Variational Model-based Policy Optimization

Yinlam Chow\textsuperscript{1}, Brandon Cui\textsuperscript{2}, MoonKyung Ryu\textsuperscript{1} and Mohammad Ghavamzadeh\textsuperscript{1}

\textsuperscript{1}Google AI
\textsuperscript{2}Facebook AI Research

yinlamchow@gmail.com, bcui@fb.com, \{mkryu, ghavamza\}@google.com

Abstract

Model-based reinforcement learning (RL) algorithms allow us to combine model-generated data with those collected from interaction with the real system in order to alleviate the data efficiency problem in RL. However, designing such algorithms is often challenging because the bias in simulated data may overshadow the ease of data generation. A potential solution to this challenge is to jointly learn and improve model and policy using a universal objective function. In this paper, we leverage the connection between RL and probabilistic inference, and formulate such an objective function as a variational lower-bound of a log-likelihood. This allows us to use expectation maximization (EM) and iteratively fix a baseline policy and learn a variational distribution, consisting of a model and a policy (E-step), followed by improving the baseline policy given the learned variational distribution (M-step). We propose model-based and model-free policy iteration (actor-critic) style algorithms for the E-step and show how the variational distribution learned by them can be used to optimize the M-step only from model-generated samples. It is important to note that both algorithms are model-based and only differ in using model-based and model-free algorithms for the E-step. We call our algorithm Variational Model-based Policy Optimization (VMBPO), which we refer to as variational model-based policy optimization (VMBPO), is more sample-efficient and robust to hyper-parameter tuning than its model-free (E-step) counterpart. Using the same control tasks, we also compare VMBPO with several state-of-the-art model-based and model-free RL algorithms and show its sample efficiency and performance.

1 Introduction

Model-free reinforcement learning (RL) algorithms that learn a good policy without constructing an explicit model of the system’s dynamics have shown promising results in complex simulated problems [Mnih et al., 2013; Mnih et al., 2015; Schulman et al., 2015; Haarnoja et al., 2018]. However, these methods are not sample efficient, and thus, not suitable for problems in which data collection is burdensome. Model-based RL algorithms address the data efficiency issue of the model-free methods by learning a model, and combining model-generated data with those collected from interaction with the real system [Sutton, 1990; Janner et al., 2019]. However, designing model-based RL algorithms is often challenging because the bias in model may affect the process of learning policies and result in worse asymptotic performance than the model-free counterparts. A potential solution to this challenge is to incorporate the policy/value optimization method in the process of learning the model. An ideal case here would be to have a universal objective function that is used to learn and improve model and policy jointly.

Casting RL as a probabilistic inference has a long history (e.g., [Todorov, 2008; Toussaint, 2009; Kappen et al., 2012; Rawlik et al., 2013]). This formulation has the advantage that allows powerful tools for approximate inference to be employed in RL. One such class of tools are variational techniques that have been successfully used in RL (e.g., [Neumann, 2011; Levine and Koltun, 2013; Abdolmaleki et al., 2018]). Another formulation of RL with strong connection to probabilistic inference is the formulation of policy search as an expectation maximization (EM) style algorithm (e.g., [Dayan and Hinton, 1997; Peters and Schaal, 2007; Peters et al., 2010; Chebotar et al., 2017; Abdolmaleki et al., 2018]). The main idea here is to write the expected return of a policy as a (pseudo)-likelihood function, and then conditioning on the success in maximizing the return, find the policy that most likely would have been taken. Another class of RL algorithms that are related to the inference formulation are entropy-regularized algorithms that add an entropy term to the reward and find the soft-max optimal policy (e.g., [Levine and Abbeel, 2014; Nachum et al., 2017; Haarnoja et al., 2018; Fellows et al., 2019]). For a comprehensive tutorial on RL as probabilistic inference, we refer readers to [Levine, 2018].

In this paper, we leverage the connection between RL and probabilistic inference, and formulate an objective function for jointly learning and improving model and policy as a variational lower-bound of a log-likelihood. This allows us to use EM: iteratively fix a baseline policy and learn a variational distribution, consisting of a model and a policy (E-step), followed by improving the baseline policy given the learned variational distribution (M-step). We propose model-based and model-free policy iteration (PI) style algorithms for the E-step and show how the variational distribution that they learn can be used to optimize the M-step only from model-generated samples. It is important to note that both algorithms are model-based and only differ in using model-based and model-free algorithms for the E-step. We call our algorithm
that uses model-based PI for the E-step, variational model-based policy optimization (VMBPO). Our experiments on a number of continuous control tasks show that VMBPO is more sample-efficient and robust to hyper-parameter tuning than its model-free counterpart. Using the same control tasks, we also compare VMBPO with several state-of-the-art model-based and model-free RL algorithms, including model-based policy optimization (MBPO) [Janner et al., 2019] and maximum a posteriori policy optimization (MPO) [Abdolmaleki et al., 2018], and show its sample efficiency and performance.

2 Preliminaries

We study the RL problem in which the agent’s interaction with the environment is modeled as a Markov decision process (MDP) $M = (X, A, r, p, p_0)$, where $X$ and $A$ are state and action spaces; $r : X \times A \rightarrow \mathbb{R}$ is the reward function; $p : X \times A \rightarrow \Delta_X$ is the transition kernel; and $p_0 : X \rightarrow \Delta_X$ is the initial state distribution. A stationary Markovian policy $\pi : X \rightarrow \Delta_X$ is a probabilistic mapping from states to actions. Each policy $\pi$ is evaluated by its expected return, i.e., $J(\pi) = \mathbb{E}(\sum_{t=0}^{T-1} r(x_t, a_t) \mid p_0, p, \pi)$, where $T$ is the (random) time of hitting a terminal state. We denote by $X$ the set of all terminal states. The agent’s goal is to find a policy with maximum expected return, i.e., $\pi^* = \arg\max_{\pi \in \Delta_X} J(\pi)$.

