Model-based Policy Gradients with Entropy Exploration through Sampling

Samuel Stanton [1] Ke Alexander Wang [2] Andrew Gordon Wilson [1]

Abstract

Training a deterministic policy with a model-based reinforcement learning algorithm is a delicate procedure. While deterministic policies are very data-efficient, they are not nearly as forgiving of model error and gradient noise as stochastic policies. In practice this makes algorithms like PILCO very time-consuming to implement, with brittle dependence on design choices like task specific initialization schemes and policy architecture choice. In this work we present two practical improvements that make PILCO significantly more robust and easier to implement. We relax the moment-matching approximation and instead rely only on Monte Carlo reward estimates, and propose a model entropy regularization term that improves the reliabilty and data-efficiency of the algorithm while replicating the exceptional data-efficiency of PILCO.

1. INTRODUCTION

There have been many papers that report impressive data efficiency on tasks ranging from automatic control (Deisenroth et al., 2015; Kurutach et al., 2018) to video games (Ha and Schmidhuber, 2018). However the aspiring researcher who undertakes a reimplementation or refinement of one of these methods soon finds that even reproducing reported results is no easy task. Because of online interactions between the model and the policy during the course of learning, successful model-based algorithms rely on the emergence of positive feedback loops as training progresses. Ideally, as the policy improves new data is collected that improves the model's predictive power, which further improves the policy. Unfortunately these feedback loops can be extremely fragile, causing the most minor of implementation choices to have outsize effects on the overall success of the algorithm. The result is a collection of brittle methods that require an exceptional amount of time and effort to implement. In this work we will focus on some practical improvements to PILCO (Deisenroth and Rasmussen, 2011), a probabilistic model-based RL algorithm. Our contributions focus on modifications to the model inference procedure by transitioning from deterministic or stochastic Gaussianity assumptions on the marginal state distributions over time, and instead rely solely on trajectory sampling. To improve the stability of the algorithm to small design changes, we propose a modified loss function with a model entropy regularization term.

[1]Department of Operations Research and Information Engineering, Cornell University, Ithaca, NY, U.S.A. [2]Department of Computer Science, Cornell University, Ithaca, NY, U.S.A.. Correspondence to: Samuel Stanton <ss3765@cornell.edu>, Ke Alexander Wang <kaw293@cornell.edu>.

Proceedings of the 36^{th} International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019. Copyright 2019 by the author(s).

2. RELATED WORK

Sampling is a natural approach to probabilistic model inference, and there is a good body of literature on the subject of model-based RL with trajectory sampling. Algorithms like Stochastic Value Gradients (SVG, Heess et al. (2015)) and Model-Ensemble Trust-Region Policy Optimization (Kurutach et al., 2018) draw a single trajectory sample while estimating policy gradients. Though stochastic policies are more forgiving of gradient noise and model error, it comes at the expense of data-efficiency, which truly is a primary consideration for real-world applications. Our method is related to intrinsic motivation for model-free algorithms, explored in Bellemare et al. (2016); Burda et al. (2018).

PILCO was extended to use probabilistic ensembles of neural networks in Gal et al. (2016). This work relied on trajectory sampling for model inference, however a stochastic version of the moment-matching approximation, Gaussian resampling, was necessary to acheive good performance on the cartpole problem. In a follow-up paper Parmas et al. (2019) attempted to exactly replicate PILCO with the exception of Monte Carlo expected cost estimates. They argued that gradient estimate variance reduction techniques were necessary to discard the moment-matching approximation. This variance was attributed to inherent chaotic behavior of the backpropagation of reparameterization gradients through many timesteps. Chua et al. (2018) proposed an algorithm (PETS) for model predictive control (MPC) with probabilistic neural network ensembles. While they used trajectory sampling to form reward estimates, they did not estimate gradients, but instead relied on guided random search to find good action sequences.

