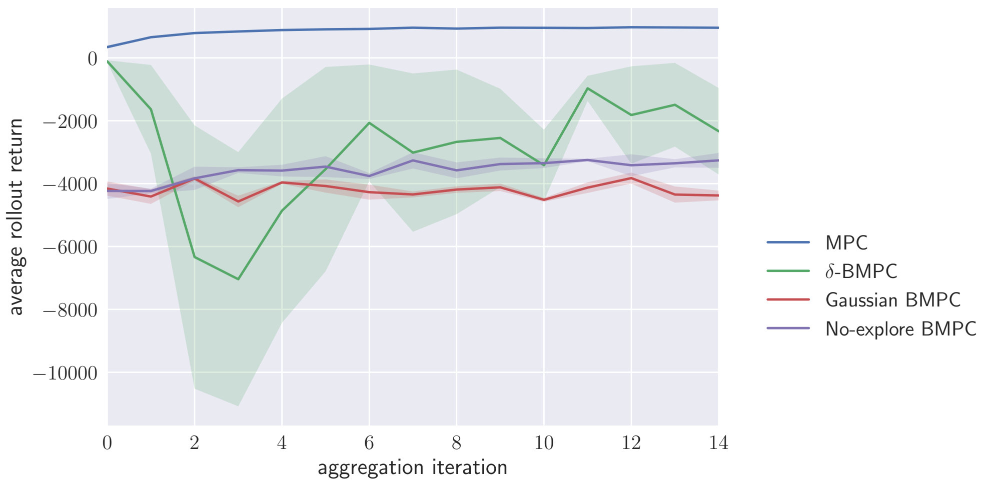

Figure 2: Average return over policy aggregation iterations for an agent sampling from A_2 for each model, along with the original MPC as a baseline. A random agent gets about -5000 return here, so the learned policies are only slight improvements relative to that.

The poor learner performance does not kill the original hypothesis: perhaps even weak learners can improve reward estimates R_{i_*}. But this does suggest an new hypothesis for the improvement: perhaps the learners prevent catastrophic changes in the policy. Maybe simply reducing the variance of estimates R_{i_*} yields the improvement. We note that, at least for this simple case, learners don't improve performance by making the dynamics more learnable: the learned dynamics are actually less performant for the learners (Figure 3).

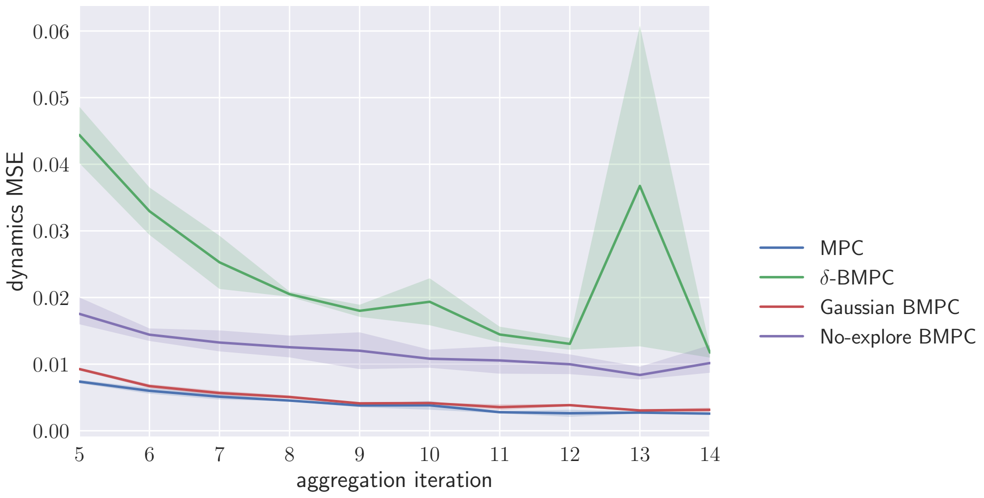

Figure 3: We evaluate dynamics in the setting of Figure 1. This evaluation is measures how predictive the dynamics were of the actual transitions in validation rollouts (it is not the training MSE). We omit the first couple of iterations because their scales are vastly different than the rest.

4 Hard Cost

If we force ourselves to only rely on the usual `HalfCheetah-v1` reward, which only rewards forward movement and discourages large-magnitude actions, we are faced with a more difficult setting.

In this case, it is unclear if bootstrapping still helps (Figure 4).

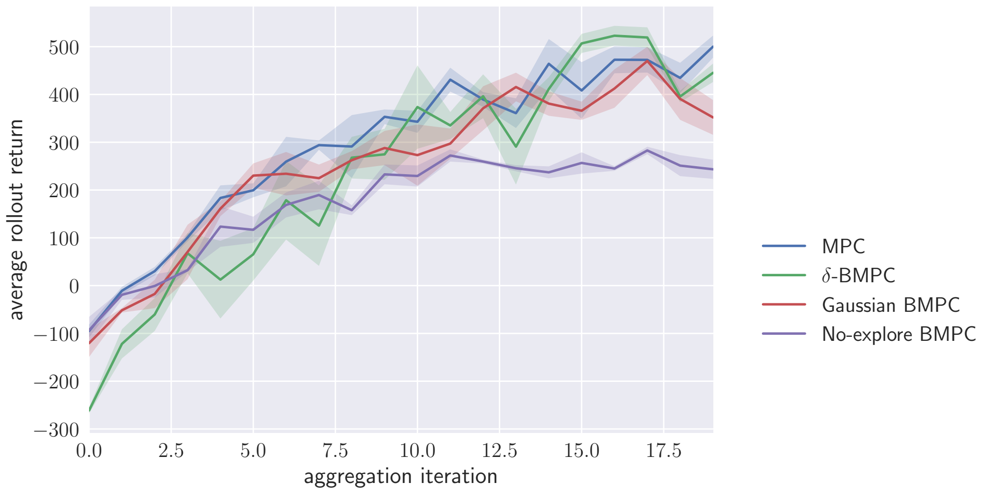

Figure 4: Average return over policy aggregation iterations for the learning-based sampling MPC agent, compared to the standard uniform sampling MPC agent, this time with a hard cost function.

