Constrained Expectation-Maximization Methods for Effective Reinforcement Learning

Gang Chen
School of Engineering
and Computer Science,
Victoria University of Wellington,
Wellington, New Zealand
aaron.chen@ecs.vuw.ac.nz

Yiming Peng
School of Engineering
and Computer Science,
Victoria University of Wellington,
Wellington, New Zealand
yiming.peng@ecs.vuw.ac.nz

Mengjie Zhang
School of Engineering
and Computer Science,
Victoria University of Wellington,
Wellington, New Zealand
mengjie.zhang@ecs.vuw.ac.nz

***Abstract*—Recent advancement on reinforcement learning (RL) algorithms shows that effective learning of parametric action-selection policies can often be achieved through direct optimization of a performance lower bound subject to pre-defined policy behavioral constraints. Driven by this understanding, this paper seeks to develop new policy search techniques where RL is achieved through maximizing a performance lower bound obtained originally based on an Expectation-Maximization method. For reliable RL, our new learning techniques must also simultaneously guarantee constrained policy behavioral changes measured through KL divergence. Two separate approaches will be pursued to tackle our constrained policy optimization problems, resulting in two new RL algorithms. The first algorithm utilizes a conjugate gradient technique and a Bayesian learning method for approximate optimization. The second algorithm focuses on minimizing a loss function derived from solving the Lagrangian for constrained policy search. Both algorithms have been experimentally examined on several benchmark problems provided by OpenAI GYM. The experiment results clearly demonstrate that our algorithms can be highly effective in comparison to several well-known RL algorithms.**

I. Introduction

Since the invention of the REINFORCE algorithm in [28], many *Policy Search* (PS) algorithms have been developed in the past decades to tackle difficult *Reinforcement Learning* (RL) problems [27, 22, 1, 22, 13, 5, 16, 26, 25, 18, 12, 24]. Without building an *action-selection policy* indirectly from learned value functions, PS methods explicitly maintain parametric policies and are expected to achieve near optimal learning performance through direct updating of the corresponding *policy parameters*.

For a comprehensive summary of PS algorithms, please refer to [21, 13, 22]. It is important to note that the effectiveness of almost all PS algorithms depend heavily on their *policy update strategies*. While *policy gradient* methods remain as the most popular strategy for policy update [1, 18, 9, 11], the prominent success of several recently developed PS algorithms such as the Policy Learning by Weighting Exploration with the Returns (PoWER) algorithm proposed in [14] and the Trust Region Policy Optimization (TRPO) algorithm proposed in [26] clearly show that *direct optimization of a performance lower bound* may stand for a potentially more effective strategy for policy update.

For example, while developing the PoWER algorithm, Kober and Peters first established a lower bound on policy improvements based on the *Expectation-Maximization* (EM) method originally studied in [4]. Policy update was subsequently achieved through direct maximization of this lower bound with respect to all policy parameters, resulting in more effective RL in comparison to regular policy gradient methods [27, 22, 1, 3]. Meanwhile, any policy to be learned through PoWER is treated as a linear function of its parameters. Based on this linearity assumption, optimal policy parameter values can be determined analytically [14]. Although PoWER can effectively solve many difficult robotic control problems [14, 15], there are two major limitations of this algorithm. First, it is difficult for PoWER to train policies represented as deep neural networks (DNNs). Second, each individual update to policy parameters in PoWER can lead to significant policy behavioral changes since bound optimization is never subject to any behavioral constraints. Consequently, the overall learning reliability can be significantly affected.

Different from PoWER, for reliable RL, TRPO introduced a constrained optimization problem where any update to policy parameters must never violate the constraint defined on the *average KL divergence* in between the previous policy and the updated policy [26]. Guided by this constraint, a *conjugate gradient method* is further exploited to ensure steady policy improvement. Optimizing the performance bound in TRPO requires a RL agent to constantly and accurately estimate an *advantage function*. In this paper, we aim to show that effective RL can also be achieved through optimizing an EM-based performance bound defined purely on the *long-term cumulative rewards* of trajectories [4, 23, 14]. Such a bound may also enable more flexible RL since a trajectory's cumulative reward can be formulated as an arbitrary function of the rewards obtained in each state visited along the trajectory, other than a linear summation of these rewards.

Inspired by the key strengths of both PoWER and TRPO, in this paper we aim to develop new policy search techniques to directly maximize the EM-based performance lower bound while simultaneously satisfy a behavioral constraint measured through the KL divergence. To the best of our knowledge, this constrained optimization problem has seldom been studied indepth before. By solving this problem, we expect to improve the reliability and applicability of EM-based algorithms for RL. In the meantime, effective RL can be achieved without the necessity of learning the advantage function.

Our research goal will be pursued through two different approaches, resulting in two new PS algorithms. In the first approach, following the algorithmic design of TRPO, we will adopt the conjugate gradient technique followed by linear line search to approximately solve our constrained optimization problem. To reduce the variance of policy gradient estimation, we will employ a Bayesian RL method introduced in [7, 8] that strictly supports the *likelihood principle*. The algorithm constructed subsequently will be called the *Constrained Bayesian Expectation-Maximization* (CBEM) algorithm for RL. Different from the first approach, in the second approach, we will analytically solve a *Lagrangian* obtained from the constrained policy optimization problem. The solution lays the foundation for our second PS algorithm which is designed to minimize a new loss function defined over sampled trajectories. For convenience of discussion, this algorithm will be called the *Lagrangian Expectation-Maximization* (LEM) algorithm for RL.

Similar to LEM, the Relative Entropy Policy Search (REPS) algorithm was also developed by using the method of Lagrangian multipliers [24]. However, different from LEM, REPS aims at optimizing the expected performance of a parametric policy at the level of individual state-action pairs rather than a trajectory-based performance lower bound. Also, the same as in PoWER, REPS assumes that the value function is linear with respect to state features and policy parameters. It is therefore not suitable for training DNNs. As pointed out in [10], majority of deep RL methods for problems with discrete actions are based around the estimation of value functions. Our CBEM and LEM algorithms, on the other hand, support discrete action selection under a pure policy search paradigm. They are also compatible with many model-guided policy search techniques [16]. However the potential use of environment models for effective policy search will not be explored in this paper.

