Artificial Intelligence Review manuscript No.
(will be inserted by the editor)

Importance Sampling Policy Gradient Algorithms in Reproducing Kernel Hilbert Space

Tuyen Pham Le · Vien Anh Ngo · P.Marlith Jaramillo · TaeChoong Chung

Received: date / Accepted: date

Abstract Modeling policies in Reproducing Kernel Hilbert Space (RKHS) offers a very flexible and powerful new family of policy gradient algorithms called RKHS policy gradient algorithms. They are designed to optimize over a space of very high or infinite dimensional policies. As a matter of fact, they are known to suffer from a large variance problem. This critical issue comes from the fact that updating the current policy is based on a functional gradient that does not exploit all old episodes sampled by previous policies. In this paper, we introduce a generalized RKHS policy gradient algorithm that integrates the following important ideas: i) policy modeling in RKHS; ii) normalized importance sampling, which helps reduce the estimation variance by reusing previously sampled episodes in a principled way; and iii) regularization terms, which avoid updating the policy too over-fit to sampled data. In the experiment section, we provide an analysis of the proposed algorithms through bench-marking domains. The experiment results show that the proposed algorithm can still enjoy a powerful policy modeling in RKHS and achieve more data-efficiency.

Keywords Reproducing Kernel Hilbert Space · Policy Search · Reinforcement Learning · Importance Sampling · Policy Gradient · Non-parametric

Tuyen Pham Le
Artificial Intelligence Lab, Computer Science and Engineering Department, Kyung Hee University, Yongin, Gyeonggi, 446-701, South Korea. E-mail: tuyenple@khu.ac.kr

Vien Anh Ngo
Machine Learning and Robotics Lab, University of Stuttgart, Germany. E-mail: vien.ngo@ipvs.uni-stuttgart.de

P.Marlith Jaramillo
Artificial Intelligence Lab, Computer Science and Engineering Department, Kyung Hee University, Yongin, Gyeonggi, 446-701, South Korea. E-mail: marlith_jdm@ieee.org

TaeChoong Chung
Artificial Intelligence Lab, Computer Science and Engineering Department, Kyung Hee University, Yongin, Gyeonggi, 446-701, South Korea.
Tel.: +82 031-201-2569
Fax: +82 031-201-1723
E-mail: tcchung@khu.ac.kr

1 Introduction

Policy search is a powerful approach in reinforcement learning (RL) for learning on continuous and high-dimensional tasks such as robotics (we refer the interested reader to a survey paper by (Deisenroth et al., 2013)). Many policy search algorithms have been proposed in literature and have shown many successes. For example, the episodic natural actor-critic (eNAC) algorithm (Peters and Schaal, 2008a) in which natural gradients are analytically computed, the Weighting Exploration with the Returns (PoWER) algorithm (Kober and Peters, 2011) in which the policy variance representing exploration is dependent on states and is updated using weighted returns, the Cost-Regularized Kernel Regression (CRKR) algorithm (Kober et al., 2011), Trust Region Policy Optimization (TRPO) (Schulman et al., 2015), Relative Entropy Search (REPS) (Peters et al., 2010) and Hierarchical REPS (Daniel et al., 2012) in which the policy update is derived by constraining the information loss, and policy search techniques based on variational inference (Neumann, 2011; Levine and Koltun, 2013b).

Policy gradient is one of the policy search approaches which use gradient-descent to update the policy. Many policy gradient methods use very different techniques to estimate gradients, such as finite difference (Peters and Schaal, 2008b) using linear regression, REINFORCE (Williams, 1992) using likelihood ratio ideas from statistics, policy gradient theorem based on the Markov property to derive analytic gradients (Sutton et al., 1999; Baxter et al., 2001), and natural gradient based on Riemannian geometry (Peters and Schaal, 2008a). However, these methods must rely on a parametric policy which is often represented by a span of pre-defined features such as linear policies, radial basis functions, and dynamic movement primitives (Deisenroth et al., 2013). The use of pre-defined features makes policies rigid while representing a domain's space. If good features are chosen, the parametric policy might obtain good performance. However, in practice, choosing good features is difficult due to spaces of very high or infinite dimensional policies. There is a naive fix to this problem by considering all weights and centers of the features as parameters. Optimizing the weights and centers can be handled by analyzing their respective derivatives. However, this approach does not represent a principled way due to the problem of finding a suitable scaling metric in a combined parameter space which is discussed in (Lever and Stafford, 2015).

Recently, there has been a growing interest in modeling policies in reproducing kernel Hilbert spaces (Bagnell and Schneider, 2003; Lever and Stafford, 2015; Vien et al., 2016) that does not depend on pre-defined features, therefore alleviates the dependencies of policy representation on domain knowledge. Lever and Stafford (Lever and Stafford, 2015) proposed a method, called Compatible RKHS Actor-Critic, that uses non-parametric policy modeling in RKHS and represents both the gradients of the policy and the value functions in RKHS. Policy modeling in RKHS can enable policy optimization in a very high or infinite dimensional space but still enjoys compact and sparse representation. Though RKHS Actor-Critic has shown many potentials and performed very well in many complex tasks, it is still an on-policy method that handles policy improvement using only episodes generated from the current policy. Therefore, it is known to suffer from a large variance problem.

One solution for the above issue is to use off-policy techniques. Specifically, they try to exploit trajectories collected from other policies to compute gradients while updating the current policy. Among this methods, importance sampling (Bishop, 2006) is a useful technique that has been used in many RL algorithms. It can also be combined with many other methods to achieve a trade-off between the variance and the bias in the gradient estimation, such as adding eligibility trace (Precup et al., 2000), truncating (Wawrzyński, 2009), flattening (Hachiya et al., 2009), normalizing (Shelton, 2001), or adding a baseline (Zhao et al., 2013).

Unfortunately, off-policy algorithms have one major limitation. They are based on the density ratios of time steps in a trajectory but do not take the magnitude of these densities into account. This implies that when the ratios are peaky at a few time steps, the estimator would return a premature convergent policy. Therefore, this estimator has a very poor generalization ability. This issue is presented in the study (Levine and Koltun, 2013a) in which the authors propose to use a regularization term in the form of the expected discounted return.

In this paper, we introduce an efficient policy search algorithm based on policy gradient in RKHS. The contributions are two-fold. First, we took advantage of non-parametric policies in RKHS and employed importance sampling techniques to reuse previous trajectories in order to reduce the estimation variance. The importance weights in RKHS were computed analytically. Second, we integrated a regularization term in order to render a more efficient importance sampling RKHS policy search algorithm. In fact, regularization is expected to offer a better ability to generalize. To the best of our knowledge, this paper is the first to integrate regularized importance sampling into an RKHS policy search framework.

The rest of the paper is organized as follows. Section 2 reviews underlying knowledge such as markov decision process and policy gradient algorithms. Section 3 introduces components in RKHS, which constitute our proposed algorithms. Our contributions are described in Section 4. The usefulness of the proposed algorithms is demonstrated through many bench-marking RL problems in Section 5.

2 Background

In this section, we first review background on the Markov Decision Process, then describe policy gradient methods.

2.1 Markov Decision Process

We assume that the underlying problem in RL is modeled as a discrete-time Markov Decision Process (MDP) $\{\mathcal{S}, \mathcal{A}, \mathcal{P}, r\}$. MDP consists of a state space $\mathcal{S}$, an action space $\mathcal{A} \in Re^m$ (m-dimensional action space), a transition function $\mathcal{P}(s_{t+1}, s_t, a_t) = p(s_{t+1}|s_t, a_t)$ that measures the probability of obtaining the next state s_{t+1} given a current state-action pair (s_t, a_t) (denoted as ξ_t), $r\,(s_t, a_t)$ defines the immediate reward achieved at each state-action pair, and $\gamma \in (0, 1)$ denotes a discount factor. A characteristic of MDP that distinguishes it from other processes is that the next state only depends on the current state-action pair:

$$p(s_{t+1}|\{s_1, a_1, s_2, a_2, ..., s_t, a_t\}) = p(s_{t+1}|s_t, a_t).$$

Let $\pi(a|s)$ be a stochastic policy that represents the conditional probability of taking an action a given a state s. Assume that a Gaussian policy is used and is of the form

$$\pi_{h,\mathbf{\Sigma}}(a|s) = \frac{1}{\sqrt{2\pi\,|\mathbf{\Sigma}|}} e^{-\frac{1}{2}(a-h(s))^\top \mathbf{\Sigma}^{-1}(a-h(s))}, \tag{1}$$

where $h(s) : \mathcal{S} \to \mathcal{A}$ is a mapping from states to actions, $\mathbf{\Sigma}$ is a co-variance matrix in $\mathbb{R}^{m\times m}$, and $|\mathbf{\Sigma}|$ is a determinant. Parametric methods would design $h(s)$ to depend on a set of sequential fixed points called features. If linear functions are used, the function h could be written as $h(s) = \boldsymbol{\theta}^\top \phi(s)$, where $\boldsymbol{\theta} \in \Re$ is a policy parameter vector, and $\phi(s)$ is a feature function. For this setting, we denote $\pi^{\boldsymbol{\theta}}$ a parametric policy.