We denote by $\xi = (x_0, a_0, \ldots, x_{T-1}, \alpha_{T-1}, x_T)$ a system trajectory of length $T$, whose probability under a policy $\pi$ is defined as $p_\pi(\xi) = p_0(x_0) \prod_{t=0}^{T-1} \pi(a_t | x_t) p(x_{t+1} | x_t, a_t)$. Finally, we define $[T] := \{0, \ldots, T-1\}$.

3 Policy Optimization as Inference

Policy search in RL can be formulated as a probabilistic inference problem (e.g., [Todorov, 2008; Toussaint, 2009; Kappen et al., 2012; Levine, 2018]). The goal in the conventional RL formulation is to find a policy whose generated trajectories maximize the expected return. In contrast, in the inference formulation, we start with a prior over trajectories and then estimate the posterior conditioned on a desired outcome, such as reaching a goal state. In this formulation, the notion of a desired (optimal) outcome is introduced via independent binary random variables $O_t, t \in [T]$, where $O_t = 1$ denotes that we acted optimally at time $t$. The likelihood of $O_t$, given the state $x_t$ and action $a_t$, is modeled as

$$p(O_t = 1 \mid x_t, a_t) = \exp(\eta \cdot r(x_t, a_t)),$$

where $\eta > 0$ is a temperature parameter. This allows us to define the log-likelihood of $\pi$ being optimal as

$$\log p_\pi(O_{0:T-1} = 1 | \xi) = \log \int_\xi p_\pi(O_{0:T-1} = 1, \xi) = \mathbb{E}_{\xi \sim p_\pi} \left[ p(O_{0:T-1} = 1 \mid \xi) \right],$$

where $p(O_{0:T-1} = 1 \mid \xi)$ is the likelihood of trajectory $\xi$ being optimal and is defined as

As a result, finding an optimal policy in this setting would be equivalent to maximizing the log-likelihood in (1), i.e., $\pi^* = \arg\max_{\pi} \log p_\pi(O_{0:T-1} = 1)$.

A potential advantage of formulating RL as an inference problem is the possibility of using a wide range of approximate inference algorithms, including variational methods. In variational inference, we approximate a distribution $p(\cdot)$ with a potentially simpler (e.g., tractable factored) distribution $q(\cdot)$ in order to make the whole inference process more tractable. If we approximate $p_\pi(\xi)$ with a variational distribution $q(\xi)$, we will obtain the following variational lower-bound for the log-likelihood in (1):

$$\log p_\pi(O_{0:T-1} = 1) = \log \mathbb{E}_{\xi \sim p_\pi} \left[ \exp \left( \sum_{t=0}^{T-1} r(x_t, a_t) \right) \right]$$

$$= \log \mathbb{E}_{\xi \sim q(\xi)} \left[ \frac{p_\pi(\xi)}{q(\xi)} \cdot \exp \left( \sum_{t=0}^{T-1} r(x_t, a_t) \right) \right]$$

$$\geq \mathbb{E}_{\xi \sim q(\xi)} \left[ \log \frac{p_\pi(\xi)}{q(\xi)} + \sum_{t=0}^{T-1} r(x_t, a_t) \right]$$

$$= \eta \cdot \mathbb{E}_q \left[ \sum_{t=0}^{T-1} r(x_t, a_t) \right] - \mathbb{K}(q || p_\pi) := J(q; \pi).$$

(a) is from Jensen’s inequality and $J(q; \pi)$ is the evidence lower-bound (ELBO) of the log-likelihood function. A variety of algorithms have been proposed (e.g., [Peters and Schaal, 2007; Hachiya et al., 2009; Neumann, 2011; Levine and Koltun, 2013; Abdolmaleki et al., 2018; Fellows et al., 2019]), whose main idea is to approximate $p_\pi$ by maximizing $J(q; \pi)$ w.r.t. both $q$ and $\pi$. This often results in an EM-style algorithm in which we first fix $\pi$ and maximize $J(q; \pi)$ for $q$ (E-step), and then given the $q$ obtained in the E-step, we maximize $J(q; \cdot)$ for $\pi$ (M-step).

4 Variational Model-based Policy Optimization

In this section, we describe the ELBO objective function used by our algorithms, study the properties of the resulted optimization problem, and propose algorithms to solve it. We propose to use the variational distribution $q(\xi) = p_0(x_0) \prod_{t=0}^{T-1} q_0(a_t | x_t) q_d(x_{t+1} | x_t, a_t)$ to approximate $p_\pi(\xi)$. Note that $q$ has the same initial state distribution as $p_\pi$ (defined in Section 2), but has different control strategy (policy), $q_\pi$, and dynamics, $q_d$.