3. BACKGROUND

3.1. General Setting

Consider a discrete-time dynamic system with a state-space $\mathcal{S} \subset \mathbb{R}^D$ and an agent with a finite time horizon T and action-space $\mathcal{A} \subset \mathbb{R}^F$. The agent's actions are governed by a deterministic policy $\pi : \mathcal{S} \to \mathcal{A}$, parameterized by θ, so for each $t \in [T]$, $\mathbf{a}^{(t)} = \pi(\mathbf{s}^{(t)}|\theta)$. The action taken induces a change in the system state according to an unknown transition function f. In particular, we say $\mathbf{s}_{t+1} = f(\mathbf{s}^{(t)}, \mathbf{a}^{(t)}) + \mathbf{w}$, where $\mathbf{w} \sim \mathcal{N}(0, \Sigma)$. Suppose we have an initial state distribution $p(\mathbf{s}^{(0)})$, and a reward function $r : \mathcal{S} \to \mathbb{R}$. We would like to find the optimal policy $\pi(\mathbf{s}|\theta^*)$, where

$$\theta^* = \arg\max_\theta \sum_{t=1}^{T} \mathbb{E}[r(\mathbf{s}^{(t)})|\theta]$$

For example, consider the cartpole system consisting of an inverted pendulum on a cart. The state is completely determined by $\begin{bmatrix} x & \dot{x} & \dot{\phi} & \cos\phi & \sin\phi \end{bmatrix}$, where x is the horizontal position of the cart, and ϕ is the angle of the pendulum off of the upright vertical, clockwise. The action is the application of a horizontal force $u \in [-1, 1]$ to the cart. Hence $\mathcal{S} \subset \mathbb{R}^5$, $\mathcal{A} \subset \mathbb{R}$. In the swingup task, the pole begins hanging down and we wish to learn a policy that moves the cart in a way that swings the pole up and balances it. As an example, a simple policy parameterization would be a linear controller, $u = A\mathbf{s} + b$ (so $\theta = \{A, b\}$). One choice of reward function for this task is

$$r(\mathbf{s}) = \exp\{-\frac{1}{2}(\mathbf{s} - \mathbf{s}^*)^\top Q(\mathbf{s} - \mathbf{s}^*)\} \tag{1}$$

where $\mathbf{s}^*$ is the *target* state, $\begin{bmatrix} 0 & 0 & 0 & 1 & 0 \end{bmatrix}$, and Q is a matrix of lengthscale hyperparameters.

3.2. Gaussian Process Dynamics Models

For a thorough treatment of Gaussian processes, see Rasmussen and Williams (2006). Briefly, a GP is defined by a mean function $\eta(\cdot)$ and kernel function $k(\cdot, \cdot)$. Let $\mathbf{x} = (\mathbf{s}, \mathbf{a})$, and $\mathbf{\Delta}_t = \mathbf{s}_{t+1} - \mathbf{s}_t$. We use the GP to model the change to the state of the environment as a function of the current state and our policy action, that is

$$\hat{\mathbf{s}}_{t+1} = \hat{f}(\mathbf{x}_t) = \mathbf{s}_t + \mathbf{\Delta}_t, \quad \mathbf{\Delta}_t \sim p(\mathbf{\Delta}_t|\mathbf{x}_t)$$

Modeling the changes between the states, rather than the next state directly, allows us to use have a constant prior mean of 0. For a test input $\mathbf{x}^*$, given training data $\mathcal{D} = (X, \mathbf{y})$, we have the predictive distribution

$$p(\mathbf{\Delta}^* \mid \mathbf{x}^*, \mathcal{D}) = \mathcal{N}(m(\mathbf{x}^*), \Sigma(\mathbf{x}^*)),$$

where the mean and covariance functions m, Σ are defined as

$$m(\mathbf{x}^*) = (m_1(\mathbf{x}^*), \ldots, m_D(\mathbf{x}^*)) \tag{2}$$
$$m_d = k_d(\mathbf{x}^*, X)(K_d + \sigma_d^2 I)^{-1} y_d \tag{3}$$
$$\Sigma(\mathbf{x}^*) = \text{diag}(s_1^2(\mathbf{x}^*), \ldots, s_D^2(\mathbf{x}^*)) \tag{4}$$
$$s_d^2 = \sigma_{f_d}^2 - k_d(\mathbf{x}^*, X)(K_d + \sigma_d^2 I)^{-1} k_d(X, \mathbf{x}^*) \tag{5}$$