On three benchmark RL problems provided by OpenAI GYM [2], we will experimentally show that CBEM and LEM can optimize nonlinear policies represented as DNNs with thousands of parameters. Moreover our algorithms can achieve highly competitive performance and sometimes outperform a few well-known RL algorithms including TRPO with generalized advantage estimation [25], Deep Q-learning (DQN) [19], and the REINFORCE algorithm [28].

II. Constrained Optimization for Effective Reinforcement Learning

In this paper we consider mainly RL problems with *continuous states* $\boldsymbol{s} \in \mathbb{S} \subseteq \mathbb{R}^n$ and *discrete actions* $a \in \{a^1, \ldots, a^m\}$. Such problems frequently appear in many real-world applications including industrial control, automation and robotics. Our RL algorithms can be extended to support continuous and multi-dimensional actions. However this may require us to investigate a NN model (different from the NNs used in TRPO and DQN) that produces both the mean as well as the standard deviation of its action output and is beyond the scope of this paper. At each time step t, a RL agent observes the current state $\boldsymbol{s}_t$ of the environment and performs one of the m alternative actions a_t, resulting in a *state transition* to a new state $\boldsymbol{s}_{t+1}$. Meanwhile, a scalar reward $r(\boldsymbol{s}_t, a_t)$ will be produced as an environmental feedback. Starting from an initial state $\boldsymbol{s}_0$ at time $t = 0$, the agent performs T actions at consecutive time steps to produce a trajectory ξ (also known as an episode or path). The *long-term cumulative reward* of ξ, i.e. $R(\xi)$, is defined subsequently as

$$R(\xi) = F\left(r(\boldsymbol{s}_0, a_0), \ldots, r(\boldsymbol{s}_{T-1}, a_{T-1})\right) \tag{1}$$

where F is an arbitrary function of rewards obtained upon following the trajectory ξ. It can be possibly defined in unlimited number of different forms and studying these possibilities is beyond the scope of this paper. Instead, a common γ-discounted linear summation of all reward terms $r(\boldsymbol{s}_t, a_t)$ in (1) will be adopted in this paper to calculate $R(\xi)$.

Depending on the policy π under use, an agent may produce each trajectory ξ through direct interaction with the learning environment at a certain probability, denoted as $Pr_\pi(\xi)$. As a common ground for designing many PS algorithms [13], π is considered as a parametric function that controls the probability of choosing any action a_t in arbitrary state $\boldsymbol{s}_t$. Since the behavior of policy π is completely determined by its policy parameters $\boldsymbol{\theta}$, $Pr_\pi(\xi)$ can be re-written as $Pr_{\boldsymbol{\theta}}(\xi)$. Therefore, given a parametric policy $\pi_{\boldsymbol{\theta}}$, its performance is measured as

$$J(\boldsymbol{\theta}) = \int_\xi Pr_{\boldsymbol{\theta}}(\xi) R(\xi) \mathrm{d}\xi \tag{2}$$

The ultimate goal for RL is hence to identify the optimal policy parameters $\boldsymbol{\theta}^*$ so as to achieve the maximum learning performance in (2). Driven by this goal, some EM-based RL algorithms such as PoWER choose to optimize a performance lower bound $L_{\boldsymbol{\theta}}(\boldsymbol{\theta}')$ as derived below

$$\begin{aligned}\log J(\boldsymbol{\theta}') &= \log \int_\xi \frac{Pr_{\boldsymbol{\theta}}(\xi)}{Pr_{\boldsymbol{\theta}}(\xi)} Pr_{\boldsymbol{\theta}'}(\xi) R(\xi) \mathrm{d}\xi \\ &\geq \int_\xi Pr_{\boldsymbol{\theta}}(\xi) R(\xi) \log \frac{Pr_{\boldsymbol{\theta}'}(\xi)}{Pr_{\boldsymbol{\theta}}(\xi)} \mathrm{d}\xi + \mathbb{C} \\ &\propto \int_\xi Pr_{\boldsymbol{\theta}}(\xi) R(\xi) \log Pr_{\boldsymbol{\theta}'}(\xi) \mathrm{d}\xi = L_{\boldsymbol{\theta}}(\boldsymbol{\theta}')\end{aligned} \tag{3}$$

where $\boldsymbol{\theta}$ refers to the policy parameters before update and $\boldsymbol{\theta}'$ stands for the updated policy parameters. Given fixed $\boldsymbol{\theta}$, the aim is to find suitable $\boldsymbol{\theta}'$ that maximize $L_{\boldsymbol{\theta}}(\boldsymbol{\theta}')$. We assume that, during RL, a repository of sampled trajectories will be made available to the RL agent and can be jointly used to approximately optimize $L_{\boldsymbol{\theta}}(\boldsymbol{\theta}')$. No restrictions apply to the possible choice of these trajectories. We will adopt a simple strategy of randomly sampling a fixed number of trajectories at each learning iteration for this purpose (see Algorithm 1 and Algorithm 2).

In order to achieve high learning reliability, similar to [26, 25], the optimization of $L_{\boldsymbol{\theta}}(\boldsymbol{\theta}')$ in (3) must satisfy certain behavioral constraints. In this paper, we will consider two related behavioral constraints, each of which will be utilized to build a different RL algorithm. They are presented respectively in (4) and (5) below.

$$D^1_{KL}(\pi_{\boldsymbol{\theta}}, \pi_{\boldsymbol{\theta}'}) = \int_s Pr_{\boldsymbol{\theta}}(s) \sum_a \pi_{\boldsymbol{\theta}}(s,a) \log\left(\frac{\pi_{\boldsymbol{\theta}}(s,a)}{\pi_{\boldsymbol{\theta}'}(s,a)}\right) \mathrm{d}s \\ \leq \delta \tag{4}$$

$$D^2_{KL}(\pi_{\boldsymbol{\theta}}, \pi_{\boldsymbol{\theta}'}) = \int_\xi Pr_{\boldsymbol{\theta}'}(\xi) \log\left(\frac{Pr_{\boldsymbol{\theta}'}(\xi)}{Pr_{\boldsymbol{\theta}}(\xi)}\right) \mathrm{d}\xi \leq \delta \tag{5}$$

where $Pr_{\boldsymbol{\theta}}(s)$ in (4) refers to the probability for an agent to reach state s upon following policy $\pi_{\boldsymbol{\theta}}$ and δ controls the size of the trust region [26]. It is obvious to see that D^1_{KL} in (4) measures the average KL divergence at the level of each individual state. On the other hand, D^2_{KL} measures the KL divergence at the trajectory level. In the sequel, the CBEM algorithm will be developed first to optimize $L_{\boldsymbol{\theta}}(\boldsymbol{\theta}')$ subject to $D^1_{KL} \leq \delta$. Afterwards the LEM algorithm will be further proposed to optimize $L_{\boldsymbol{\theta}}(\boldsymbol{\theta}')$ subject to $D^2_{KL} \leq \delta$.