Using this variational distribution, we may write the ELBO objective (3) as

$$J(q; \pi) = \mathbb{E}_q \left[ \sum_{t=0}^{T-1} r(x_t, a_t) - \log \frac{q_\pi(a_t | x_t)}{\pi(a_t | x_t)} \right]$$

$$- \log \frac{q_d(x_{t+1} | x_t, a_t)}{p_d(x_{t+1} | x_t, a_t)} \right], \text{ where } \mathbb{E}_q[\cdot] := \mathbb{E}_q[\cdot | p_0, q_d, q_\pi].$$

To maximize $J(q; \pi)$ w.r.t. $q$ and $\pi$, we first fix $\pi$ and compute the variational distribution (E-step):
We define the variational distribution $q^*$, i.e., arg max$_{q_\nu \in \mathcal{A}}$ $\mathbb{E}[\sum_{t=0}^{T-1} \eta \cdot r(x_t, a_t)]$.

\begin{align}
q^* = (q_\nu^*, q_\rho^*) &\in \arg \max_{q_\nu \in \mathcal{A}_\nu, q_\rho \in \mathcal{A}_\rho} \mathbb{E}[\sum_{t=0}^{T-1} \eta \cdot r(x_t, a_t) \\
&- \log q_\nu(a_t|x_t) \pi(a_t|x_t) - \log q_\rho(x_{t+1}|x_t, a_t) p(x_{t+1}|x_t, a_t) | p_0, q_\rho, q_\nu],
\end{align}

and then optimize $\pi$ given $q^*$, i.e., arg max$_{\pi}$ $\mathcal{J}(q^*; \pi)$ (M-step). Note that in (5), $q_\nu^*$ and $q_\rho^*$ are both functions of $\pi$, but we remove $\pi$ from the notation to keep it lighter.

**Remark 1.** In our formulation (our choice of the variational distribution $q$), the M-step is independent of the true dynamics, $p$, and thus, can be implemented offline (using samples generated by the model $q_d$). Moreover, as we will see in Section 5, we also use the model, $q_d$, in the E-step. As discussed throughout the paper, using simulated samples (from $q_d$) and reducing the need for real samples (from $p$) is an important feature of our proposed model-based formulation and algorithms.

**Remark 2 (relationship with MPO).** There are similarities between our variational formulation and the one in the maximum a posteriori policy optimization (MPO) algorithm \cite{Abdolmaleki et al., 2018}. However, MPO sets its variational dynamics, $q_\nu$, to be the dynamics of the real system, $p$, which results in a model-free algorithm, while our approach is model-based, since we learn $q_\nu$ and use it to generate samples in both E-step and M-step of our algorithms.

In the rest of this section, we study the E-step optimization (5) and propose algorithms to solve it.

### 4.1 Properties of the E-step Optimization

We start by defining two Bellman-like operators related to the E-step optimization (5). For any variational policy $q_\nu : \mathcal{X} \rightarrow \Delta_\mathcal{A}$ and any value function $V : \mathcal{X} \rightarrow \mathbb{R}$, such that $V(x) = 0$, $\forall x \in \mathcal{X}^0$, we define the $q_\nu$-induced operator, $T_{q_\nu}$, and the optimal operator, $\mathcal{T}$, as

\begin{align}
T_{q_\nu}[V](x) &= E_{a \sim q_\nu(.|x)} [\eta \cdot r(x, a) - \log q_\nu(a|x) p(a|x)] \\
&+ \max_{q_\rho \in \mathcal{A}_\rho} E_{x' \sim q_\rho(.|x)} [V(x') - \log q_\rho(x'|x, a)], \tag{6}
\end{align}

\begin{align}
\mathcal{T}[V](x) &= \max_{q_\nu \in \mathcal{A}_\nu} T_{q_\nu}[V](x). \tag{7}
\end{align}

We define the **optimal value function** of the E-step, $V_\nu$, as

\begin{align}
V_\nu(x) &= \mathbb{E}[\sum_{t=0}^{T-1} \eta \cdot r(x_t, a_t) - \log q_\nu^*(a_t|x_t) \pi(a_t|x_t)] \\
&- \log q_\rho^*(x_{t+1}|x_t, a_t) p(x_{t+1}|x_t, a_t) | p_0, q_\rho, q_\nu^*, \tag{8}
\end{align}

For any value function $V$, we define its associated action-value function $Q : \mathcal{X} \times \mathcal{A} \rightarrow \mathbb{R}$ as

\begin{align}
Q(x, a) := \eta \cdot r(x, a) + \mathbb{E}_{x' \sim p(.|x, a)} [\exp(V(x'))]. \tag{9}
\end{align}

The following lemmas, whose proofs are reported in Appendices A.1 and A.2, show properties of operators $T_{q_\nu}$ and $\mathcal{T}$, and their relation with the (E-step) optimal value function, $V_\nu$.

**Lemma 1.** The $q_\nu$-induced and optimal operators, defined by (6) and (7), can be rewritten as

\begin{align}
T_{q_\nu}[V](x) &= E_{a \sim q_\nu(.|x)} [Q(x, a) - \log q_\nu^*(a|x) p(a|x)], \tag{10}
\end{align}

\begin{align}
\mathcal{T}[V](x) &= \log E_{a \sim \pi(.|x), x' \sim p(.|x, a)} [\exp (\eta \cdot r(x, a) + V(x'))]. \tag{11}
\end{align}

**Lemma 2.** The $q_\nu$-induced and optimal operators are monotonic and contractive. Moreover, the optimal value function $V_\nu$ is the unique fixed-point of $\mathcal{T}$ ($\mathcal{T}[V_\nu](x) = V_\nu(x)$, $\forall x \in \mathcal{X}$).

From the definition of $Q$-function in (9) and Lemma 2, we prove (in Appendix A.3) the following proposition for the action-value function associated with the E-step optimal value function $V_\nu$.