For the non-parametric setting, $h(s)$ takes advantage of the reproducing property of RKHS, which we will discuss in Section 3.2.

Let $\xi = \{\xi_1, \xi_2, ..., \xi_t\}$ be a trajectory (also called an episode, history or roll-out). Then, the discounted cumulative reward of a trajectory is given by

$$R(\xi) = \sum_{t=0}^{T} \gamma^t r(s_t, a_t).$$

A control algorithm in RL tries to find an optimal policy π^* in order to maximize the expected total discounted reward as follows:

$$J(\pi) = \mathbb{E}[R(\xi)] = \int p(\xi|\pi) R(\xi) d\xi, \tag{2}$$

where $p(\xi|\pi)$ is the probability of generating a trajectory ξ given the policy π. It can be computed using multiplication rule

$$p(\xi|\pi) = p(s_0) \prod_{t=0}^{T-1} p(s_{t+1}|s_t, a_t)\pi(a_t|s_t). \tag{3}$$

where $p(s_0)$ is the starting state distribution.

2.2 Policy Gradient Algorithms

Policy gradient is one of the popular policy search methods that optimizes a policy $\pi^{\boldsymbol{\theta}}$ using gradient ascent. It updates $\boldsymbol{\theta}$ along the gradient direction of the expected return as

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \alpha \nabla_{\boldsymbol{\theta}} J(\pi(\boldsymbol{\theta}_k)),$$

where α denotes a learning rate and k is the current update number. Policy gradient methods guarantee convergence to an optimum (local optimum). The main computational challenge of a policy gradient method is to estimate the gradient from given trajectories such that its variance is small.

The likelihood ratio method (Williams, 1992) (LR) is a policy gradient method. It computes the gradient of the expected return by using the objective function shown in Eq. 2 and the definition of the trajectory distribution shown in Eq. 3 as:

$$\nabla_{\boldsymbol{\theta}} J(\pi) = \int \nabla_{\boldsymbol{\theta}} p(\xi|\pi) R(\xi) d\xi = \int p(\xi|\pi) \nabla_{\boldsymbol{\theta}} \log p(\xi|\pi) R(\xi) d\xi \tag{4}$$

$$= \int p(\xi|\pi) \sum_{t=1}^{T} \nabla_{\boldsymbol{\theta}} \log \pi(a_t|s_t) R(\xi) d\xi, \tag{5}$$

where the "log trick" $\nabla_{\boldsymbol{\theta}} p(\xi|\pi) = p(\xi|\pi)\nabla_{\boldsymbol{\theta}} \log p(\xi|\pi)$ is used, and the derivative of $\log p(\xi|\pi)$ becomes the sum of terms $\nabla_{\boldsymbol{\theta}} \log p(s_{t+1}|s_t, a_t)$ and $\nabla_{\boldsymbol{\theta}} \log \pi(a_t|s_t)$. The first terms must be zero because the transition function is independent from the policy parameters, therefore

$$\nabla_{\boldsymbol{\theta}} \log p(\xi|\pi) = \sum_{t=1}^{T} \nabla_{\boldsymbol{\theta}} \log \pi(a_t|s_t).$$

Since $p(\xi|\pi)$ is unknown, the integral in Eq. 5 can be estimated via Monte-Carlo simulations by averaging N trajectories generated from the policy $\pi^{\boldsymbol{\theta}}$ as:

$$\nabla_{\boldsymbol{\theta}} \widehat{J}(\pi) = \frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T} \nabla_{\boldsymbol{\theta}} \log \pi(a_t^n|s_t^n) R(\xi^n), \tag{6}$$

where ξ^n is the n^{th} trajectory.

Another well-known policy gradient method is the Policy Gradient Theorem (Sutton et al., 1999) (PGT). PGT is based on the intuitive observation that future actions do not depend on past rewards. PGT is able to reduce variance in policy gradient estimation. The policy gradient of PGT is given by:

$$\nabla_{\boldsymbol{\theta}} J(\pi) = \int p(\xi|\pi) Q^{\pi_{\boldsymbol{\theta}}}(\xi) \nabla_{\boldsymbol{\theta}} \log \pi(a_t|s_t) d\xi.$$

Similar to the LR method, we can avoid the integral computation by using an empirical average form:

$$\nabla_{\boldsymbol{\theta}} \widehat{J}(\pi) = \frac{1}{N} \sum_{n=1}^{N} \sum_{t=0}^{T} Q^{\pi_{\boldsymbol{\theta}}}(\xi_t^n) \nabla_{\boldsymbol{\theta}} \log \pi(a_t^n|s_t^n), \tag{7}$$

where $Q^{\pi_{\boldsymbol{\theta}}}(\xi_t^n)$ is action-value function at state-action pair ξ_t^n and defined as

$$Q^{\pi_{\boldsymbol{\theta}}}(\xi_t^n) = \sum_{k=t}^{T} r(s_k^n, a_k^n)$$

3 Policy Search in Reproducing Kernel Hilbert Space

Representing functions in RKHS has become a powerful tool in many applications of machine learning and signal processing (Milanfar, 2013). It allows to switch between a highly nonlinear task and an equivalent linear one while avoiding computational problems (the curse of dimensionality). In RL, there are some studies that model policies in RKHS in order to solve highly nonlinear tasks (Bagnell and Schneider, 2003; Lever and Stafford, 2015; Vien et al., 2016). In these studies, the policy of the RL problem works in RKHS to allow the modeling of highly nonlinear tasks without relying on a priori features, and adaptively finding a compact form. In this section, we briefly describe the background of RKHS and policy gradient methods in RKHS.

3.1 Reproducing Kernel Hilbert Spaces

There are many different definitions and theorems about RKHS. In this paper, we use a vector-valued RKHS (Micchelli and Pontil, 2005) which directly implies the form of non-parametric policies. The definition given in Definition 1 is generalized to both scalar-valued RKHS and vector-valued RKHS.

Definition 1 Let $\mathcal{X}$ be a set, $\mathcal{Y}$ be a Hilbert Space with inner product $\langle \cdot, \cdot \rangle_{\mathcal{Y}}$, and $\mathcal{H}_k$ be a linear space of functions on $\mathcal{X}$ with values in $\mathcal{Y}$ ($\mathcal{H} \subseteq \mathcal{Y}^{\mathcal{X}}$). We call $\mathcal{H}_k$ a Reproducing Kernel Hilbert Space (RKHS) if there exists a kernel function $K : \mathcal{X} \times \mathcal{X} \to \mathcal{L}(\mathcal{Y})$, where $\mathcal{L}(\mathcal{Y})$ is linear operators on $\mathcal{Y}$, with the following two important properties:

- For every $x \in \mathcal{X}$ and $y \in \mathcal{Y}$, $K(\cdot, x)y$ belongs to $\mathcal{H}_k$ (or equivalently K spans $\mathcal{H}_k$, i.e., $\mathcal{H}_k = span\{K(\cdot, x)y, x \in \mathcal{X}, y \in \mathcal{Y}\}$)
- K has the “reproducing property”, i.e., $\langle h(x), y \rangle_{\mathcal{Y}} = \langle h, K(\cdot, x)y \rangle_{\mathcal{H}}$ for all $h \in \mathcal{H}_k$, $x \in \mathcal{X}$, and $y \in \mathcal{Y}$. Further, $\left\langle K(\cdot, x)y, K(\cdot, x')y' \right\rangle_{\mathcal{H}} = \left\langle y, K(x, x')y' \right\rangle_{\mathcal{Y}}$.

When $\mathcal{Y} = \mathbb{R}$ with the Euclidean inner product, RKHS is equivalent to a scalar-valued RKHS (Scholkopf and Smola, 2001), where $K(x, x')$ is a scalar. In the general case that $\mathcal{Y} = \mathbb{R}^n$, $n > 1$, RKHS becomes a vector-valued RKHS (Micchelli and Pontil, 2005) and $K(x, x')$ is an operator-valued kernel, i.e., K is in the form of a matrix. Based on the definition of RKHS above, a linear function $h \in \mathcal{H}_k$ is typically specified by an expansion of the form:

$$h(\cdot) = \sum_i K(x_i, \cdot) y_i .$$

With this form, we can use h as the functional parameter of a non-parametric policy that will be explained below.