Here we use the subscript d to denote the fact that we have independent GPs with distinct kernels and hyperparameters for each state dimension $d \in [D]$. Hence $K_d = k_d(X, X)$, σ_d^2 is the noise variance hyperparameter in GP regression, $\sigma_{f_d}^2$. If each k_d is an ARD kernel, then they have $D + F$ lengthscale hyperparameters. All hyperparameters are trained by maximizing the marginal log-likelihood of the training data (also known as evidence maximization).

3.3. Model-based Reinforcement Learning

Model-based RL algorithms estimate cumulative reward by learning the underlying transition dynamics and predicting possible trajectories. Model-based RL algorithms have a major challenge. In the early stages of learning the model has very little data, and cannot be expected to accurately predict the trajectories of the policies it is supposed to evaluate. If the model does not accurately quantify the large amount of uncertainty in its prediction, it will heavily bias the policy search. Gaussian processes (GPs) are a class of machine learning models particularly suited to the task of prediction with accompanying uncertainty estimates in the form of predictive variances. Deisenroth et al. (2015) applied GP regression models to the model-based RL setting with the PILCO algorithm (Alg. 1).

PILCO evaluates policies with *rollouts*, serial predictions of the marginal state distributions $p(\mathbf{s}_t|\theta)$ for each $t \in [T]$. For notational simplicity from here we will omit the conditioning of the marginals on θ. Given $p(\mathbf{s}_1), \ldots, p(\mathbf{s}_T)$, the predicted cumulative reward is given by

$$J(\theta) = \sum_{t=1}^{T} \mathbb{E}_{p(\mathbf{s}_t)}[r(\mathbf{s}_t)] \tag{6}$$

Starting with $p(\mathbf{s}_0)$, a central challenge is to infer the predicted marginal density of $\mathbf{s}_{t+1} = \mathbf{s}_t + \mathbf{\Delta}_t$ for each t. $p(\mathbf{s}_t)$ and π induce a distribution $p(\mathbf{x}_t)$ on the inputs to the GP, so the distribution of $\mathbf{\Delta}_t$ is given by

$$p(\mathbf{\Delta}_t) = \int p(\mathbf{\Delta}_t|\mathbf{x}_t) p(\mathbf{x}_t) d\mathbf{x}_t \tag{7}$$

Since this integral does not have a closed analytic form, PILCO approximates it with a moment-matched Gaussian.

$$p(\mathbf{\Delta}_t) \approx \mathcal{N}(\mu_{\Delta_t}, \Sigma_{\Delta_t}) \quad (8)$$
$$\mu_{\Delta_t} = \mathbb{E}[\mathbf{\Delta}_t] \quad (9)$$
$$\Sigma_{\Delta_t} = \text{covar}(\mathbf{\Delta}_t, \mathbf{\Delta}_t) \quad (10)$$

If $p(\mathbf{s}_0)$ is Gaussian, then for $t \in [T]$, $\mathbf{s}_{t+1}$ is approximately a sum of Gaussian random variables, and so itself has a Gaussian marginal density with moments

$$\mu_{t+1} = \mu_t + \mu_{\Delta_t}$$
$$\Sigma_{t+1} = \Sigma_t + \Sigma_{\Delta_t} + \text{covar}(\mathbf{s}_t, \mathbf{\Delta}_t) + \text{covar}(\mathbf{\Delta}_t, \mathbf{s}_t)$$