**Proposition 1.** The E-step optimal value function $V_\nu$ and its associated action-value function $Q_\nu$, defined by (9), are related as $V_\nu(x) = \log E_{a \sim \pi(.|x), x' \sim p(.|x, a)} [\exp (Q_\nu(x, a))]$, $\forall x \in \mathcal{X}$.

In the rest of this section, we show how to derive a closed-form expression for the variational distribution $q^* = (q_\nu^*, q_\rho^*)$. For any action-value function $V$, we define its corresponding variational dynamics, $q_d^V$, as the solution to the maximization problem in the definition of $T_{q_d}$ (see Eq. 6), i.e.,

\begin{align}
q_d^V (.|x, a) &\in \arg \max_{q_\rho \in \mathcal{A}_\rho} E_{x' \sim q_\rho(.)} [V(x') - \log q_d(x'|x, a) p(x'|x, a)], \tag{12}
\end{align}

and its corresponding variational policy $q_\nu^V$ ($Q$ is the action-value function associated with $V$, defined by Eq. 9), as the solution to the maximization problem in the definition of $\mathcal{T}$ (see Eqs. 7 and 10), i.e.,

\begin{align}
q_\nu^V(.|x) &\in \arg \max_{q_\nu \in \mathcal{A}_\nu} E_{a \sim q_\nu(.)} [Q(x, a) - \log q_\nu(a|x) p(a|x)], \tag{13}
\end{align}

We now derive (proof in Appendix A.4) closed-form expressions for the variational distributions $q_d^V$ and $q_\nu^V$.

**Lemma 3.** The variational dynamics and policy corresponding to a value function $V$ and its associated action-value function $Q$ can be written in closed-form as

\begin{align}
q_d^V(x'|x, a) &= p(x'|x, a) \cdot \exp(V(x')) \\
&\quad \div E_{x' \sim p(.|x, a)} [\exp(V(x'))], \tag{14}
\end{align}

\begin{align}
q_\nu^V(a|x) &= \pi(a|x) \cdot \exp(Q(x, a)) \\
&\quad \div E_{a \sim \pi(.|x)} [\exp(Q(x, a))], \forall x \in \mathcal{X}, \forall a \in \mathcal{A}. \tag{15}
\end{align}

Equations 14 and 15 show that the variational dynamics, $q_d^V$, and policy, $q_\nu^V$, can be seen as an exponential twisting of the dynamics $p$ and policy $\pi$ with weights $V$ and $Q$, respectively. For the special case $V = V_\nu$ (the E-step optimal value function), these distributions can be written in closed-form as

\begin{align}
q_d^V(x'|x, a) &= p(x'|x, a) \cdot \exp(V_\nu(x')) \\
q_\nu^V(a|x) &= \pi(a|x) \cdot \exp(Q_\nu(x, a)) \\
&\quad \div \exp(V_\nu(x)), \tag{16}
\end{align}

where the denominator of $q_\nu^V$ is obtained by applying Proposition 1 to replace $Q_\nu$ with $V_\nu$.\"
4.2 Policy and Value Iteration for the E-step

Using the results of Section 4.1, we now propose model-based and model-free dynamic programming (DP) style algorithms, i.e., policy iteration (PI) and value iteration (VI), for solving the E-step problem (5). The model-based algorithms compute the variational dynamics, \( q_d \), at each iteration, while the model-free counterparts compute \( q_d \) only at the end (upon convergence). Having access to \( q_d \) at each iteration has the advantage that we may generate samples from the model, \( q_d \), when we implement the sample-based version (RL version) of these DP algorithms in Section 5.

In the **model-based** PI algorithm, at each iteration \( k \), given the current variational policy \( q_c^{(k)} \), we

**Policy Evaluation:** Compute the \( q_c^{(k)} \)-induced value function \( V_{q_c^{(k)}}(x) \) (the fixed-point of the operator \( T_{q_c^{(k)}} \)) by iteratively applying \( T_{q_c^{(k)}} \) from (6), i.e., \( V_{q_c^{(k)}}(x) = \lim_{n \to \infty} T_{q_c^{(k)}}^n[V](x), \forall x \in X \), where the variational model \( q_d \) in (6) is computed using (14) with \( V = V^{(n)} \). We then compute the corresponding Q-function \( Q_{q_c^{(k)}}(x, a) \) using (9).

**Policy Improvement:** Update the variational distribution \( q_c^{(k+1)} \) using (15) with \( Q = Q_{q_c^{(k)}} \).

Upon convergence, i.e., \( q_c^{(\infty)} = q_c^* \), we compute \( q_d^* \) from (14).

The **model-free** PI algorithm is exactly the same, except in its policy evaluation step, the \( q_c^{(k)} \)-induced operator \( T_{q_c^{(k)}} \) is applied using (10) (without the variational dynamics \( q_d \)). In this case, the variational dynamics \( q_d \) is computed only upon convergence, \( q_d^* \), using (14).

**Lemma 4.** The model-based and model-free PI algorithms converge to their optimal values, \( q_c^* \) and \( q_d^* \), defined by (5), i.e., \( q_c^{(\infty)} = q_c^* \) and \( q_d^{(\infty)} = q_d^* \). (proof in Appendix ??)