III. A Constrained Bayesian Expectation-Maximization Algorithm for Reinforcement Learning

In practice, the objective of maximizing $L_{\boldsymbol{\theta}}(\boldsymbol{\theta}')$ in (3) can be pursued through continued update of $\boldsymbol{\theta}'$ according to a new form of policy gradient shown below.

$$\frac{\partial L_{\boldsymbol{\theta}}(\boldsymbol{\theta}')}{\partial \boldsymbol{\theta}'} = \int_\xi Pr_{\boldsymbol{\theta}}(\xi) R(\xi) \frac{\partial \log Pr_{\boldsymbol{\theta}'}(\xi)}{\partial \boldsymbol{\theta}'} \mathrm{d}\xi \tag{6}$$

Notice that when $\boldsymbol{\theta}' = \boldsymbol{\theta}$, the policy gradient in (6) becomes the standard gradient used in many existing works [27, 13]. Since effective RL depends on accurate estimation of $\partial L_{\boldsymbol{\theta}}(\boldsymbol{\theta}')/\partial\boldsymbol{\theta}'$, we decide to adopt a Bayesian RL technique that can significantly reduce the estimation error [8]. Specifically, a quick inspection of (6) suggests that its integrand can be divided into two parts:

$$F(\xi) = R(\xi) \text{ and } G(\xi) = Pr_{\boldsymbol{\theta}}(\xi) \frac{\partial \log Pr_{\boldsymbol{\theta}'}(\xi)}{\partial \boldsymbol{\theta}'} \tag{7}$$

Let us consider $F(\xi)$ as a unknown function of ξ. Furthermore the value of $F(\xi)$ for arbitrary trajectory ξ can be approximated through a *Gaussian Process* (GP) model based on a given repository of μ sampled trajectories and their respective observed cumulative rewards, denoted as $\boldsymbol{D}_\mu = \{(\xi_i, R(\xi_i))\}_{i=1}^{\mu}$. Meanwhile $G(\xi)$ is deemed as a known function of ξ. In line with (7), the integral in (6) can be reliably estimated by using *Bayesian Quadrature* [20]. Specifically, following similar derivations in [7, 8], the expected policy gradient in (6) can be obtained as

$$E\left[\frac{\partial L_{\boldsymbol{\theta}}(\boldsymbol{\theta}')}{\partial \boldsymbol{\theta}'} | \boldsymbol{D}_\mu\right] = \boldsymbol{U}_{\boldsymbol{\theta}'} \boldsymbol{C}_{\boldsymbol{\theta}'}(\boldsymbol{D}_\mu) \boldsymbol{Y} \tag{8}$$

where $\boldsymbol{Y} = \{R(\xi_1) - \bar{R}, \ldots, R(\xi_\mu) - \bar{R}\}$ and $\bar{R}$ refers to the average cumulative reward over all sampled trajectories. $\boldsymbol{U}_{\boldsymbol{\theta}'}$ is a matrix with μ columns. Its i-th column, $1 \leq i \leq \mu$, equals to $\partial \log Pr_{\boldsymbol{\theta}'}(\xi_i)/\partial\boldsymbol{\theta}'$, where ξ_i stands for the i-th sampled trajectory in the repository $\boldsymbol{D}_\mu$. To obtain (8), we assume that, in the GP model for $F(\xi)$, the *prior mean* of $F(\xi)$ is 0 for any trajectory ξ. Meanwhile the *kernel function* below specifies the *prior covariance* in between any two trajectories ξ_1 and ξ_2.

$$k_{\boldsymbol{\theta}'}(\xi, \xi') = \left(\frac{\partial \log Pr_{\boldsymbol{\theta}'}(\xi)}{\partial \boldsymbol{\theta}'}\right)^T \cdot \boldsymbol{G}^{-1}_{\boldsymbol{\theta}'} \cdot \frac{\partial \log Pr_{\boldsymbol{\theta}'}(\xi)}{\partial \boldsymbol{\theta}'} \tag{9}$$

where $\boldsymbol{G}_{\boldsymbol{\theta}'}$ in (9) is a matrix defined as

$$\boldsymbol{G}_{\boldsymbol{\theta}'} = \int_\xi Pr_{\boldsymbol{\theta}}(\xi) \frac{\partial \log Pr_{\boldsymbol{\theta}'}(\xi)}{\partial \boldsymbol{\theta}'} \left(\frac{\partial \log Pr_{\boldsymbol{\theta}'}(\xi)}{\partial \boldsymbol{\theta}'}\right)^T \mathrm{d}\xi \tag{10}$$

Clearly when $\boldsymbol{\theta}' = \boldsymbol{\theta}$, $\boldsymbol{G}_{\boldsymbol{\theta}'}$ in (10) becomes the standard *Fisher information matrix*. Similar to [1], we can approximate $\boldsymbol{G}_{\boldsymbol{\theta}'}$ through μ randomly sampled trajectories stored in $\boldsymbol{D}_\mu$ as

$$\boldsymbol{G}_{\boldsymbol{\theta}'} \approx \frac{1}{\mu} \sum_{\xi \in \boldsymbol{D}_\mu} \frac{\partial \log Pr_{\boldsymbol{\theta}'}(\xi)}{\partial \boldsymbol{\theta}'} \left(\frac{\partial \log Pr_{\boldsymbol{\theta}'}(\xi)}{\partial \boldsymbol{\theta}'}\right)^T \tag{11}$$

Based on the kernel function definition in (9), $\boldsymbol{C}_{\boldsymbol{\theta}'}(\boldsymbol{D}_\mu)$ in (8) is further defined below

$$\boldsymbol{C}_{\boldsymbol{\theta}'}(\boldsymbol{D}_\mu) = (\boldsymbol{K}_{\boldsymbol{\theta}'}(\boldsymbol{D}_\mu) + \sigma \cdot \boldsymbol{I})^{-1} \\ [\boldsymbol{K}_{\boldsymbol{\theta}'}(\boldsymbol{D}_\mu)]_{i,j} = k_{\boldsymbol{\theta}'}(\xi_i, \xi_j), 1 \leq i, j \leq \mu \tag{12}$$