Algorithm 1 PILCO

```
Input: Gaussian $p(\mathbf{s}_0)$, initial policy $\pi(\cdot|\theta)$
Initialize $\mathcal{D}$ by applying a random actions to environment.
for episode $= 1, \dots, M$ do
    Train GP dynamics model with all past data $\mathcal{D}$.
    Gradient-based optimization of $\theta$ w.r.t. $J(\theta)$ (Eq. 6).
    Apply $\pi(\cdot|\theta)$ to environment, add observations to $\mathcal{D}$.
end for
Output: $\pi(\cdot|\theta)$
```

4. METHOD

PILCO exploited the fact that reward functions of the form in Eq. 1 have closed form expectations under Gaussian approximations to compute analytic policy gradients for optimization. We can relax the moment-matching assumption by simply using Monte Carlo estimates of Eq. 6. This approach was previously attempted by McHutchon (2015), but his approach required fixing the simulator seed to obtain deterministic policy gradients which is only possible in simulated environments. By using automatic differentiation and the reparameterization trick, we can bypass this limitation and relax the moment-matching assumption in PILCO.

We denote a trajectory as $\tau = (\mathbf{s}_0, \mathbf{a}_0, \dots, \mathbf{a}_{T-1}, \mathbf{s}_T)$. For Markovian systems sampling from the $p(\tau)$ is straightforward through ancestral sampling. Starting with a known $p(\mathbf{s}_0)$, an initial population of samples, or particles, $\{\mathbf{s}_0^i\}_{i=1}^N \sim p(\mathbf{s}_0)$ is drawn. Then for each $t \in [T]$, $i \in [N]$,

$$\mathbf{a}_{t-1}^i \sim \pi(\mathbf{a}_{t-1}|\mathbf{s}_{t-1}^i)$$
$$\mathbf{s}_t^i \sim p(\mathbf{s}_t|\mathbf{s}_{t-1}^i, \mathbf{a}_{t-1}^i)$$

and we have

$$J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T} r(\mathbf{s}_t^i). \quad (11)$$

Note that for deterministic policies $\pi(\mathbf{a}_t|\mathbf{s}_t)$ is a pointmass at $\pi(\mathbf{s}_t;\theta)$.

4.1. Incentivizing Exploration

One of the main motivating factors behind choosing saturating rewards of the form in Eq. 1 is that they sometimes interact with model uncertainty to automatically balance the explore-exploit tradeoff (Deisenroth et al., 2015). However in our experience relying exclusively on this interaction tends to cause the algorithm's behavior to be rather unpredictable. The exceptional reliability and data efficiency of the original PILCO implementation appears to be a consequence, at least in part, of a great deal of hand-tuning. From the perspective of optimization, a saturating cost causes a majority of the loss surface to be almost flat, often resulting in vanishingly small policy gradients. Providing the agent with auxilary reward terms can greatly improve the reliability and overall performance of the algorithm. This practice is commonly employed in model-free methods such as those in Schulman et al. (2017), where the entropy of the stochastic policy is typically added to the loss to encourage exploration. Since we are training deterministic policies we consider two possible analogues. First one could simply use the sample variance of the model expected reward,

$$\frac{1}{N-1}\sum_{i=1}^{N}\left(\sum_t r(\mathbf{s}_t^i) - \mu_r\right)^2 \quad (12)$$

where μ_r is given by Eq. 11. However this term can only encourage exploration of regions of the state space with non-zero reward. Because of the choice of a saturating reward function this will have no effect in a large region of the state space, in particular regions the agent is likely to be initially exploring. An alternative choice that avoids this pitfall is the model entropy,

$$H(\theta) = \frac{1}{N}\sum_{i,t} h[\mathbf{s}_t^i, \pi(\mathbf{s}_t^i;\theta)] \quad (13)$$

Here $h[\mathbf{s}_t^i, \pi(\mathbf{s}_t^i;\theta)]$ is the differential entropy of the predictive distribution $p(\mathbf{s}_{t+1}|\mathbf{s}_t^i, \pi(\mathbf{s}_t^i;\theta))$. Conditioned on $\mathbf{s}_t^i, \pi(\mathbf{s}_t^i;\theta)$ the predictive distributions for each dimension of s_{t+1} are independent Gaussians with variance $\hat{\sigma}_d^2$, so

$$h[\mathbf{s}_t^i, \pi(\mathbf{s}_t^i;\theta)] = \frac{1}{2}\sum_{d=1}^{D}\log((\hat{\sigma}_d^2)_t^i) + C$$

Adding this term to the reward encourages the agent to explore regions where the predictive variance is high. In a low-noise environment these correspond to a regions where the model has little training data.