We can similarly derive **model-based** and **model-free** (VI) algorithms for the E-step. These algorithms start from an arbitrary value function \( V \) and iteratively apply the optimal operator \( T \) from (6) and (7) (model-based) and (11) (model-free) until convergence, i.e., \( V_{q_c^*}(x) = \lim_{n \to \infty} T_{q_c^*}^n[V](x), \forall x \in X \). Given \( V_{q_c^*} \), these algorithms first compute \( Q_{q_c^*} \) from Proposition 1, and then compute \( q_c^{(\infty)} \) and \( q_d^{(\infty)} \) using (16). From the properties of the optimal operator \( T \) in Lemma 2, both model-based and model-free VI algorithms converge to \( q_c^* \) and \( q_d^* \).

In the rest of the paper, we focus on the PI approach, in particular the model-based one, and only report the details of the VI algorithms in Appendix B. In the next section, we show how the PI algorithms can be implemented and combined with a routine for solving the M-step, when the true MDP model \( p \) is unknown (the RL setting) and the state and action spaces are large that require using function approximation.

5 Variational Model-based RL Algorithm

In this section, we propose a RL algorithm, called **variational model-based policy optimization (VMBPO)**. It is an EM-style algorithm based on the variational formulation proposed in Section 4. The E-step of VMBPO is the sample-based implementation of the model-based PI algorithm, described in Section 4.2. We describe the E-step and M-step of VMBPO in details in Sections 5.1 and 5.2, and report its pseudo-code in Algorithm 1 in Appendix C. VMBPO uses 8 neural networks to represent: policy \( \pi \), variational dynamics \( q_d \), variational policy \( q_c \), log-likelihood ratio \( \nu = \log q_d/p \), value function \( V \), action-value function \( Q \), target value function \( V' \), and target action-value function \( Q' \), with parameters \( \theta_\pi, \theta_d, \theta_c, \theta_v, \theta_{\nu_v}, \theta_q, \theta_{\nu_q}, \text{ and } \theta_{q_d} \), respectively.

5.1 The E-step of VMBPO

At the beginning of the E-step, we generate a number of samples \( (x, a, r, x') \) from the current baseline policy \( \pi \), i.e., \( a \sim \pi(\cdot|x) \) and \( r = r(x, a) \), and add them to the buffer \( D \). The E-step consists of four updates: 1) computing the variational dynamics \( q_d \), 2) estimating the log-likelihood ratio \( \log(p/q_d) \), 3) computing the \( q_c \)-induced value and action-value functions \( V_q \), and \( Q_q \), (critic update), and finally 4) computing the new variational policy \( q_c \) (actor update). We describe the details of each step below.

**Step 1.** (Computing \( q_d \))

We find \( q_d \) as a solution to the optimization problem (12) for \( V \) equal to the target value network \( V' \). Since the \( \gamma V' \) in (14) is the solution of (12), we compute \( q_d \) by minimizing \( \text{KL}(q_d' || q_d) \), which results in the following forward KL loss for all \( x \in X \) and \( a \in A \):

\[
\theta_d = \arg\min_{\theta} \text{KL}(p(x, a) \cdot \exp(r(x, a) + V^{(\cdot)}(\cdot; \theta_q')) ) - Q'(x, a; \theta_q') || q_d(\cdot|x; \theta_q'()) \]
\[
= \arg\max_{\theta} \mathbb{E}_{x' \sim p(x, a)} \left[ \exp(\eta \cdot r(x, a) + V^{(\cdot)}(\cdot; \theta_q')) - Q'(x, a; \theta_q') \cdot \log(q_d(\cdot|x; \theta_q'()) ) \right],
\]

(17) where \( (\cdot) \) is by removing the \( \theta \)-independent terms from (17). We update \( \theta_d \) by taking several steps in the direction of the gradient of a sample average of the loss function (18), i.e.,

\[
\theta_d = \arg\max_{\theta} \sum_{(x, a, r, x') \sim D} \exp(\eta \cdot r + V'(x'; \theta_q'))
\[
- Q'(x, a; \theta_q') \cdot \log(q_d(x'|x, a; \theta_q'))
\]

(19) where \( (x, a, r, x') \) are randomly sampled from \( D \). The intuition here is to focus on learning the dynamics model in the regions of the state-action space that have higher temporal difference (regions with higher anticipated future return).

**Step 2.** (Computing \( \log(p/q_d) \))

Using the duality of \( f \)-divergence [Nguyen et al., 2008] w.r.t. the reverse KL-divergence, the log-likelihood ratio \( \log(p/q_d(\cdot|x, a; \theta_d)/p(\cdot|x)) \) is a solution to

\[
\log \left( \frac{q_d(x'|x, a; \theta_d)}{p(x'|x, a)} \right) = \arg\max_{\nu \in X \times A \times X \times R} \mathbb{E}_{x' \sim q_d(\cdot|x, a, \theta_d)} \left[ \nu(x'|x, a) \right] - \mathbb{E}_{x' \sim p(\cdot|x, a)} \left[ \exp(\nu(x'|x, a)) \right],
\]

(20) for all \( x, x' \in X \) and \( a \in A \). Note that the optimizer of (20) is unique almost surely (at \( (x, a, x') \) with \( \mathbb{P}(x'|x, a) > 0 \), because \( q_d \) is absolutely continuous w.r.t. \( p \) (see the definition of \( q_d \) in Eq. 14) and the objective function of (20) is strictly concave. The optimization problem (20) allows us to compute \( \nu(\cdot; \theta_v) \) as an approximation to the log-likelihood ratio.
log(q_d(·; θ_d)/p). We update θ_e by taking several steps in the
direction of the gradient of a sample average of (20), i.e.,

\[\theta_e = \arg \max_\theta \mathbb{E}_{(x,a) \sim D} \left[ \log \frac{q_e(a|x)}{\pi(a|x)} \right] - \frac{1}{T} \sum_{t=0}^{T-1} \log \big( q_e(f(x,c;\theta_e)) \big) - Q(x,a;\theta_q) \]  

(23)

where \( E \) is the set of samples for which \( x' \) is drawn from
the variational dynamics, i.e., \( x' \sim q_d(·|x,a) \). Here we first
sample \((x,a,x')\) randomly from \( D \) and use them in the second
sum. Then, for all \((x,a)\) that have been sampled, we generate
\( x' \) from \( q_d \) and use the resulting samples in the first sum.