Different from [7, 8], the policy gradient in (6) can be estimated in (8) based on trajectories randomly sampled by following arbitrary policies. To solve our constrained optimization problem introduced previously, using (8) alone is insufficient. Particularly, in order to handle the constraint in (4), based on a given trajectory repository $\boldsymbol{D}_\mu$, we can redefine D^1_{KL} as

$$D^1_{KL} = \frac{1}{\mu T} \sum_{\xi \in \boldsymbol{D}_\mu} \sum_{i=1}^{T} \sum_a \pi_{\boldsymbol{\theta}}(s_i, a) \log \frac{\pi_{\boldsymbol{\theta}}(s_i, a)}{\pi_{\boldsymbol{\theta}'}(s_i, a)} \tag{13}$$

where the summation runs through each state s_i visited by following every trajectory ξ stored in $\boldsymbol{D}_\mu$. Adopting the technique introduced in [26], assuming that $\boldsymbol{\theta}' \approx \boldsymbol{\theta}$, a quadratic approximation of D^1_{KL} in (13) can be exploited to further determine suitable update to policy parameters. Specifically,

$$D^1_{KL} \approx \frac{1}{2}(\boldsymbol{\theta}' - \boldsymbol{\theta})^T \boldsymbol{A} (\boldsymbol{\theta}' - \boldsymbol{\theta}) \tag{14}$$

where $\boldsymbol{A}$ is a matrix with $\boldsymbol{A}_{i,j} = \frac{\partial}{\partial \boldsymbol{\theta}'_i}\frac{\partial}{\partial \boldsymbol{\theta}'_j} D^1_{KL}|_{\boldsymbol{\theta}'=\boldsymbol{\theta}}$. If we also approximate the improvement to $L_{\boldsymbol{\theta}}(\boldsymbol{\theta}')$ linearly based on (8), a simplified constrained policy optimization problem can be obtained as

$$\begin{aligned} \max_{\boldsymbol{\theta}'} \quad & (\boldsymbol{\theta}' - \boldsymbol{\theta})^T \left[\boldsymbol{U}_{\boldsymbol{\theta}'} \boldsymbol{C}_{\boldsymbol{\theta}'}(\boldsymbol{D}_\mu)\boldsymbol{Y}|_{\boldsymbol{\theta}'=\boldsymbol{\theta}}\right] \\ s.t. \quad & (\boldsymbol{\theta}' - \boldsymbol{\theta})^T \boldsymbol{A}(\boldsymbol{\theta}' - \boldsymbol{\theta}) \le \delta \end{aligned} \tag{15}$$

Upon solving (15), the learning rule below can be derived for policy update.

$$\boldsymbol{\theta}' = \boldsymbol{\theta} + \frac{1}{g}\boldsymbol{A}^{-1}\boldsymbol{U}_{\boldsymbol{\theta}'}\boldsymbol{C}_{\boldsymbol{\theta}'}(\boldsymbol{D}_\mu)\boldsymbol{Y} = \boldsymbol{\theta} + \frac{1}{g}\boldsymbol{\Delta} \tag{16}$$

with

$$\frac{1}{g} = \sqrt{\frac{2\delta}{\boldsymbol{\Delta}^T \boldsymbol{A}\boldsymbol{\Delta}}}$$

With the help of (16), the CBEM algorithm can be finally constructed as summarized in Algorithm 1. When implementing CBEM, we follow exactly the technique used in TRPO to efficiently compute the *Fisher-vector product* of the form $\boldsymbol{A} \cdot \boldsymbol{\phi}$ where $\boldsymbol{\phi}$ is an arbitrary vector. Subsequently $\boldsymbol{\Delta}$ in (16) can be determined straightforwardly by using the conjugate gradient method with up to 10 iterations [26].

Algorithm 1 The Constrained Bayesian Expectation-Maximization Algorithm for RL

```
Require:
    $\delta$: a predefined step size
    $\mu$: number of trajectories used for training
    $b$: batch size
    $T$: max length of a sampled trajectory
1: repeat for a given number of iterations:
2:     Use $\pi_{\boldsymbol{\theta}}$ to sample $n = \frac{b}{T}$ new trajectories and update repository $\boldsymbol{D}_\mu$
3:     Compute the Bayesian policy gradient in (8)
4:     Use the conjugate gradient method to determine $\boldsymbol{\Delta}$ in (16)
5:     Set $\alpha = 1.0$
6:     repeat while $D^1_{KL} \ge \delta$:
           $\boldsymbol{\theta}' = \boldsymbol{\theta} + \alpha\sqrt{\frac{2\delta}{\boldsymbol{\Delta}^T\boldsymbol{A}\boldsymbol{\Delta}}}\boldsymbol{\Delta}; \alpha = 0.8 \times \alpha$
```

IV. A Lagrangian Expectation-Maximization Algorithm for Reinforcement Learning

Instead of handling the constraint in (4), the job of maximizing the performance lower bound $L_{\boldsymbol{\theta}}(\boldsymbol{\theta}')$ in (3) can be subject to an alternative constraint in (5) at the trajectory level. In an attempt to solve this constrained optimization problem, we can build a *Lagrangian* of the problem as shown below.

$$\begin{aligned} \mathcal{L} &= \int_\xi Pr_{\boldsymbol{\theta}}(\xi)R(\xi)\log Pr_{\boldsymbol{\theta}'}(\xi)\mathrm{d}\xi \\ &\quad - \lambda\left(\int_\xi Pr_{\boldsymbol{\theta}'}(\xi)\log\frac{Pr_{\boldsymbol{\theta}'}(\xi)}{Pr_{\boldsymbol{\theta}}(\xi)}\mathrm{d}\xi - \delta\right) \\ &= \int_\xi I(\xi)\mathrm{d}\xi \end{aligned} \tag{17}$$

where λ is the *Lagrange multiplier* and

$$\begin{aligned} I(\xi) =& Pr_{\boldsymbol{\theta}}(\xi)R(\xi)\log Pr_{\boldsymbol{\theta}'}(\xi) - \lambda \cdot Pr_{\boldsymbol{\theta}'}(\xi)\log Pr_{\boldsymbol{\theta}'}(\xi) \\ &+ \lambda \cdot Pr_{\boldsymbol{\theta}'}(\xi)\log Pr_{\boldsymbol{\theta}}(\xi) + \mathrm{C} \end{aligned} \tag{18}$$