The dynamics and learning returns for the runs in Figure 4 paint a similar picture as for easy cost. In case hard cost required additional simulation, we changed H, K from $15, 1000$ to $50, 1000$. Performance was generally better but more chaotic, and still with no clear advantage to BMPC (Figure 5).

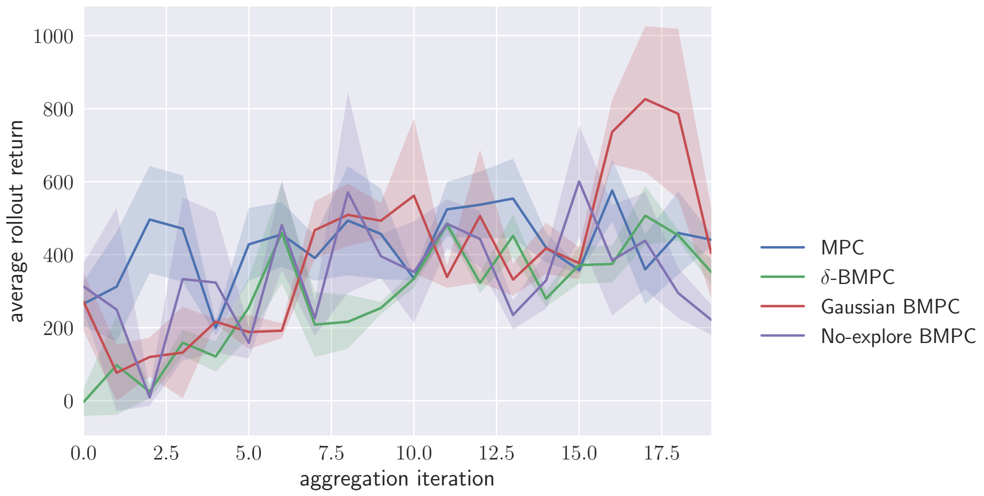

Figure 5: The same setting as Figure 5 but with longer simulated horizons.

Nonetheless, the trend in Figure 4 deserves elaboration—perhaps at later iterations BMPC overtakes MPC.

5 Planned Work

The proposition being investigated (more accurate or lower-variance estimates of reward with R_{i_*}) can be directly sampled and tested to see if it is indeed the case! After simulating several rollouts for any MetaMPC algorithm, we can sample initial state s from the empirical distribution of visited states. For each such state, we *re-simulate* H real horizon steps with independent runs (the samples A_t will differ, so the results will not be the same as the first run, even for a deterministic environment). These give us several observations of the true H-step reward $\tilde{R}_H$. For each such observation we compute the observed bias $\tilde{R}_H - R_{i_*}$ and squared error $\left(\tilde{R}_H - R_{i_*}\right)^2$. Over several samples we can estimate variance var R_{i_*} as well. Then across the samples s we can estimate average values across the entire distribution for these reward statistics. This sampling of s is not iid but the resulting estimator is still unbiased, so we will have to use the bootstrap (haha) to calculate confidence intervals.

The loss ℓ need not be replication of the expert directly: we can perhaps optimize for learner policies π_θ which re-create the observed reward directly. If the goal is to have accurate reward predictions during simulations, then we should also consider learner π_θ objectives J *that directly optimize* $\left(\tilde{R}_H - R_{i_*}\right)^2$ by back-propagating through the model f.

Next, we note that above we use tractable, analytically representable policies π_θ. There is no need for this. We should measure how well π_θ optimizes the objective J, and, if necessary, expand the class of distributions being optimized over to better capture MetaMPC actions (starting with mixtures of Gaussians and going up to GANs, since we only need generative access to π_θ).

Finally, it may be valuable to measuring the explicit action distribution shift between iterations (to see if adding learners decreases the shift). This boils down to measuring distance between conditional action distributions resulting from the MetaMPC algorithm, which is only a black-box generative distribution. We have a plan for this but excerpt it to keep the report shorter. Graders can contact the authors for more details.

In addition, we hope to expand to the harder `Ant` problem to prevent overfitting to `HalfCheetah`.

Finally, the chaotic performance but improved performance in the second hard setting suggest that we should explore various hyperparameter and template settings; but we should save this for the end.

6 Further Exploration

We also explored changing the initial exploration policy A_1 from Uniform($\mathcal{A}$) to $N(g_\theta, \lambda I)$. This resulted in some improvements, but we leave these out of the report for simplicity (graders may contact the authors directly for these results). Further, to improve how quickly π_θ learns expert actions, we consider DAgger iteration with π_θ as the learner and MetaMPC as the expert, but managing the learned dynamics for both the learner and expert will require additional work.