3.2 Non-parametric Policy in Reproducing Kernel Hilbert Space

As introduced in Section 2.1, a non-parametric policy in RKHS has the form of a stochastic Gaussian:

$$\pi_{h,\mathbf{\Sigma}}(a|s) = \frac{1}{\sqrt{2\pi\,|\mathbf{\Sigma}|}} e^{-\frac{1}{2}(a-h(s))\mathbf{\Sigma}^{-1}(a-h(s))}, \tag{8}$$

parameterized by the functional $h \in \mathcal{H}_k$, $h : \mathcal{S} \to \mathcal{A} \subseteq \mathbb{R}^m$ and a co-variance matrix $\mathbf{\Sigma}$ on $\mathbb{R}^{m \times m}$. The function $h(\cdot)$ is an element of RKHS $\mathcal{H}_k$ with the following property:

$$h(\cdot) = \sum_i K(s_i, \cdot) a_i \in \mathcal{H}_k, \tag{9}$$

where $s_i \in \mathcal{S}$, $a_i \in \mathcal{A}$ and a kernel K is an embedding kernel of $\mathcal{H}_k$. Since $\mathcal{A} \in \mathbb{R}^m$ where m is usually greater than one, the general choice of K is $K(s,s') = \kappa(s,s')\mathbf{\Gamma}$, where $\kappa(s,s')$ is a scalar kernel and $\mathbf{\Gamma}$ is a matrix that describes the correlation between states. For simplicity, $\mathbf{\Gamma} = \mathbf{I}$, where $\mathbf{I}$ is identity matrix.

3.3 Policy Gradient Algorithms in Reproducing Kernel Hilbert Space

In RKHS, the policy is improved by updating the functional h at each iteration by

$$h_{k+1} \leftarrow h_k + \alpha \nabla_h J(\pi(h_k)),$$

where α is a learning rate and k is the current iteration. An advantage of RKHS is that the policy gradient $\nabla_h J(\pi)$ can be easily and efficiently estimated. Particularly, the gradient of $\log \pi(a_t|s_t)$ (the Fréchet derivative) is of the form,

$$\nabla_h \log \pi(a|s) = K(s,\cdot)\mathbf{\Sigma}^{-1}(a - h(s)) \in H_k. \tag{10}$$

By substituting Eq. 10 into Eq. 6, the policy gradient w.r.t h using the LR method has the form:

$$\nabla_h \widehat{J}(\pi) = \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T} R(\xi^n)K(s,\cdot)\mathbf{\Sigma}^{-1}(a - h(s)) \in H_k. \tag{11}$$

Similarly, substituting Eq. 10 into Eq. 7, the policy gradient using the PGT method in RKHS has the form:

$$\nabla_h \widehat{J}(\pi) = \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T} Q^{\pi_h}(\xi_t^n)K(s,\cdot)\mathbf{\Sigma}^{-1}(a - h(s)) \in H_k. \tag{12}$$

Using the LR method, the gradient at each data point can be easily estimated by directly using the total discounted reward $R(\xi^n)$. However, with the PGT method, we must approximate Q^{π_h} by $\widehat{Q}^{\pi_h}$. To achieve this in RKHS, the authors in (Lever and Stafford, 2015) proposed to use a compatible function approximation approach. Particularly, they compute the Q^{π_h} action-value function by gathering extra trajectories and approximating it as,

$$\mathrm{Q}^{\pi_h}(\xi_i) \approx \sum_{i=t}^{T} \gamma^t r(\xi_i).$$

Next, they proved that if $\widehat{Q}^{\pi_h}$ is in a RKHS with a compatible kernel K_h that is constructed from K as

$$K_h\left[(s,a),(s',a')\right] = \left(K\left(s,s'\right)\mathbf{\Sigma}^{-1}\left(a - h\left(s\right)\right)\right)^T \mathbf{\Sigma}^{-1}\left(a - h\left(s\right)\right), \tag{13}$$

then $\widehat{Q}^{\pi_h}$ is a compatible function approximator for Q^{π_h}. $\widehat{Q}^{\pi_h}$ can be obtained by using any kernel regression algorithm with the compatible kernel K_h. For an experimental purpose, the authors in (Lever and Stafford, 2015) chose Kernel Matching Pursuit (KMP)(Vincent and Bengio, 2002) to quickly regress $\widehat{Q}^{\pi_h}$ in RKHS. A disadvantage of their method is that the computation from a greater number of trajectories may incur in more costs. In addition, this method does not use the importance sampling technique, therefore it requires much more interactions with the environment to collect data. For the purpose of deriving off-policy algorithms, we apply a kernel method to approximate Q^{π_h}, called Kernel Least-Squares Temporal Difference Q-Learning (KLSTD-Q) (Xu et al., 2007). KLSTD-Q can be easily used with importance sampling to estimate the compatible function approximator $\widehat{Q}^{\pi_h}$ from trajectories sampled from other policies.

After each policy iteration, the number of functional points h increases and becomes equal to the number of data points in the trajectories. This allows the policy to model complex tasks. However, the new policy will increase the computation time. Therefore, sparsification is necessarily performed to obtain a sparse policy. The authors in (Lever and Stafford, 2015) were also used KMP to achieve a sparse policy.

3.4 KLSTD-Q for Approximate Q-function in Reproducing Kernel Hilbert Space

Least-Squares Temporal Difference Q-Learning (LSTD-Q) (Boyan, 1999) is a well-known method to regress action-value functions in RL problems. Kernel-based Least-Squares Temporal Difference Q-Learning (KLSTD-Q) is an extended method to estimate value functions in RKHS (Xu et al., 2007). This method has been proven to be more efficient than LSTD-Q in approximating the Q-value function. Originally, the method chooses a kernel function K in a general Hilbert space. In this paper, we make a special application of KLSTD-Q in the case of a kernel K_h in RKHS.

In TD(0)-Q learning, the temporal differences have the following basis form:

$$\delta_t = r(\xi_t) + \gamma Q_t(\xi_{t+1}) - Q_t(\xi_t),$$

where ξ_{t+1} is the successive state-action pair of ξ_t, $r(\xi_t)$ is an immediate reward achieved at the state-action pair ξ_t, and $Q(\xi_t)$ is the action-value function. The term $Q(\xi_t)$ can be replaced by a compatible approximation function: $\widehat{Q}^{\pi_h}(\xi_t) = \phi^T(\xi_t)\mathbf{w}$, where $\mathbf{w}$ is a parameter vector and $\phi(\xi_t)$ is a feature map on the data. In RKHS, to obtain a compatible kernel from $\phi(\xi_t)$, the authors in (Lever and Stafford, 2015) introduced a form of the feature map as:

$$\phi : (s, a) \to K(s, \cdot)\mathbf{\Sigma}^{-1}(a - h(s)).$$

With this feature map, the constructed scalar-valued kernel $K_h : (\mathcal{S}, \mathcal{A}) \times (\mathcal{S}, \mathcal{A}) \to \mathbb{R}$ is truly a compatible kernel in RKHS as defined in Eq. 13. The TD error can be rewritten as:

$$\delta_t = r(\xi_t) + \gamma\phi^T(\xi_{t+1})\mathbf{w} - \phi^T(\xi_t)\mathbf{w}.$$

In (Tsitsiklis and Van Roy, 1997), the TD algorithm was proven to converge with probability 1 under certain assumptions and the limit of convergence was also derived, which satisfies the following equation:

$$\mathbb{E}\left[\phi(\xi_t)\left(\phi^T(\xi_t) - \gamma\phi^T(\xi_{t+1})\right)\right]\mathbf{w} = E\left[\phi(\xi_t)r(\xi_t)\right], \tag{14}$$

where $\mathbb{E}$ stands for expectation. By solving this equation, we can obtain $\mathbf{w}$ in the form of regression,

$$\mathbf{w} = \left(\sum_{t=1}^{T} \phi(\xi_t) \left(\phi^T(\xi_t) - \gamma \phi^T(\xi_{t+1}) \right) \right)^{-1} \left(\sum_{t=1}^{T} \phi(\xi_t) r(\xi_t) \right),$$

and this is the form of LSTD-Q.