If we treat $\mathcal{L}$ in (18) as a functional of $Pr_{\boldsymbol{\theta}'}(\xi)$ with respect to every possible trajectory ξ, the stationary point (i.e. optimal solution) of $\mathcal{L}$ can be determined through the Euler-Lagrange equation below:

$$\begin{aligned} \frac{\partial I(\xi)}{\partial Pr_{\boldsymbol{\theta}'}(\xi)} =& \frac{Pr_{\boldsymbol{\theta}}(\xi)R(\xi)}{Pr_{\boldsymbol{\theta}'}(\xi)} - \lambda - \lambda \cdot \log Pr_{\boldsymbol{\theta}'}(\xi) \\ &+ \lambda \cdot \log Pr_{\boldsymbol{\theta}}(\xi) \\ =& 0 \end{aligned} \tag{19}$$

By solving the equation above subject to $\int_\xi Pr_{\boldsymbol{\theta}'}(\xi)\mathrm{d}\xi = 1$, we have

$$Pr^*_{\boldsymbol{\theta}'}(\xi) \propto \frac{Pr_{\boldsymbol{\theta}}(\xi)R(\xi)}{\lambda \cdot W\left(\frac{Pr_{\boldsymbol{\theta}}(\xi)R(\xi)e^{1-\log Pr_{\boldsymbol{\theta}}(\xi)}}{\lambda}\right)} \tag{20}$$

where $W(\cdot)$ refers to the standard Lambert-W function and $Pr^*_{\boldsymbol{\theta}'}(\xi)$ gives the optimal probability for a RL agent to produce trajectory ξ upon using the trained parametric policy $\pi_{\boldsymbol{\theta}'}$. Guided by (20), in practice policy parameters can be updated to approximate $Pr^*_{\boldsymbol{\theta}'}(\xi)$ over a collection of μ sampled trajectories in repository $\boldsymbol{D}_\mu$. Particularly, for any trajectory $\xi \in \boldsymbol{D}_\mu$, since

$$\frac{Pr^*_{\boldsymbol{\theta}'}(\xi)}{Pr_{\boldsymbol{\theta}}(\xi)} \propto \frac{\prod_{t=0}^{T-1}\pi_{\boldsymbol{\theta}'}(s_t, a_t)}{\prod_{t=0}^{T-1}\pi_{\boldsymbol{\theta}}(s_t, a_t)} \tag{21}$$

Therefore

$$\prod_{t=0}^{T-1}\pi_{\boldsymbol{\theta}'}(s_t, a_t) \propto \frac{R(\xi)\prod_{t=0}^{T-1}\pi_{\boldsymbol{\theta}}(s_t, a_t)}{\lambda \cdot W\left(\frac{R(\xi)e}{\lambda}\right)} \tag{22}$$

Clearly, when $\lambda \to \infty$,

$$\lim_{\lambda\to\infty}\prod_{t=0}^{T-1}\pi_{\boldsymbol{\theta}'}(s_t, a_t) \propto \prod_{t=0}^{T-1}\pi_{\boldsymbol{\theta}}(s_t, a_t)$$

It demonstrates that the Lagrange multiplier λ controls the behavioral deviation of policy $\pi_{\boldsymbol{\theta}'}$ from $\pi_{\boldsymbol{\theta}}$. Specifically, with a sufficiently large value for λ, this deviation can be constrained to a low level, ideal for reliable RL. Taking the natural logarithm at both sides of (22), we have

$$\sum_{t=0}^{T-1} \log \pi_{\boldsymbol{\theta}'}(s_t, a_t) = -\log \lambda + \sum_{t=0}^{T-1} \log \pi_{\boldsymbol{\theta}}(s_t, a_t) + \log\left(R(\xi)/W\left(\frac{R(\xi)e}{\lambda}\right)\right) + \Psi \tag{23}$$

Based on (23), we can introduce a new *loss function* defined over μ sampled trajectories stored in $\boldsymbol{D}_\mu$ as

$$\sum_{\xi \in \boldsymbol{D}_\mu} \left(\begin{array}{c} \sum_{t=0}^{T-1} \log \pi_{\boldsymbol{\theta}'}(s_t, a_t) - \sum_{t=0}^{T-1} \log \pi_{\boldsymbol{\theta}}(s_t, a_t) \\ -\log\left(R(\xi)/W\left(\frac{R(\xi)e}{\lambda}\right)\right) + \log \lambda - \Psi \end{array} \right)^2 \tag{24}$$

where Ψ in (23) serves as a normalization factor that reduces the loss function in (24) to its minimum, Hence

$$\Psi = \frac{\sum_{\xi \in \boldsymbol{D}_\mu} \left(\begin{array}{c} \sum_{t=0}^{T-1} \log \pi_{\boldsymbol{\theta}'}(s_t, a_t) - \sum_{t=0}^{T-1} \log \pi_{\boldsymbol{\theta}}(s_t, a_t) \\ -\log\left(R(\xi)/W\left(\frac{R(\xi)e}{\lambda}\right)\right) + \log \lambda \end{array} \right)}{\mu} \tag{25}$$

Similar to the development of the CBEM algorithm, assuming that $\boldsymbol{\theta}' \approx \boldsymbol{\theta}$, to minimize the loss function in (24), we can employ the learning rule below for policy update.

$$\begin{aligned} \boldsymbol{\theta}' =& \boldsymbol{\theta} - \alpha \sum_{\xi \in \boldsymbol{D}_\mu} \frac{\partial \log Pr_{\boldsymbol{\theta}'}(\xi)}{\partial \boldsymbol{\theta}'} \cdot \\ & \left(\log \lambda - \log\left(R(\xi)/W\left(\frac{R(\xi)e}{\lambda}\right)\right) - \Psi\right) \\ =& \boldsymbol{\theta} + \alpha \sum_{\xi \in \boldsymbol{D}_\mu} \frac{\partial \log Pr_{\boldsymbol{\theta}'}(\xi)}{\partial \boldsymbol{\theta}'} \hat{R}(\xi, \lambda) \end{aligned} \tag{26}$$

where α is a pre-determined learning rate. Practical and effective use of (26) requires a suitable value for λ. As summarized in Algorithm 2, we adopt a simple strategy in LEM to search through an exponentially increasing sequence of possible values for λ, starting from a reasonably small value such as 1.0. For each candidate value of λ, once $\boldsymbol{\theta}'$ is obtained from (26), we will check the violation of the constraint in (4). Notice that the constraint in (4) is used instead of the constraint in (5) in this heuristic search process because (5) cannot be calculated in practice without knowing the state-transition model of the learning environment. Intuitively, when D^1_{KL} in (4) is reasonably small, we can expect D^2_{KL} in (5) to be small as well. If the constraint $D^1_{KL} \leq \delta$ is violated, λ will be increased by a factor of 10.0 in our experiments. The increased λ will be further utilized to determine a new update to policy parameters based on (26). This procedure will repeat until either (4) is satisfied or the maximum number of search steps (10 in our experiments) is reached.