**Step 3. (critic update)** To compute \( V_q \) and its action-
value \( Q_{q_e} \), we rewrite (6) with the maximizer \( q_d \) from Step 1
and the log-likelihood ratio \( \log(q_d/p) \) from Step 2:

\[ T_q[V](x) = \mathbb{E}_{a \sim q_e(·|x)} \left[ \eta \cdot r(x,a) - \log \frac{q_e(a|x)}{\pi(a|x)} \right] + \mathbb{E}_{x' \sim q_d(·|x,a;\theta_d)} \left[ V(x';\theta'_e) - \nu(x'|x,a;\theta_v) \right] \].

Since \( T_q \) can be written as both (10) and (22), we compute the
\( q_e \)-induced \( Q \)-function by setting the RHS of these equations
equal to each other, i.e., for all \( x \in X \) and \( a \sim q_e(·|x;\theta_e) \),

\[ Q(x,a;\theta_v) = \arg \min_{q \sim q_e(·|x,a;\theta_d)} \mathbb{E}_{x' \sim q_d(·|x,a;\theta_d)} \left[ V(x';\theta'_e) - \nu(x'|x,a;\theta_v) \right] \].

(22)

Since the expectation in (22) is w.r.t. the variational dynamics
(model) \( q_e \), we can estimate \( Q \) only with samples generated
from the model. We do this by taking several steps in the
direction of the gradient of a sample average of the square-loss
obtained by setting both sides of (22) equal, i.e.,

\[ \theta_v = \arg \min_{\theta \in \mathcal{D}} \mathbb{E}_{(x,a) \sim E} \left[ (Q(x,a;\theta) - \eta r(x,a) - \nu(x'|x,a;\theta_v))^2 \right] \].

(23)

Note that in (22), the actions are generated by \( q_e \). Thus, in (23),
we first randomly sample \( x \), then sample \( a \) from \( q_e(·|x;\theta_e) \),
and finally draw \( x' \) from \( q_d(·|x,a;\theta_d) \). If the reward function
is known (chosen by the designer of the system), then it is used
to generate the reward signals \( r = r(x,a) \) in (23), otherwise,
a reward model has to be learned.

After estimating \( Q_{q_e} \) we approximate \( V_q \), the fixed-point
of \( T_q \), using the fixed-point of \( T_q \), definition in (10) as \( T_q[V](x) \approx V(x) \approx \mathbb{E}_{a \sim q_e(·|x)} [Q(x,a;\theta_v) - \log \frac{q_e(a|x;\theta_v)}{\pi(a|x;\theta_v)}] \). This results in updating
\( V_q \) by taking several steps in the direction of the gradient of a
sample average of the square-loss obtained by setting the two sides of the above equation equal, i.e.,

\[ \theta_v = \arg \max_{\theta \in \mathcal{D}} \mathbb{E}_{(x,a) \sim E} \left[ (V(x;\theta) - Q(x,a;\theta_v) + \log \frac{q_e(a|x;\theta_v)}{\pi(a|x;\theta_v)})^2 \right] \],

(24)

where \( x \) is randomly sampled and \( a \sim q_e(·|x;\theta_e) \) (without
sampling from the true environment).

**Step 4. (actor update)** We update the variational policy
\( q_e \) (policy improvement) by solving the optimization problem
(13) for the \( Q \) estimate by the critic in Step 3. Since the
\( q_e \) that optimizes (13) can be written as (15), we update it by
minimizing \( KL(q_e||q_e^o) \). This results in the following reverse
KL loss (for all \( x \in X \)):

\[ \theta_e = \arg \min_{\theta} KL(q_e(·|x;\theta)|\log \frac{\pi(·|x;\theta_v)}{\pi(·|x;\theta_v)}) \]

\[ = \arg \max_{\theta} \mathbb{E}_{a \sim q_e} \left[ \log \frac{q_e(a|x;\theta)}{\pi(a|x;\theta_v)} - Q(x,a;\theta_v) \right] \]

If we reparameterize \( q_e \) using a transformation \( a = f(x,c;\theta_e) \),
where \( \epsilon \) is a Gaussian noise, we can update \( \theta_e \) by taking several
steps in the direction of the gradient of a sample average of the
above loss, i.e.,

\[ \theta_e = \arg \min_{\theta} \mathbb{E}_{(x,c) \sim D} \left[ \log \frac{q_e(f(x,c;\theta_e))}{Q(x,a;\theta_v)} \right] \]  

(25)

We can also compute \( q_e \) as the closed-form solution to (15), as
described in [Abdolmaleki et al., 2018]. They refer to this as
non-parametric representation of the variational distribution.