Now, to apply the kernel method, we represent the parameter vector $\mathbf{w}$ in the form of a sum of state-action pair feature vectors:

$$\mathbf{w} = \sum_{t=1}^{T} \phi(\xi_t) \alpha_t, \tag{15}$$

where α_t is the coefficient corresponding to each feature map $\phi(\xi_t)$. By substituting $\mathbf{w}$ from Eq. 15 into Eq. 14 and replacing the expectation by the empirical average from a set of samples ξ_i, we obtain:

$$\sum_{i=1}^{T} \left[\phi(\xi_i) \left(\phi^T(\xi_i) - \gamma \phi^T(\xi_{i+1}) \right) \right] \sum_{j=1}^{T} \phi(\xi_j) \alpha_j = \sum_{i=1}^{T} \phi(\xi_i) r(\xi_i) \tag{16}$$

Interpreting Eq. 16 in each step, we have:

$$\phi(\xi_i) \left(\phi^T(\xi_i) - \gamma \phi^T(\xi_{i+1}) \right) \sum_{j=1}^{T} \phi(\xi_j) \alpha_j = \phi(\xi_i) r(\xi_i). \tag{17}$$

Let $\mathbf{\Phi}_T = (\phi^T(\xi_1), \phi^T(\xi_2), ..., \phi^T(\xi_T))^T$ and

$$\overrightarrow{\mathbf{K}}(\xi_i) = (K_h(\xi_1, \xi_i), K_h(\xi_2, \xi_i), ..., K_h(\xi_T, \xi_i))^T. \tag{18}$$

By multiplying $\mathbf{\Phi}_T$ to both sides of Eq. 17, we get:

$$\overrightarrow{\mathbf{K}}(\xi_i) \left[\overrightarrow{\mathbf{K}}^T(\xi_i) \alpha - \gamma \overrightarrow{\mathbf{K}}^T(\xi_{i+1}) \alpha \right] = \overrightarrow{\mathbf{K}}(\xi_i) r_i.$$

By solving this equation, we get the solution for $\boldsymbol{\alpha}$ as follows:

$$\boldsymbol{\alpha} = \mathbf{A}_T^{-1} \mathbf{b}_T, \tag{19}$$

where

$$\mathbf{A}_T = \sum_{i=1}^{T} \overrightarrow{\mathbf{K}}(\xi_i) \left[\overrightarrow{\mathbf{K}}^T(\xi_i) - \gamma \overrightarrow{\mathbf{K}}^T(\xi_{i+1}) \right] \tag{20}$$

and

$$\mathbf{b}_T = \sum_{i=1}^{T} \overrightarrow{\mathbf{K}}(\xi_i) r. \tag{21}$$

Furthermore, in order to decrease the computation costs of kernel methods, we use the Approximate Linear Dependency method (ALD)(Xu et al., 2007) to sparsify the data samples. After performing the sparsification procedure, the feature vectors produced by the elements in the data dictionary will become representative to the feature vectors of the whole data set, i.e., $\mathbf{\Phi}^D = (\phi^T(\xi_1), \phi^T(\xi_2), ..., \phi^T(\xi_D))^T$ and $\overrightarrow{\mathbf{K}}^D(\xi_i) = (K_h(\xi_1, \xi_i), K_h(\xi_2, \xi_i), ..., K_h(\xi_D, \xi_i))^T$, where $\mathcal{D}_T = \{\xi_1, \xi_2, ..., \xi_D\}$ is the data dictionary and $D \ll T$.

4 Policy Gradient Algorithms via Importance Sampling in Reproducing Kernel Hilbert Space

We now describe the use of importance sampling and regularization ideas for RKHS policy search methods which are our two main contributions. We show that this integration leads to a new objective function in RKHS, in which its gradient can still be computed analytically and estimated efficiently.

Reusing notations, we denote $\pi_{h,\mathbf{\Sigma}}$ as the current non-parametric policy and $\widetilde{\pi}_{h,\mathbf{\Sigma}}$ as the previous policies. At the k^{th} policy iteration, we collect N trajectories of horizon T by executing the current policy $\pi_{h,\mathbf{\Sigma}}$. An on-policy algorithm would use only the trajectories generated by the current policy to estimate the functional gradient. The gradient w.r.t the current policy in this case has the following estimate form:

$$\frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T}\nabla_{h,\mathbf{\Sigma}}\log\pi(a_t^n|a_t^n)A(\xi_t^n)\xrightarrow{N\to\infty}\nabla_{h,\mathbf{\Sigma}}J(\pi),$$

where $A(\xi_t^n)$ can be $R(\xi^n)$ or $Q^{\pi_h}(\xi_t^n)$ depending on the theory applied (the LR method or PGT method respectively) and $\nabla_{h,\mathbf{\Sigma}}\log\pi(a_t^n|s_t^n)$ has the form as Eq. 10. In contrast to the on-policy situation, it is the off-policy situation where estimating the gradient w.r.t the current policy uses not only the collected trajectories of the current iteration, but also the collected trajectories of previous iterations from previous policies $\widetilde{\pi}_{h,\mathbf{\Sigma}}$. This characteristic of off-policy estimation allows the reduction of the number of collected trajectories at each iteration (N is small), and it can be applied to tasks, in which the cost of gathering data is very expensive. However, a naive use of off-policy evaluation may lead to wrong estimation of the current gradient. The left hand side of Eq. 22 has the form of a naive use of off-policy which uses all generated trajectories (denoted as $\widetilde{N}$, $\widetilde{N}=N\times K$ where N is the number of trajectories collected at each iteration and K is the current iteration) to estimate the policy gradient without any mechanisms to tailor mismatch between trajectories. As a result, this equation leads to a wrong estimation of policy gradient (right hand side)

$$\frac{1}{\widetilde{N}}\sum_{n=1}^{\widetilde{N}}\sum_{t=1}^{T}\nabla_{h,\mathbf{\Sigma}}\log\pi(a_t^n|s_t^n)A(\xi_t^n)\overset{\widetilde{N}\to\infty}{\not\longrightarrow}\nabla_{h,\mathbf{\Sigma}}J(\pi). \tag{22}$$

One of the techniques that is able to solve this problem is importance sampling. The importance sampling technique measures the difference of a sampling distribution with a target distribution, and it can be applied to reduce the mismatch between the data samples generated by previous policies and the data samples generated by the current policy.

4.1 Importance Sampling Technique in Reproducing Kernel Hilbert Space

Suppose that $X=\{x_1,x_2,...x_N\}$ is a set of i.i.d samples drawn from a strictly positive probability density function $q(x)$. In importance sampling techniques, the expectation of a function $f(x)$ over another probability density function $p(x)$ can be consistently estimated by the importance-weighted average:

$$\frac{1}{N}\sum_{n=1}^{N} f(x_n)\frac{p(x_n)}{q(x_n)} \xrightarrow{N\to\infty} \underset{\pi(x)}{E}\left[f(x)\right].$$

The ratio of two densities $\frac{p(x_n)}{q(x_n)}$ is called the importance weight of a data sample x. Intuitively, the importance weight is used to adjust the distribution of previous data samples to be closed to the target distribution. Applying this importance sampling technique, we obtain a consistent estimation of the policy gradient as:

$$\frac{1}{\widetilde{N}}\sum_{n=1}^{\widetilde{N}} w(\xi^n)\sum_{t=1}^{T}\nabla_{h,\mathbf{\Sigma}}\log\pi(a_t^n|s_t^n)A(\xi_t^n) \xrightarrow{\widetilde{N}\to\infty} \nabla_{h,\mathbf{\Sigma}}J(\pi) \tag{23}$$

with an importance weight $w(\xi^n) = \frac{p(\xi^n|\pi_{h,\mathbf{\Sigma}})}{q(\xi^n|\widetilde{\pi}_{h,\mathbf{\Sigma}})}$. We can easily recognize that the estimation with importance sampling technique (Eq. 23) represents the naive version in Eq. 22 combined with an importance weight for each trajectory. Furthermore, by assuming that all data samples are i.i.d, the importance weight can be estimated without knowledge of $p(\xi^n|\pi_{h,\mathbf{\Sigma}})$ or $p(\xi^n|\widetilde{\pi}_{h,\mathbf{\Sigma}})$. The estimated importance weight at the t-step of a trajectory has the form:

$$\widehat{w}(\xi_t) = \frac{\prod_{t'=1}^{t}\pi(a_{t'}|s_{t'})}{\prod_{t'=1}^{t}\widetilde{\pi}(a_{t'}|s_{t'})}. \tag{24}$$

Inspired from (Wawrzyński, 2009), we also have that the non-parametric policy setting satisfies the necessary conditions for using the truncating technique, such as the type of probability distributions, and the sufficient conditions of bounded variance and asymptotic unbiasedness. Besides this technique, the normalizing technique provides the form of weighted importance sampling, which has a lower variance. This variance is much lower than the unnormalized estimator even though it is a biased estimator. Combining these two techniques, the truncated and normalized form of the gradient estimation can be written as:

$$\frac{1}{Z_{h,\mathbf{\Sigma}}}\sum_{n=1}^{\widetilde{N}}\frac{p(\xi^n|\pi)}{q(\xi^n|\widetilde{\pi})}\sum_{t=1}^{T}\nabla_{h,\mathbf{\Sigma}}\log\pi(a_t^n|s_t^n)\left[A(\xi_t^n) - \widetilde{J}_{h,\mathbf{\Sigma}}\right] \xrightarrow{\widetilde{N}\to\Omega} \nabla_{h,\mathbf{\Sigma}}J(\pi), \tag{25}$$

where $\widetilde{J}_{h,\mathbf{\Sigma}} = \sum_{n=1}^{N}\frac{p(\xi^n|\pi)}{q(\xi^n|\widetilde{\pi})}A(\xi_t^n)$, $Z_{h,\mathbf{\Sigma}} = \sum_{n=1}^{\widetilde{N}}\frac{p(\xi^n|\pi)}{q(\xi^n|\widetilde{\pi})}$ and Ω is the number of most recent trajectories. Fig. 1 compares performance of a non-importance weight algorithm, a naive importance sampling algorithm (unnormalized) and a normalized importance weight algorithm on the Toy domain (see the Experiments section for the problem description). With the same configuration as in (Vien et al., 2016), the result shows that the use of normalized importance weights gives a much better convergence rate than the other methods.