In view of the learning rules formulated respectively in (16) and (26), it is apparent to see that they are closely related to the (slightly modified) learning rule of the REINFORCE algorithm shown in (27). Specifically, if the kernel matrix $\boldsymbol{C}_{\boldsymbol{\theta}'}(\boldsymbol{D}_\mu)$ in (16) reduces to an identity matrix, then (16) in fact presents the natural policy gradient variation of (27). Meanwhile, policy update in (26) is achieved through modified reward function $\hat{R}(\xi, \lambda)$ rather than $R(\xi)$ in (27). Based on this understanding, (16) and (26) can be considered as two separate extensions of REINFORCE, obtained respectively by handling two different constraints, one based on D^1_{KL} and the other based on D^2_{KL}.

$$\boldsymbol{\theta}' = \boldsymbol{\theta} + \alpha \sum_{\xi \in \boldsymbol{D}_\mu} \frac{\partial \log Pr_{\boldsymbol{\theta}}(\xi)}{\partial \boldsymbol{\theta}} R(\xi, \lambda) \tag{27}$$

Algorithm 2 The Lagrangian Expectation-Maximization Algorithm for RL

```
Require:
    $\delta$: a predefined step size
    $\alpha$: learning rate
    $\mu$: number of trajectories used for training
    $b$: batch size
    $T$: max length of a sampled trajectory
1: repeat for a given number of iterations:
2:     Use $\pi_{\boldsymbol{\theta}}$ to sample $n = \frac{b}{T}$ new trajectories and update
         repository $\boldsymbol{D}_\mu$
3:     Set $\lambda = 1.0$
4:     repeat while $D^1_{KL} \geq \delta$:
5:         Calculate $\boldsymbol{\theta}'$ according to (26)
6:         $\lambda = 10.0 \times \lambda$
```

V. Empirical Analysis

In this section, we experimentally evaluate CBEM and LEM on three benchmark problems explained below. Three well-known RL algorithms, namely REINFORCE, DQN [19] and TRPO with generalized advantage estimation, have been chosen for comparison. Both DQN and TRPO were proposed recently to solve RL problems based on DNN models. On the other hand, REINFORCE is studied in this paper due to its close connection with CBEM and LEM. We however did not test other cutting-edge RL algorithms such as the Deep Deterministic Policy Gradient (DDPG) algorithm [17] which is designed to learn policies that produce continuous rather than discrete actions. To ensure a fair comparison, we implement CBEM and LEM in *rllab* [6], a software package that supports several RL algorithms including TRPO.

A. Benchmark RL Problems

In our experiments, we utilize benchmark problems provided by OpenAI GYM [2] due to their wide accessibility and popularity. We focus specifically on control tasks in the free Box2D simulator. A brief description of each problem follows.

- The *Cart Pole* problem is a classic control problem. Its state space consists of four continuous dimensions, whereas its controller is designed to support two discrete actions, i.e. {force applied from left, force applied from right}). The goal is to balance the pole at an upright position while the cart moves horizontally. As long as the pole remains upright, a reward of +1 is given. The problem is defined to be solved whenever the RL agent obtains an average total rewards of 195.0 for 100 consecutive episodes.

- The *Lunar Lander* problem is a robotic control problem. A RL agent is expected to learn a control policy that accepts an 8-dimensional continuous state input and produces one of four discrete actions, i.e. {do nothing, fire left orientation engine, fire main engine, fire right orientation engine}. The goal of the problem is to smoothly and accurately guide a lander to land on a target landing pad. A reward ranging from 100 to 140 will be provided based on the distance that the agent moves from its start point to the landing pad, in consideration also of its landing speed. The main engine costs -0.3 rewards for every frame. Meanwhile, -100 or +100 will be awarded to the situations if the lander crashes or comes to rest. Moreover, with each leg of the lander contacts the ground, +10 will be awarded. The solving condition of the problem requires the agent to achieve average (non-discounted) cumulative rewards of 200 over 100 consecutive episodes.
- The *Bipedal Walker* problem is also a robotic control problem. It is originally constituted of a 24-dimensional continuous state space and a 4-dimensional continuous action space. To facilitate our experiments, we modified the problem by discretizing the action space. The resulting action space contains 81 discrete actions that together enumerate all possible combinations of three alternative values (i.e., -1, 0, +1) over 4 original action dimensions. The objective of the problem remains for a RL agent to learn to walk on a plain landscape in avoidance of accidental falling down. +300 rewards will be given when the agent reaches the end. It will be penalized with -100 while falling down. The use of its monitor torque will cost a few points as well. Due to our change of the action space, we no longer check the problem solving conditions which demand for average total rewards of 300 over 100 consecutive episodes.

B. Experiment Setup

Similar to [26], we choose NNs with two fully connected hidden layers as learning models. Detailed experimental settings have been summarized in Table I. It is worthwhile to note that the step size for TRPO is set to 0.001 instead of 0.01, as reported previously in [26]. This is because we found TRPO performed more effectively with our setting on the benchmark problems.

C. Results and Discussion

In this section, we discuss and analyze experiment results which will be reported through two different learning curves. The first curve depicts the change of average cumulative rewards obtained across a predetermined number learning iterations. The second curve shows the maximum cumulative rewards achieved by at least one trajectory sampled with the trained NNs at every iteration.