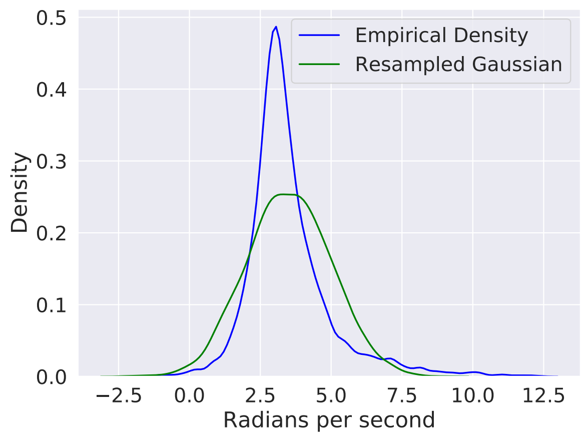

Figure 1. An example of the non-Gaussian marginal distribution of the pole velocity at $t = 30$ estimated using trajectory sampling and Gaussian resampling.

5. RESULTS

We implement our GP models as described in Section 3.2 with GPyTorch, a modern Gaussian Process library for PyTorch (Gardner et al., 2018). All experiments were run on desktop workstations with an Intel Core i7-8700K CPU and a single Nvidia GeForce GTX 1080 Ti GPU. We evaluated our method by learning controllers for the cartpole swingup task in the Deepmind Control Suite (Tassa et al., 2018). Figure 1 shows an example of a empirical state marginal distribution from trajectory sampling and its Gaussian resampled approximation for this task.

We followed the procedure outlined in Algorithm 1 and we matched the interaction time and sample frequency of the code provided in Deisenroth et al. (2015). Each interaction with the environment was set to 3 seconds with data sampled at 10Hz. Similarly, our rollouts were set to predict with a horizon of 3 seconds. All trials were initialized with a single episode of data generated from uniformly random actions. During each episode of learning, we retrained the Gaussian process dynamics model by maximizing the log marginal likelihood with 32 iterations of the Adam optimizer by Kingma and Ba (2015) with a learning rate of 0.1. We updated the policy using 256 iterations of policy optimization with Adam with a learning rate of 0.1, drawing $N = 256$ samples for both trajectory sampling and Gaussian resampling for rollouts. We used the regularized RBF network policy parameterization of Deisenroth et al. (2015) for $\pi(\cdot; \theta)$ with m support points and support targets. Figure 2 shows the results for $m = 50$.

In order to test the robustness of our method, we ran a seperate experiment with the same hyperparameters, but varied the number of support points, and hence the flexibility of the policy class. Specifically we tested trajectory sampling with and without an entropy bonus for $m = 10, 25, 50$. Using these settings, trajectory sampling alone solved the task $0/3, 2/3$, and $4/6$ of the time. On the other hand, trajectory sampling with entropy bonus solve it $2/3, 3/3$, and $6/6$ of the time.