One problem with the estimator in Eq. 25 is that it only considers the ratio of densities of each sample, but does not assume any constraint over the magnitude

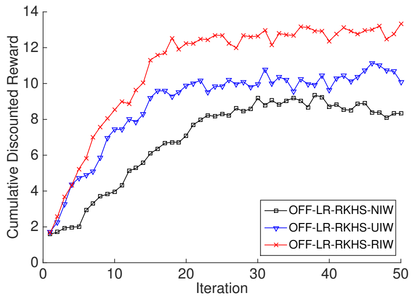


Fig. 1 Comparison of the performance of different versions of the likelihood ratio based on the off-policy algorithms: a non-importance weight algorithm (NIW), a unnormalized importance weight (UIW) and a regularized normalized importance weight (RIW)

of these densities. This implies that when the weights are peaky at a few samples, the estimator returns a premature convergent policy. Hence, this estimator has a very poor ability to be generalized. This issue is recognized in (Levine and Koltun, 2013a) and is solved by introducing a regularization term in the form of the expected total discounted reward. For more details, the expected cumulative discounted reward with importance weight and regularization term has the form

$$J(\boldsymbol{\theta}) = \frac{1}{Z_{\boldsymbol{\theta}}} \sum_{n=1}^{N} \frac{p(\xi^n|\pi)}{q(\xi^n|\widetilde{\pi})} R(\xi^n) + w_r \log(Z_{\boldsymbol{\theta}}). \tag{26}$$

where the regularization term is the logarithm of $Z_{\boldsymbol{\theta}} = Z_{h,\boldsymbol{\Sigma}}$. Parameter w_r is used to control the importance of the regularization term. The gradient of Eq. 26 can be written in terms of the gradients of $Z_{\boldsymbol{\theta}}$ and $\pi_{\boldsymbol{\theta}}$:

$$\nabla J(\boldsymbol{\theta}) = \frac{1}{Z_{\boldsymbol{\theta}}} \sum_{n=1}^{N} \frac{\nabla_{\boldsymbol{\theta}} p(\xi^n|\pi)}{q(\xi^n|\widetilde{\pi})} R(\xi^n) - \frac{\nabla Z_{\boldsymbol{\theta}}}{Z_{\boldsymbol{\theta}}^2} \sum_{n=1}^{N} \frac{p(\xi^n|\pi)}{q(\xi^n|\widetilde{\pi})} R(\xi^n) + w_r \frac{\nabla Z_{\boldsymbol{\theta}}}{Z_{\boldsymbol{\theta}}}.$$

From the definition of $Z_{\boldsymbol{\theta}}$, we have that,

$$\frac{\nabla Z_{\boldsymbol{\theta}}}{Z_{\boldsymbol{\theta}}} = \frac{1}{Z_{\boldsymbol{\theta}}} \sum_{n=1}^{N} \frac{\nabla p(\xi^n|\pi)}{q(\xi^n|\widetilde{\pi})}.$$

Letting $\widetilde{J}(\boldsymbol{\theta}) = \frac{1}{Z_{\boldsymbol{\theta}}} \sum\limits_{n=1}^{N} \frac{p(\xi^n|\pi)}{q(\xi^n|\widetilde{\pi})} R(\xi^n)$, we can rewrite the gradient as

$$\begin{aligned} \nabla J(\boldsymbol{\theta}) &= \frac{1}{Z_{\boldsymbol{\theta}}} \sum_{n=1}^{N} \frac{\nabla_{\boldsymbol{\theta}} p(\xi^n|\pi)}{q(\xi^n|\widetilde{\pi})} \left[R(\xi^n) - \widetilde{J}(\boldsymbol{\theta}) + w_r\right] \\ &= \frac{1}{Z_{\boldsymbol{\theta}}} \sum_{n=1}^{N} \frac{p(\xi^n|\pi)}{q(\xi^n|\widetilde{\pi})} \sum_{t=1}^{T} \nabla_{\boldsymbol{\theta}} \log \pi(a_t|s_t) \left[R(\xi^n) - \widetilde{J}(\boldsymbol{\theta}) + w_r\right], \end{aligned} \tag{27}$$

where we use the log trick $\nabla_{\boldsymbol{\theta}} p(\xi|\pi) = p(\xi|\pi)\nabla_{\boldsymbol{\theta}} \log p(\xi|\pi)$, and the derivative of $\log p(\xi|\pi)$ can be computed without knowledge of $p(\xi|\pi)$ as $\nabla_{\boldsymbol{\theta}} \log p(\xi|\pi) =$

$\sum_{t=1}^{T} \nabla_{\boldsymbol{\theta}} \log \pi(a_t|s_t)$. Eq. 27 is equivalent to the likelihood ratio approach. The general version of the likelihood ratio approach is the policy gradient theorem, which uses the fact that past rewards do not depend on future actions:

$$\nabla J(\boldsymbol{\theta}) = \frac{1}{Z_{\boldsymbol{\theta}}} \sum_{n=1}^{N} \frac{p(\xi^n|\pi)}{q(\xi^n|\widetilde{\pi})} \sum_{t=1}^{T} \nabla_{\boldsymbol{\theta}} \log \pi(a_t|s_t) \left[Q(\xi_t^n) - \widetilde{J}(\boldsymbol{\theta}) + w_r \right]. \tag{28}$$

In RKHS, $\nabla J(\boldsymbol{\theta})$ truly belongs to $\mathcal{H}_K$ and the term $\nabla_{\boldsymbol{\theta}} \log \pi(a_t|s_t)$ is computed as

$$\nabla_{\boldsymbol{\theta}} \log \pi(a_t|s_t) = K(s, \cdot)\boldsymbol{\Sigma}^{-1}(a_t - h(s_t)) \in \mathcal{H}_k.$$

Fig. 2 shows effects of different values of w_r on the Swing-up Pendulum domain (the experimental section for the domain description). Selecting an appropriate parameter w_r is not easy. Thus, for the next section, we develop algorithms that allow for adaptive changes in the value of w_r so that it can automatically converge to an optimal value.

4.2 Proposed Algorithms

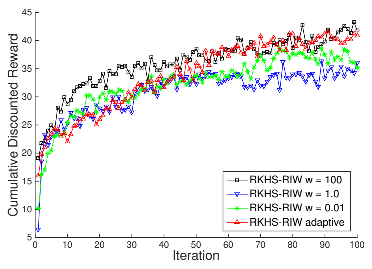
Importance sampling and regularization are combined in order to introduce a new class of RKHS policy gradient algorithms. We classify these into two kinds of off-policy algorithms, those based on the Likelihood Ratio Approach (OFF-LR-RKHS-RIW) and those based on Policy Gradient Theorem Approach (OFF-PGT-RKHS-RIW). The pseudo-code of the two algorithms is summarized in Algorithm 1.

Two algorithms use a non-parametric policy $\pi_{h,\boldsymbol{\Sigma}}$ in form of Eq. 8. The number of centers in the policy is at most C. Initially, there is not a center in the policy. For each iteration, N trajectories are collected using the current policy and are stored in a database $\varXi$. The database only contains at most $\varOmega$ trajectories. Thus, when the number of collected episodes exceeds the capacity of the database, the oldest episodes will be discarded. The algorithms loop through the database to approximate policy gradient $\nabla_h J$, which uses Eq. 27 or Eq. 28 depending on LR based algorithm or PGT based algorithm, respectively. These equations require to compute importance weights $\widehat{w}(\xi^n)$ (line 10) and advantage value $A(\xi^n)$ (line 13) in advance. In LR based algorithm, the advantage value is equivalent to the accumulated discounted reward of an episode and can be easily computed. However, in PGT based algorithm, we need to estimate the advantage value using KLSTD-Q method as described in Section 3.4 and summarized in Algorithm 2. In that method, first, we loop through all state-action pairs to construct a Compatible Kernel Matrix $\mathbf{K}$ (line 4 to line 11), which is then used to approximate state-action value in a regression procedure (line 12 to line 20).

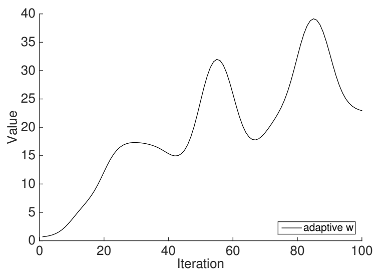
The policy in Algorithm 1 is updated in line 15 and is sparsified in line 16. The sparsified policy only has at most C centers, which are selected based on KMC method. The sparsified policy is used for the next iteration if its performance is better than the performance of the previous one. The parameter w_r of the regularizer is adaptively changed based on the performance of the updated policy.