Settings	Cart Pole	Lunar Lander	Bipedal Walker
State dimensions	4	8	24
Action dimensions	1	4	81
Number of Hidden Layers	2	2	2
Hidden Layer Size	32×32	150×150	150×150
Hidden Layer Activation Function	tanh	tanh	tanh
Output Layer Activation Function	linear	linear	linear
Output Layer Activation Function	softmax	softmax	softmax
Learning Iterations	200	2000	2000
Batch Size	800	8000	8000
Discount Factor	0.99	0.99	0.99
Step Size (CBEM)	0.0001	0.0001	0.0001
Step Size or Learning Rate (Other Algorithms)	0.001	0.001	0.001
Maximum Length of Sampled Trajectory	200	200	200
Number of Trajectories for Training	20	160	160

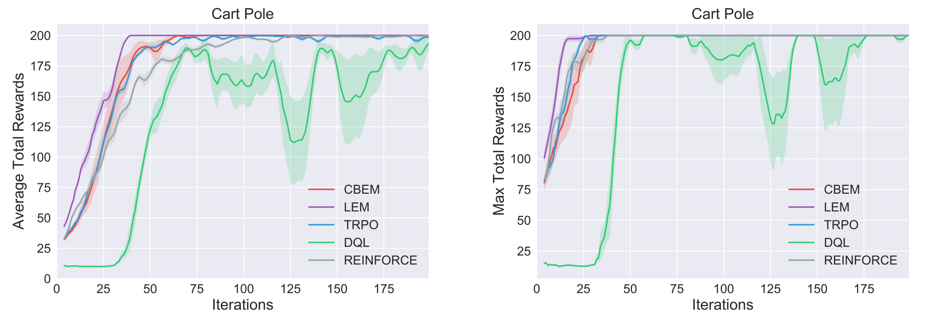
TABLE I: Experiment settings for all algorithms on all benchmark problems.

1) Results on Cart Pole: The classical Cart Pole problem is used to verify the effectiveness of all experimented RL algorithms. The average cumulative rewards graphed in Figure 1 (a) show that all algorithms, except DQN, can solve the problem effectively. The performance of DQN fluctuates significantly during the entire learning process. This is understandable since DQN always uses the epsilon greedy policy with 0.1 exploration rate in our experiments. Meanwhile, LEM appears to converge faster than other competing algorithms because it is more likely to discover high-reward trajectories. In fact, as evidenced in Figure 1 (b), LEM can produce trajectories with the maximum reward of 200.0 after only 18 learning iterations. It is also interesting to note that CBEM and TRPO achieved similar performance on the Cart Pole problem.

2) Results on Lunar Lander: Figure 2 shows that only CBEM, LEM and TRPO can solve the Lunar Lander problem satisfactorily since the problem is more challenging than the Cart Pole problem. Due to the same reason mentioned a priori, DQN failed to solve the problem by using an exploratory policy but the algorithm managed to find high-reward trajectories (the maximum cumulative rewards are over 200.0), as evidenced in Figure 2 (b). Meanwhile REINFORCE fell apparently behind other algorithms in both Figure 2 (a) and Figure 2 (b).

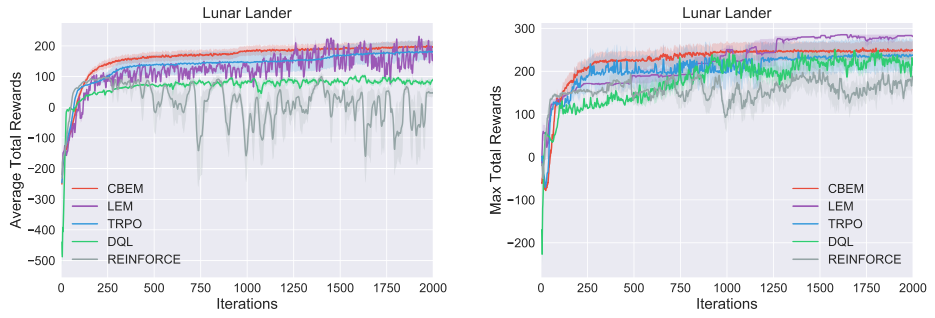
We also notice that CBEM consistently outperforms its competitors in terms of the average cumulative rewards obtained. In the meantime, despite of more oscillating behaviors, LEM exhibited comparable performance as TRPO. Since LEM is closely related to REINFORCE, our results suggest that, by constraining policy behavioral changes during learning, we can achieve more effective and reliable learning outcome. The NNs trained through LEM can produce trajectories with prominently higher cumulative rewards (approx. 250.2) than other algorithms as well.

3) Results on Bipedal Walker: As can be seen in Figure 3 (a), the learning performance can be improved continuously over 2000 iterations while using CBEM. Particularly, after 1700 iterations, CBEM surpasses all other algorithms in terms of the obtained average cumulative rewards. In the meantime, LEM achieved competitive performance as TRPO and can also produce trajectories with clearly higher maximum cumulative



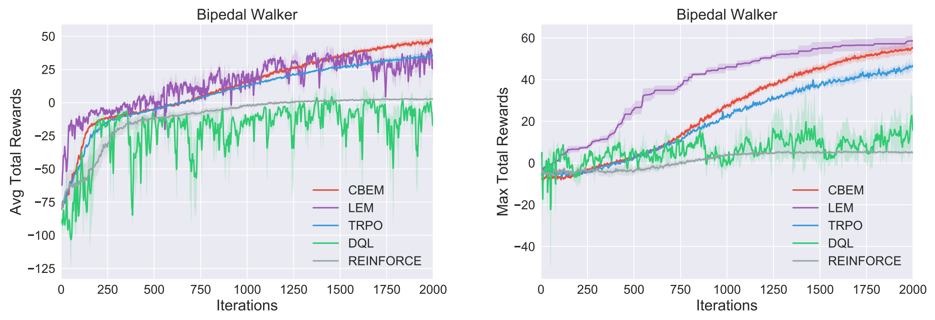
(a) (b)

Fig. 1: The average/maximum long-term cumulative rewards achieved per iteration by CBEM, LEM, TRPO, DQN and REINFORCE on the Cart Pole problem.



(a) (b)

Fig. 2: The average/maximum long-term cumulative rewards achieved per iteration by CBEM, LEM, TRPO, DQN and REINFORCE on the Lunar Lander problem.



(a) (b)

Fig. 3: The average/maximum long-term cumulative rewards achieved per iteration by CBEM, LEM, TRPO, DQN and REINFORCE on the Bipedal Walker problem.

rewards. Finally it is to be noted that both DQN and REINFORCE are comparatively less effective on the Bipedal Walker problem.