**5.2 The M-step of VMBPO**

As described in Section 4, the goal of the M-step is to
improve the baseline policy \( \pi \), given the variational model
\( q^* = (q_{a}^*, q_{d}^*) \) learned in the E-step, by solving the
following optimization problem:

\[ \pi \leftarrow \arg \max_{\pi} \mathbb{E}_{x \sim \pi} \left[ \sum_{t=0}^{T-1} \eta r(x,a) - \log \frac{q_{a}^*(a|x)}{\pi(a|x)} - \log \frac{q_{d}^*(x_{t+1}|x_t,a_t)}{p(x_{t+1}|x_t,a_t)} \right] \].

(26)

A nice feature of (26) is that it can be solved using only the
variational model \( q^* \), without the need for samples from the
true environment \( p \). However, it is easy to see that if the policy
space considered in the M-step, II, contains the policy space
used for \( q_e \) in the E-step, then we can trivially solve the M-
step by setting \( \pi = q^* \). Although this is an option, it is more
efficient in practice to solve a regularized version of (26).

A practical way to regularize (26) is to make sure that the new
baseline policy \( \pi \) remains close to the old one, which results
in the following optimization problem:

\[ \theta_e \leftarrow \arg \max_{\theta} \mathbb{E}_{x \sim \pi} \left[ \sum_{t=0}^{T-1} \log \frac{\pi(a|x;\theta)}{\pi(a|x;\theta_v)} - \lambda \cdot KL(\pi(·|x;\theta_v)||\pi(·|x;\theta)) \right] \].

(27)

This is equivalent to the weighted MAP formulation used in
the M-step of MPO [Abdolmaleki et al., 2018]. In MPO, they
define a prior over the parameter \( \theta \) and add it as \( \log P(\theta) \)
to the objective function of (26). Then, they set the prior \( P(\theta) \)
to a specific Gaussian and obtain an optimization problem
similar to (27) (see Section 3.3 in [Abdolmaleki et al., 2018]).

However, in the absence of a variational dynamics model (i.e.,
\( q_d = p \)), they need real samples to solve their optimization
problem, while our model-based approach uses simulated
samples.

**6 Experiments**

To illustrate the effectiveness of VMBPO, we (i) compare
it with several state-of-the-art RL methods, and (ii) evaluate
sample efficiency of MBRL via ablation analysis.

**Comparison with Baseline RL Algorithms.** We compare
VMBPO with five baselines, two popular model-free algo-
rithms: MPO [Abdolmaleki et al., 2018] and SAC [Haarnoja
et al., 2018], and three recent model-based algorithms:
MBPO [Janner et al., 2019], PETS [Chua et al., 2019], and
STEVE [Buckman et al., 2018]. We also compare VMBPO
with its variant that uses a model-free PI algorithm to solve the
E-step (see Section 4.2). We refer to this variant as VMBPO-
MFE and describe it in details in Appendix D. We evaluate all
the algorithms on a classical control task: *Pendulum*, and five MuJoCo tasks: *Hopper*, *Walker2D*, *HalfCheetah*, *Reacher*, and *Reacher7DOF*. We use similar neural network architectures (for the dynamics model, value functions, and policies) for VMBPO and MBPO. The detailed description of the network architectures and hyper-parameters is reported in Appendix E. Since we use a parametric representation for \( q \), in the E-step of VMBPO, as discussed in Section 5.2, we simply set \( \pi = q \) in its M-step. We set the number of training steps to 400,000 for the difficult environments (Walker2D, HalfCheetah), to 150,000 for the medium one (Hopper), and to 50,000 for the simpler ones (Pendulum, Reacher, Reacher7DOF). We evaluate policy performance every 1,000 training steps. Each measurement is an average return over 5 episodes, generated with a separate random seed.

To illustrate the relative performance of the algorithms, we report the average return of VMBPO, VMBPO-MFE, and the baselines, with their best hyper-parameters, in Table 1 and Figure 1 (see Appendix E.1). The results show that VMBPO performs better than the baselines in most of the tasks, and usually converges faster even when the final performances are similar. The data-efficiency of VMBPO is mainly the result of using synthetic data generated by the learned model, and its extra performance can be attributed to jointly learning model and policy using a universal objective function.

The results also show that VMBPO-MFE outperforms MPO in 4 out of the 6 domains. However, in some cases its learning curve degrades and results in poor final performance. This is partly due to the instability in critic learning caused by sample variance amplification in exponential-TD minimization (see Eq. 39 in Sec. D.1). A way to alleviate this issue is to add a temperature term \( \tau \) to the exponential-TD update [Borkar, 2002], although tuning this hyper-parameter is often non-trivial.\(^3\)

3The variance is further amplified with a large \( \tau \), but the critic learning is hampered by a small \( \tau \).

To study the sensitivity of the algorithms w.r.t. the hyper-parameters, we report their performance averaged over all hyper-parameter/random-seed configurations in Table 2 and Figure 2 (see Appendix E.1). These results show that VMBPO outperforms other baselines. VMBPO performs better than MPO but is sometimes unstable.

### Table 1: The mean ± standard deviation of the final returns with the best hyper-parameter configuration for each algorithm.