Algorithm 1: Off-Policy Algorithms in RKHS

```
   Input: MDP $M = \{\mathcal{S}, \mathcal{A}, r, \mathcal{P}\gamma\}$
          Kernel $K : \mathcal{S} \times \mathcal{S} \to \mathcal{L}(\mathcal{A})$;
          Maximum number of centers in the policy $h$: $C$
          Initial co-variance $\mathbf{\Sigma}$
          Regularizer parameter $w_r$
1  Initialize: $h_1 = 0$
2  for $k = 1, 2, \dots K$ do
3      Collect trajectories: collect $N$ trajectories $\{\xi^1, \xi^2, \dots \xi^N\}$ of horizontal $T$ using
       the current policy $\pi_{h,\mathbf{\Sigma}}$
4      Store trajectories $\{\xi^1, \xi^2, \dots \xi^N\}$ to the database $\Xi$
5      Truncate the database $\Xi$ (only keep the $\Omega$ most recent trajectories in the
       database)
6      # Compute importance weights
7      for $n = 1, 2, \dots \Omega$ do
8          Importance weight: $\widehat{w}(\xi^n) = 1$
9          for $t = 1, 2, \dots T$ do
10             Update importance weight: $\widehat{w}(\xi_t) = \widehat{w}(\xi^n) \times \frac{\pi_{h,\mathbf{\Sigma}}(a_t|s_t)}{\widetilde{\pi}_{h,\mathbf{\Sigma}}(a_t|s_t)}$
11         end
12     end
13     Estimate $A(\xi^n)$: $A(\xi^n) = \begin{bmatrix} R(\xi^n) & \text{LR based approach} \\ Q^{\pi_h}(\xi^n) & \text{PGT based approach (Algorithm 2)} \end{bmatrix}$
14     Approximate: $\nabla_h J$ using $\begin{bmatrix} \text{Eq. 27} & \text{LR based approach} \\ \text{Eq. 28} & \text{PGT based approach} \end{bmatrix}$
15     Compute potential policy: $h^* = h_i + \alpha \nabla_h J$ for learning rate $\alpha$
16     Sparsify potential policy: Sparsify $h^*$ using KMP method
17     if $h^*$ is better than $h_i$ then
18         Update new policy: $h_{i+1} = h^*$
19         Adaptive changing $w_r$: Decrease $w_r$
20     else
21         Update new policy: $h_{i+1} = h_i$
22         Adaptive changing $w_r$: Increase $w_r$
23     end
24 end
```



(a) Comparison of different values of w_r



(b) Adaptive w_r, $H = 100$

Fig. 2 Comparison of different values of w_r

Algorithm 2: KLSTD-Q in RKHS

```
Input: Database $\Xi$ of maximum $\Omega$ trajectories
       Kernel $K : \mathcal{S} \times \mathcal{S} \to \mathcal{L}(\mathcal{A})$
1  Sparsify trajectories (optional) using the ALD method (refer to (Xu et al., 2007)).
   the output of this method is the dictionary $\mathcal{D} = \{\xi_1, \xi_2, ..., \xi_D\}$
2
3  # Compute Compatible Kernel Matrix K with size $(\Omega \times T, D)$
4  for $n = 1, 2, \dots \Omega$ do
5      for $t = 1, 2, \dots T$ do
6          for $i = 1, 2, \dots D$ do
7              Compute Compatible Kernel:
8              $\mathbf{K}(\xi_t^n, \xi_i) = K_h(\xi_t^n, \xi_i)$ constructed from $K$ as in Eq. 13
9          end
10     end
11 end
12 Let $\mathbf{A} = []$ and $\mathbf{b} = []$
13 for $n = 1, 2, \dots \Omega$ do
14     for $t = 1, 2, \dots T$ do
15         Compute $\mathbf{A} = \mathbf{A} + \overrightarrow{\mathbf{K}}(\xi_t^n)\left[\overrightarrow{\mathbf{K}}^T(\xi_t^n) - \gamma\overrightarrow{\mathbf{K}}^T(\xi_{t+1}^n)\right]$
16         Compute $\mathbf{b} = \mathbf{b} + \overrightarrow{\mathbf{K}}(\xi_t^n) r$
17     end
18 end
19 Compute coefficients $\boldsymbol{\alpha} = \mathbf{A}^{-1}\mathbf{b}$
20 Compute Q-function approximation $\widehat{Q}^{\pi_h} = \sum_{i=1}^{\Omega \times T} \overrightarrow{\mathbf{K}}(\xi_i^n)\alpha_i$
Output: $\widehat{Q}^{\pi_h}$
```

4.3 Complexity analysis of the derived algorithms

This section provides an analysis on the complexity of the derived algorithms. Given the use of the kernel method (Hofmann et al., 2008), the running time of the derived algorithms would depend on the size of the training data (E.g. state-action pairs). Assuming that the derived algorithms always maintain a trajectory's storage Ξ in the memory, which keeps the Ω most recent trajectories along the learning process ($\Omega \gg N$). Ω significantly influences the complexity of the algorithms because for each trajectory in the storage and for each state-action pair in a trajectory, we must compute importance weights $\widehat{w}$. Let K be the number of iterations along with the learning process, T be the average number of state-action pairs in a trajectory, and C be the maximum number of centers in the policy. We analyze the time complexity of the Likelihood Ratio Approach and the Policy Gradient Theorem Approach as in Theorem 1.

Theorem 1 *The time complexity of the algorithms based on the Likelihood Ratio Approach is $\mathcal{O}(K \times \Omega \times T \times C)$ and for those based on Policy Gradient Theorem Approach is $\mathcal{O}(K \times \Omega \times T \times C \times D)$.*

We have that the time complexity to obtain all of the importance weights is $\mathcal{O}(K \times \Omega \times T \times C)$. In the algorithms based on the Policy Gradient Theorem Approach, an additional overhead in complexity is used to estimate the Q value function. Particularly, the complexity to estimate the Q value function is $\mathcal{O}(K \times \Omega \times T \times C \times D^3)$ (due to the inversion of A), where D is the number of state-action pairs in the dictionary after sparsify using ALD method. The time complexity of other parts in algorithms, such as truncating trajectory, approximate policy gradient, sparsify the policy, is less than those mentioned above. Totally, the time complexity of the algorithms based on the Likelihood Ratio Approach is $\mathcal{O}(K \times \Omega \times T \times C)$ and for those based on Policy Gradient Theorem Approach is $\mathcal{O}(K \times \Omega \times T \times C \times D^3)$. Hence, the most computation is from the evaluation of the Q value function via KLSTD-Q.

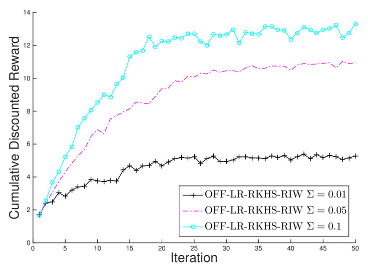
5 Experiments

In this section, we evaluate the usefulness of our proposed off-policy algorithms: Likelihood Ratio based Policy Search in RKHS with Regularized Importance Weight (OFF-LR-RKHS-RIW) and Policy Gradient Theorem based Policy Search in RKHS with Regularized Importance Weight (OFF-PGT-RKHS-RIW). These algorithms are compared with versions that use unnormalized importance weight (OFF-LR-RKHS-UIW, OFF-PGT-RKHS-UIW) and versions that use non-importance weight (OFF-LR-RKHS-NIW, OFF-PGT-RKHS-NIW). We compare these algorithms on benchmarks with increasing complexity: the Toy problem, Swing-up pendulum problem, Cannon problem, and quadrotor navigation problem. All methods use a discount factor of $\gamma = 0.99$ and are compared based on the cumulative discounted reward. For fairness in comparison, we use the same number of centers in RKHS for each domain problem, the same number of collected trajectories for each policy iteration, the same line search function (over a grid of 40 choices) in order to select an appropriate learning rate, and the same kernel (Gaussian) function. The parameter Ω for controlling the truncation technique is five.