4) Summary and Discussion: Experimental evidences provided above indicate that our proposed algorithms, i.e. CBEM and LEM, can perform consistently well on all the three benchmark problems. Compatible with the findings reported in [26], we found that optimizing a performance lower bound can be an effective approach for RL. In particular, our algorithms achieved highly competitive performance without requiring any value function estimation techniques.

VI. Conclusions

In this paper we aimed at developing policy-oriented PS algorithms for effective RL. Motivated by the fact that effective search of parametric policies can often be achieved through direct optimization of a performance lower bound, we decided to maximize specifically a bound obtained originally based on an Expectation-Maximization method due to its flexibility and wide applicability. Moreover, for the purpose of improving learning reliability and further enhancing learning performance, we studied two separate policy behavioral constraints measured through KL divergence. Driven by our constraint policy optimization problems, we have successfully developed two new RL algorithms, i.e. CBEM and LEM. Both algorithms have been experimentally examined on three benchmark problems provided by OpenAI GYM with highly competitive results, in comparison to three well-known RL algorithms.

Looking into the future, it is interesting to study the effectiveness of CBEM and LEM on solving a wide range of RL problems including computer game play problems as well as multi-task RL problems. It is also interesting to investigate the possible combined use of both CBEM and LEM since the modified reward function introduced by LEM in (26) can be naturally adopted in the GP model used by CBEM to estimate the cumulative rewards of arbitrary trajectories.

References

[1] S. Bhatnagar et al. "Natural actor-critic algorithms". In: *Journal Automatica* 45.11 (2009), pp. 2471–2482.

[2] Greg Brockman et al. *OpenAI Gym*. 2016. eprint: arXiv: 1606.01540.

[3] G. Chen, C. Douch, and M. Zhang. "Using Learning Classifier Systems to Learn Stochastic Decision Policies". In: *IEEE Transactions on Evolutionary Computation* 19.6 (2015), pp. 885–902.

[4] P. Dayan and G. E. Hinton. "Using expectation-maximization for reinforcement learning". In: *Neural Computation* 9.2 (1997), pp. 271–278.

[5] M. Deisenroth and C. E. Rasmussen. "PILCO: A model-based and data-efficient approach to policy search". In: *Proceedings of the 28th International Conference on machine learning (ICML-11)*. 2011, pp. 465–472.

[6] Y. Duan et al. "Benchmarking deep reinforcement learning for continuous control". In: *Proceedings of the 33rd International Conference on Machine Learning (ICML)*. 2016.

[7] Y. Engel and M. Ghavamzadeh. "Bayesian policy gradient algorithms". In: *Proceedings of the 2006 conference on advances in neural information processing systems*. Vol. 19. 2007, p. 457.

[8] R. A. Ghanea-Hercock, F. Wang, and Y. Sun. "Self-Organizing and Adaptive Peer-to-Peer Network". In: *IEEE Transactions on Systems, Man, and Cybernetics–Part B* 36.6 (2006), pp. 1230–1236.

[9] M. Ghavamzadeh, Y. Engel, and M. Valko. "Bayesian policy gradient and actor-critic algorithms". In: *Journal of Machine Learning Research* 17.66 (2016), pp. 1–53.

[10] S. Gu et al. "Continuous Deep Q-Learning with Model-based Acceleration". In: *Proceedings of the 33rd International Conference on Machine Learning*. 2016.

[11] S. Gu et al. "Interpolated Policy Gradient: Merging On-Policy and Off-Policy Gradient Estimation for Deep Reinforcement Learning". In: *arXiv preprint arXiv:1706.00387* (2017).

[12] N. Heess et al. "Learning Continuous Control Policies by Stochastic Value Gradients". In: *Advances in Neural Information Processing Systems 28*. 2015.

[13] Jens J. Kober, J. A. Bagnell, and J. Peters. "Reinforcement learning in robotics: A survey". In: *The International Journal of Robotics Research* (2013).

[14] J. Kober and J. Peters. "Policy search for motor primitives in robotics". In: *Advances in neural information processing systems*. 2009, pp. 849–856.

[15] P. Kormushev, S. Calinon, and D. G. Caldwell. "Robot motor skill coordination with EM-based reinforcement learning". In: *The 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2010, pp. 3232–3237.

[16] S. Levine and V. Koltun. "Guided Policy Search." In: *ICML (3)*. 2013, pp. 1–9.

[17] T. P. Lillicrap et al. "Continuous control with deep reinforcement learning". In: *arXiv.org* (Sept. 2015). arXiv: 1509.02971v5 `[cs.LG]`.

[18] V. Mnih et al. "Asynchronous methods for deep reinforcement learning". In: *Proceedings of the 33rd International Conference on Machine Learning*. 2016.

[19] V. Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* 518 (2015).

[20] A. O'Hagan. "Bayes-Hermite quadrature". In: *Journal of Statistical Planning and Inference* 29.3 (1991), pp. 245–260.

[21] J. Peters. "Policy gradient methods". In: *Scholarpedia* 11.5 (2010).

[22] J. Peters and S. Schaal. "Natural Actor-Critic". In: *Neurocomputing* (2008), pp. 1180–1190.

[23] J. Peters and S. Schaal. "Reinforcement learning by reward-weighted regression for operational space control". In: *Proceedings of the 24th international conference on Machine learning*. 2007.

[24] Jan Peters, Katharina Mülling, and Yasemin Altün. "Relative Entropy Policy Search." In: *AAAI*. Atlanta. 2010, pp. 1607–1612.

[25] J. Schulman et al. "High-Dimensional Continuous Control Using Generalized Advantage Estimation". In: *Proceedings of the 4th International Conference on Learning Representations*. 2016.

[26] J. Schulman et al. "Trust region policy optimization". In: *Proceedings of the 32nd International Conference on Machine Learning*. 2015.

[27] R. S. Sutton et al. "Policy Gradient Methods for Reinforcement Learning with Function Approximation". In: *Advances in Neural Information Processing Systems 12 (NIPS 1999)*. MIT Press, 2000, pp. 1057–1063.

[28] R. J. Williams. "Simple statistical gradient-following algorithms for connectionist reinforcement learning". In: *Machine learning* 8.3-4 (1992), pp. 229–256.