<table>
<thead>
<tr>
<th>Environment</th>
<th>VMBPO</th>
<th>MBPO</th>
<th>STEVE</th>
<th>PETS</th>
<th>VMBPO-MFE</th>
<th>SAC</th>
<th>MPO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pendulum</td>
<td>-125.7 ± 72.4</td>
<td>-120.6 ± 8.4</td>
<td>-258.3 ± 297.9</td>
<td>-183.8 ± 172.9</td>
<td>-125.7 ± 190.1</td>
<td>-147.4 ± 197.0</td>
<td>-151.9 ± 355.7</td>
</tr>
<tr>
<td>Hopper</td>
<td>3897.4 ± 830.7</td>
<td>2493.1 ± 236.6</td>
<td>2975.0 ± 237.6</td>
<td>94.5 ± 114.2</td>
<td>1349.7 ± 184.1</td>
<td>2020.8 ± 354.3</td>
<td>1589.7 ± 706.0</td>
</tr>
<tr>
<td>Walker2D</td>
<td>4226.1 ± 834.3</td>
<td>3881.3 ± 735.3</td>
<td>356.3 ± 196.3</td>
<td>915.7 ± 354.5</td>
<td>3334.5 ± 122.8</td>
<td>3264.6 ± 388.3</td>
<td>2859.4 ± 112.7</td>
</tr>
<tr>
<td>HalfCheetah</td>
<td>11312.0 ± 933.1</td>
<td>1187.7 ± 997.1</td>
<td>842.9 ± 596.9</td>
<td>13272.6 ± 4926.4</td>
<td>4647.3 ± 505.8</td>
<td>5088.3 ± 1625.1</td>
<td>4909.2 ± 623.7</td>
</tr>
<tr>
<td>Reacher</td>
<td>-11.4 ± 27.0</td>
<td>-12.6 ± 25.9</td>
<td>-141.8 ± 355.7</td>
<td>-55.5 ± 39.0</td>
<td>-23.9 ± 23.8</td>
<td>-75.9 ± 336.7</td>
<td></td>
</tr>
<tr>
<td>Reacher7DOF</td>
<td>-13.3 ± 293.6</td>
<td>-15.1 ± 398.8</td>
<td>-45.6 ± 36.1</td>
<td>-35.3 ± 49.6</td>
<td>-2.4 ± 112.0</td>
<td>-18.8 ± 33.8</td>
<td></td>
</tr>
</tbody>
</table>

### Table 2: The mean ± standard deviation of the average of the final returns over all hyper-parameter configurations. VMBPO is much more robust to change in hyper-parameters than the other baselines. We do not include PETS and STEVE because their hyper-parameters are directly adopted from their papers.

<table>
<thead>
<tr>
<th>Environment</th>
<th>VMBPO</th>
<th>MBPO</th>
<th>VMBPO-MFE</th>
<th>SAC</th>
<th>MPO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pendulum</td>
<td>-147.4 ± 76.1</td>
<td>-146.8 ± 27.6</td>
<td>-146.8 ± 430.6</td>
<td>-106.8 ± 209.0</td>
<td></td>
</tr>
<tr>
<td>Hopper</td>
<td>-1256.2 ± 1256.0</td>
<td>-1256.3 ± 202.3</td>
<td>-1256.2 ± 202.3</td>
<td>-1256.2 ± 202.3</td>
<td></td>
</tr>
<tr>
<td>Walker2D</td>
<td>3326.6 ± 1276.3</td>
<td>-2977.3 ± 397.3</td>
<td>-1247.1 ± 767.1</td>
<td>-1247.1 ± 767.1</td>
<td></td>
</tr>
<tr>
<td>HalfCheetah</td>
<td>10586.7 ± 4577.2</td>
<td>7588.1 ± 4314.1</td>
<td>2834.6 ± 1062.9</td>
<td>6312.0 ± 2299.7</td>
<td>3258.2 ± 9703.1</td>
</tr>
<tr>
<td>Reacher</td>
<td>13.8 ± 48.5</td>
<td>-17.5 ± 48.3</td>
<td>-22.2 ± 507.6</td>
<td>-12.2 ± 301.5</td>
<td>-108.2 ± 347.1</td>
</tr>
<tr>
<td>Reacher7DOF</td>
<td>-15.2 ± 66.3</td>
<td>-17.2 ± 101.8</td>
<td>-78.9 ± 439.1</td>
<td>-114.2 ± 1067.9</td>
<td>-95.3 ± 4265.9</td>
</tr>
</tbody>
</table>

### Figure 1: Performance of VMBPO with different number of samples generated from the dynamics model \( q_d \).

Figure 1 shows the learning performance averaged over all hyper-parameter/random-seed configurations and illustrates how synthetic data can help with policy learning. The results show that increasing the amount of synthetic data generally improves the policy convergence rate. In the early phases, when the model is inaccurate, sampling from it may slow down learning, while in the later phases, with an improved model, adding more synthetic data can lead to a more significant performance boost.

### 7 Conclusion

We formulated the problem of jointly learning and improving model and policy in RL as a variational lower-bound of a log-likelihood and proposed EM-style algorithms to solve it. Our algorithm, called variational model-based policy optimization (VMBPO), uses model-based policy iteration for solving the E-step. We compared our (E-step) model-based and model-free algorithms with each other, and with a number of state-of-the-art model-based (e.g., MBPO) and model-free (e.g., MPO) RL algorithms, and showed its sample efficiency and performance.

We briefly discussed VMBPO algorithms in which the E-step is solved by value iteration methods. However, full implementation of these algorithms and studying their relationship with the existing methods requires more work that we leave for future. Another future directions are: 1) finding more efficient implementation for VMBPO, and 2) using VMBPO style algorithms in solving control problems from high-dimensional observations, by learning a low-dimensional latent space and a latent dynamics, and perform control there. This class of algorithms is referred to as learning controllable embedding [Levine et al., 2020].
References