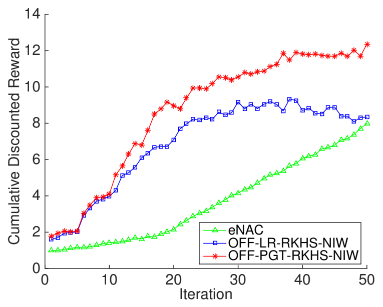
5.1 The Toy Problem

This domain problem is introduced in (Lever and Stafford, 2015). It is a Markov chain with state space $\mathcal{S} \in [-4, 4]$, action space $\mathcal{A} \in [-1, 1]$ and reward function $r(s, a) = e^{-|s-3|}$. The transition function is $s_{t+1} = s_t + a_t + \varepsilon$, where ε is a small Gaussian noise (standard deviation is 0.01). Each trajectory starts at an initial state $s_0 = 0$. All methods use a non-parametric policy of at most 20 centers and a length of trajectory $T = 20$. For each policy iteration, we only collect one trajectory by following the current policy. In order to evaluate the effect of co-variance $\mathbf{\Sigma}$ on the performance of the derived algorithms, we compare the performance on different values of $\mathbf{\Sigma}$ (Fig. 3(a)). The value of co-variance $\mathbf{\Sigma}$ determines how much exploration is in the action space. Thus, a small value of co-variance (0.05) will slowly explore the action space and lead to a poor performance. For each of the experiments in other domains, we use a diagonal co-variance matrix that returns the highest performance.

Another benchmark compares the standard non-parametric methods without using importance sampling (OFF-LR-RKHS-NIW and OFF-PGT-RKHS-NIW) and the episodic Natural Actor Critic (eNAC) algorithm (Peters and Schaal, 2008a) (a state-of-the-art parametric method). The eNAC uses a Gaussian policy with a fixed number of centers (20 centers). Because eNAC is an on-policy algorithm, we need to collect more trajectories at each iteration (five trajectories). The performance of the benchmark is shown in Fig. 3(b). With the same number of iterations, eNAC needs more trajectories than the standard algorithms (250 versus 50), and it cannot achieve the same performance as the standard algorithms. Intuitively, because pre-defined features in the parametric policy of eNAC are only learned in a rigid space, it is hard to find a suitable set of features without tweaking in advance. More evaluations found in (Lever and Stafford, 2015) also prove that algorithms with non-parametric policy in RKHS can outperform or equal algorithms with parametric policy.



(a) Performance of toy domain on different values of co-variance $\mathbf{\Sigma}$ using OFF-LR-RKHS-RIW algorithm



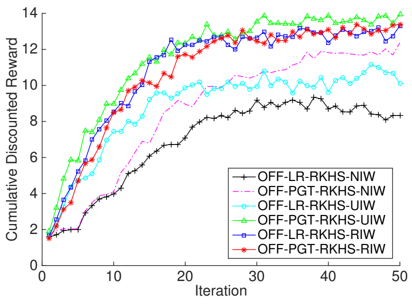
(b) Performance of toy domain when comparing standard non-parametric methods with eNAC algorithm

Fig. 3 Evaluate toy domain on different benchmarks

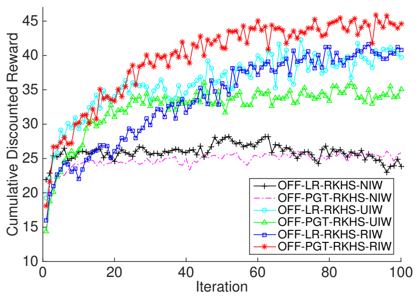
Using a fixed value of $\mathbf{\Sigma}$ (0.1), Fig. 4(a) shows a comparison of performance between the algorithms, which is averaged over 50 trials. From the figure, we can see the benefit of using importance weight algorithms (*-UIW and *-RIW) compared to non-importance weight algorithms (*-NIW). Due to the use of an incorrect gradient estimation, algorithms without importance sampling show slow convergence compared to important weight algorithms. This evaluation does not show the difference in performance between importance sampling based algorithms because the regularization term only makes a difference for domains with a long trajectory. All algorithms using importance sampling reach an optimal reward of 14, which is equivalent to the result reported in (Lever and Stafford, 2015).

5.2 The Swing-up Pendulum Problem

This benchmark appears in a lot of studies on RL and we follow the setup from (Lever and Stafford, 2015). At every time instance ($\Delta t = 0.05$), the controller applied a torque $\mathcal{A} \in [-3, 3]$ to a pendulum, which has the state is $s = (\theta, \omega)$. Here, θ is the angular position in the interval $\Theta = [-\pi, \pi)$ ($\theta = 0$ means the pendulum is straight up) and ω is the angular velocity in the interval $\Omega = [-4\pi, 4\pi]$. The reward function is defined by $r(s, a) = e^{-0.5\theta^2}$ and the transition function is $\theta_{t+1} = \theta_t + \omega_t \Delta t + \varepsilon_1$ and $\omega_{t+1} = \omega_t + a\Delta t + 2\varepsilon_2$, where ε_1 and ε_2 are small



(a) Toy Problem, $H = 20$



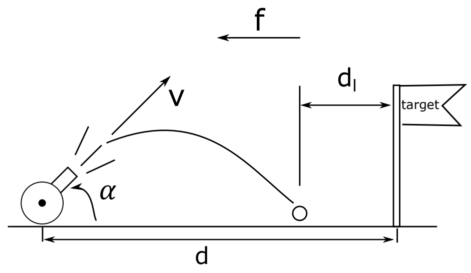
(b) Swing-up Pendulum Problem, $H = 100$

Fig. 4 The Toy Problem and Swing-up Pendulum Problem

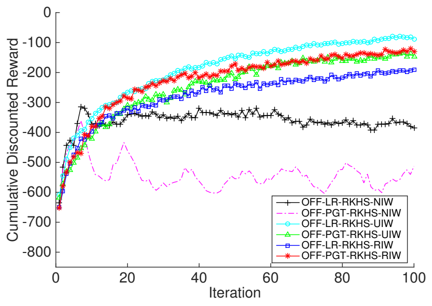
Gaussian noise with standard deviation 0.02. Each transition function is applied twice before being applied to the next action selected by the policy controller. The initial state is $s_0 = (-\pi, 0)$. For each iteration, we collect one trajectory of horizon of length $T = 100$. The maximum number of centers in each policy controller is 50. The performance for all algorithms is averaged over 25 experiments and is shown in Fig. 4(b). The results show that the importance weight algorithms outperform the non-importance weight algorithms. In contrast to the toy domain, this domain has a longer horizon which might be enough to see the importance of the regularization term. Here the algorithms with regularization term (*-RIW) present better convergence compared to the ones without regularization term (*-UIW). The optimal reward is around 45 which implies that the pole remains swung up and balanced until the trajectory ends.

5.3 The Cannon Problem

This domain problem is introduced in (Lawrence et al., 2002) and is used as a benchmark in (Neumann, 2011)(Kober et al., 2012). The task of this domain (Fig. 5(a)) is to hit a target located at distance d with a cannon ball. The cannon ball is launched at an angle α with velocity v, where α is in the interval $[0, \pi/2]\,rad$ and the velocity is in the interval $[0, 10]\,m/s$. The cannon ball was modeled as a 1 kg point mass. The cannon is affected by a horizontal wind force f, which can be in the interval $[0, 1]\,N$. The distance from the cannon to the target d is in the interval of $[0, 10]$ m. The cannon problem can be modeled as an RL problem with a two-dimensional state space $[d, f]$ and a two-dimensional action space $[\alpha, v]$. The reward function is defined as $r(s, a) = -20d_l^2$, where d_l is the distance between the target position and the position where the cannon ball hits ground after launching. For each iteration, we collect 50 trajectories of horizontal length $T = 1$. The maximum number of centers of the policy controller is 100. Fig. 5(b) shows the performances of the algorithms averaged over 100 experiments. After 100 iterations, the importance weight algorithms can reach within 2 m of the target at arbitrary initial states. In this domain, algorithms without importance sampling still show poor performance. Algorithms without regularization (OFF-LR-RKHS-UIW) show the best performance while algorithms with regularization do not show efficiency compared to other domains. It seems that the regularization term is not necessary for domains with short trajectory (The length of a trajectory in cannon task is only one).