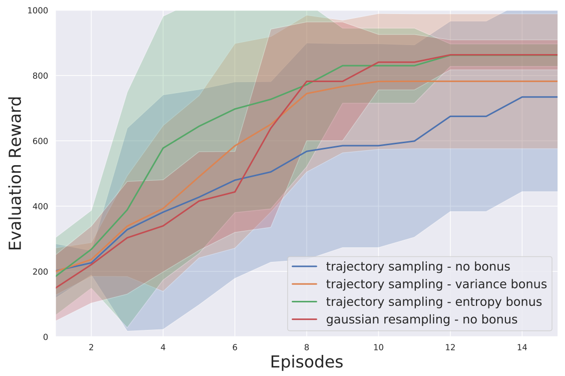

Figure 2. Shown above are the reward curves for each method evaluated on the Deepmind Control Suite cartpole swingup task. The task is considered completely solved after reaching a reward of 800. The curves are averaged over 6 trials. The shaded region indicates a 90% credible set. Though vanilla trajectory sampling (blue) does learn reasonable policies, its behavior is highly variable, and does not match the overall performance of our implementation of PILCO (red). The addition of auxilary exploration terms to the reward function completely overcome this shortcoming, actually improving the overall data-efficiency. On average, trajectory sampling with an entropy bonus solved the problem with 18 sec of simulated data.

6. DISCUSSION

Our initial results on the cartpole swingup task are noteable because they demonstrate that it is indeed possible to train deterministic policies with Monte Carlo cost estimates using only reparameterization gradients and a modified loss function. We expect that our implementation would further benefit by incorporating contributions from the literature such as likelihood ratio gradients to avoid backpropagating through long predictions. This is a direction we intend to pursue. It should be noted that such a direction is complementary to the observations of this paper.

References

Bellemare, M., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., and Munos, R. (2016). Unifying count-based exploration and intrinsic motivation. In *Advances in Neural Information Processing Systems*, pages 1471–1479.

Burda, Y., Edwards, H., Pathak, D., Storkey, A., Darrell, T., and Efros, A. A. (2018). Large-scale study of curiosity-driven learning. *arXiv preprint arXiv:1808.04355*.

Chua, K., Calandra, R., McAllister, R., and Levine, S. (2018). Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems*, pages 4754–4765.

Deisenroth, M. and Rasmussen, C. E. (2011). Pilco: A model-based and data-efficient approach to policy search. In *ICML*, pages 465–472.

Deisenroth, M. P., Fox, D., and Rasmussen, C. E. (2015). Gaussian processes for data-efficient learning in robotics and control. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(2):408–423.

Gal, Y., McAllister, R., and Rasmussen, C. E. (2016). Improving pilco with bayesian neural network dynamics models. In *Data-Efficient Machine Learning workshop, ICML*, volume 4.

Gardner, J., Pleiss, G., Weinberger, K. Q., Bindel, D., and Wilson, A. G. (2018). Gpytorch: Blackbox matrix-matrix gaussian process inference with gpu acceleration. In *NeurIPS*, pages 7587–7597.

Ha, D. and Schmidhuber, J. (2018). Recurrent world models facilitate policy evolution. In *Advances in Neural Information Processing Systems*, pages 2450–2462.

Heess, N., Wayne, G., Silver, D., Lillicrap, T., Erez, T., and Tassa, Y. (2015). Learning continuous control policies by stochastic value gradients. In *Advances in Neural Information Processing Systems*, pages 2944–2952.

Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In *ICLR*.

Kurutach, T., Clavera, I., Duan, Y., Tamar, A., and Abbeel, P. (2018). Model-ensemble trust-region policy optimization. *arXiv preprint arXiv:1802.10592*.

McHutchon, A. J. (2015). *Nonlinear modelling and control using Gaussian processes*. PhD thesis, Citeseer.

Parmas, P., Rasmussen, C. E., Peters, J., and Doya, K. (2019). Pipps: Flexible model-based policy search robust to the curse of chaos. *arXiv preprint arXiv:1902.01240*.

Rasmussen, C. E. and Williams, C. K. (2006). *Gaussian processes for machine learning*, volume 1. MIT press Cambridge.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Tassa, Y., Doron, Y., Muldal, A., Erez, T., Li, Y., Casas, D. d. L., Budden, D., Abdolmaleki, A., Merel, J., Lefrancq, A., et al. (2018). Deepmind control suite. *arXiv preprint arXiv:1801.00690*.