(a) Cannon problem illustration

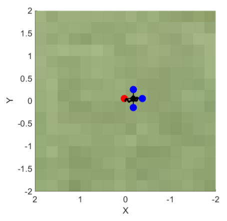


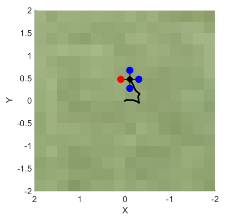
(b) Cannon performance, $H = 1$

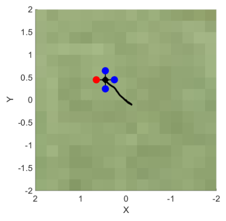
Fig. 5 The Cannon problem

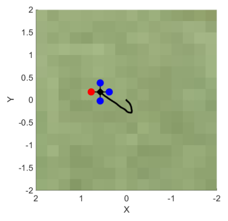
5.4 The Quadrotor navigation problem

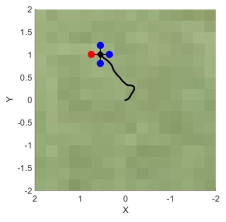
This problem is introduced in (Lever and Stafford, 2015) and uses a MATLAB simulation (De Nardi, 2013) that models the dynamics of Pelican^{TM} quadrotor. This is a high-dimensional nonlinear task with a state of 13 dimensions $s = (x, y, z, \theta, \phi, \psi, \dot{x}, \dot{y}, \dot{z}, \dot{\theta}, \dot{\phi}, \dot{\psi}, F)$, where $x, y, z, \dot{x}, \dot{y}, \dot{z}$ are the position and velocity of the quadrotor in NED coordinates; $\theta, \phi, \psi, \dot{\theta}, \dot{\phi}, \dot{\psi}$ are the roll, pitch, yaw angle and corresponding velocities; and the thrust F applied to the rotors. The action space is a three-dimensional space $\mathcal{A} \in \mathbb{R}^3$ that represents the desired velocities of the quadrotor. An internal PID controller will translate these actions to low level control and send them to rotors at a rate of 50 Hz. The platform is initialized at position $(0, 0, -10)$ and the target of the task is to move to a goal position of $(1, 1, -10)$. The reward is defined as $r(s, a) = e^{-||goal_{xyz} - s_{xyz}||}$. The policy controller uses 100 centers and at each policy iteration, we collect a trajectory of 25 horizontal steps ($T = 25$). Due to the presence of a high dimensional continuous state and action space, the quadrotor navigation problem is complex enough to challenge our proposed algorithms. Even though the complexity of this task is high, our off-policy methods only require to gather one trajectory for each policy iteration while on-policy methods (Bagnell and Schneider, 2003; Lever and Stafford, 2015; Vien et al., 2016) request more trajectories to estimate a policy gradient and the Q-value function. Our proposed methods can find a near optimal policy, which provides an optimal return of around 19. With this policy, the quadrotor can gradually navigate to the goal position and hover around the goal until the end of the trajectory (Fig. 6). The performance of our algorithms after 25 iterations (averaged over 25 experiments) is shown in Fig. 7. The algorithm with regularization (OFF-LR-RKHS-RIW) has shown the best performance.

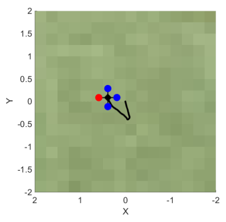

(a) Iteration 1

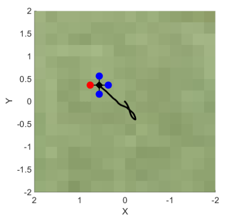

(b) Iteration 3


(c) Iteration 5


(d) Iteration 7


(e) Iteration 8


(f) Iteration 9


(g) Iteration 10

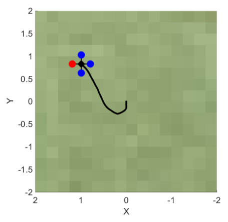

(h) Iteration 11

Fig. 6 Trajectories of the quadrotor during the learning process. The black line indicates trajectory of the quadrotor

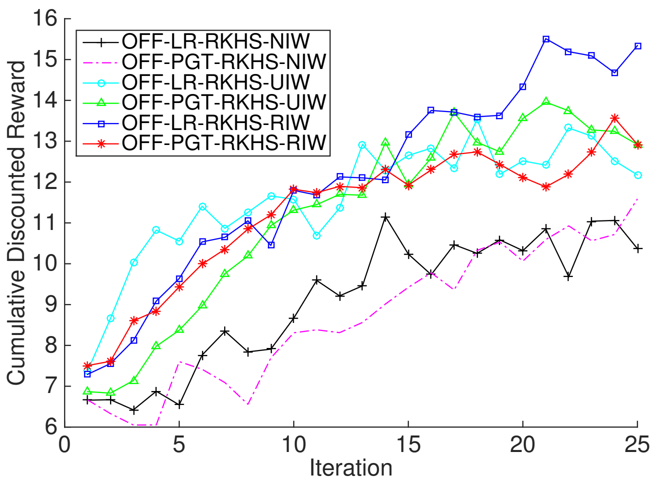


Fig. 7 The quadrotor navigation problem

6 Conclusion

We introduced off-policy algorithms whose policies are modeled in RKHS. These algorithms reduce the cost of gathering trajectories and can deal with high-dimensional continuous tasks; additionally, a priori settings for the policy do not represent a problem for these algorithms. We used a regularized importance sampling technique to develop off-policy algorithms and employed a truncation technique to obtain a trade-off between the variance and bias in estimating the policy gradient. For future work, we consider the use of other techniques to efficiently reduce the variance without increasing the bias. In addition, adding a regularization term for feature selection such an idea in (Kolter and Ng, 2009) is a good way to improve the generalization of our algorithm and will be left as future work. Another promising direction would be to study other off-policy algorithms in RKHS, for instance, other methods of policy search such as Expectation Maximization based approaches or Information-theoretic based approaches.

Acknowledgements This research was supported by the MSIP(Ministry of Science, ICT and Future Planning), Korea, under the Global IT Talent support program (IITP-2016-R01341610330001002) supervised by the IITP(Institute for Information and Communication Technology Promotion); the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science, and Technology (2014R1A1A2057735); and the Kyung Hee University in 2016 (KHU-20160601).

7 Compliance with Ethical Standards

Funding: This study was funded by the MSIP(Ministry of Science, ICT and Future Planning), Korea, under the Global IT Talent support program (IITP-2016-R01341610330001002) supervised by the IITP(Institute for Information and Communication Technology Promotion); the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science, and Technology (2014R1A1A2057735); and the Kyung Hee University in 2016 (KHU-20160601).

Conflict of Interest: The authors declare that they have no conflict of interest.

Ethical approval: This article does not contain any studies with human participants or animals performed by any of the authors.

Informed consent: Informed consent was obtained from all individual participants included in the study.

References

Bagnell, J. A. and Schneider, J. (2003). Policy search in kernel hilbert space. Technical report, Robotics Institute, Carnegie Mellon University.

Baxter, J., Bartlett, P. L., and Weaver, L. (2001). Experiments with infinite-horizon, policy-gradient estimation. *Journal of Artificial Intelligence Research*, 15:351–381.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Boyan, J. A. (1999). Least-squares temporal difference learning. In *Proceedings of the Sixteenth International Conference on Machine Learning*, pages 49–56, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Daniel, C., Neumann, G., and Peters, J. (2012). Learning concurrent motor skills in versatile solution spaces. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 3591–3597. IEEE.

De Nardi, R. (2013). The qrsim quadrotors simulator. *RN*, 13(08):08.

Deisenroth, M., Neumann, G., and Peters, J. (2013). A survey on policy search for robotics. *Foundations and Trends in Robotics*, 2(1-2):1–142.

Hachiya, H., Akiyama, T., Sugiayma, M., and Peters, J. (2009). Adaptive importance sampling for value function approximation in off-policy reinforcement learning. *Neural Network*, 22(10):1399–1410.

Hofmann, T., Schölkopf, B., and Smola, A. J. (2008). Kernel methods in machine learning. *The annals of statistics*, pages 1171–1220.

Kober, J., Oztop, E., Peters, J., and Walsh, T. (2011). Reinforcement learning to adjust robot movements to new situations. In *Twenty-Second International Joint Conference on Artificial Intelligence (IJCAI 2011)*, pages 2650–2655. AAAI Press.

Kober, J. and Peters, J. (2011). Policy search for motor primitives in robotics. *Machine Learning*, 84(1):171–203.

Kober, J., Wilhelm, A., Oztop, E., and Peters, J. (2012). Reinforcement learning to adjust parametrized motor primitives to new situations. *Autonomous Robots*, 33(4):361–379.

Kolter, J. Z. and Ng, A. Y. (2009). Regularization and feature selection in least-squares temporal difference learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 521–528. ACM.

Lawrence, G., Cowan, N., and Russell, S. (2002). Efficient gradient estimation for motor control learning. In *Proceedings of the Nineteenth conference on Uncertainty in Artificial Intelligence*, pages 354–361, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc., Morgan Kaufmann Publishers Inc.

Lever, G. and Stafford, R. (2015). Modelling policies in mdps in reproducing kernel hilbert space. In *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*, pages 590–598.

Levine, S. and Koltun, V. (2013a). Guided policy search. In *Proceedings of The 30th International Conference on Machine Learning*, pages 1–9.

Levine, S. and Koltun, V. (2013b). Variational policy search via trajectory optimization. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States.*, pages 207–215.

Micchelli, C. A. and Pontil, M. (2005). On learning vector-valued functions. *Neural computation*, 17(1):177–204.

Milanfar, P. (2013). A tour of modern image filtering: New insights and methods, both practical and theoretical. *IEEE Signal Processing Magazine*, 30(1):106–128.

Neumann, G. (2011). Variational inference for policy search in changing situations. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 817–824.

Peters, J., Mülling, K., and Altun, Y. (2010). Relative entropy policy search. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010.*

Peters, J. and Schaal, S. (2008a). Reinforcement learning of motor skills with policy gradients. *Neural networks*, 21(4):682–697.

Peters, J. and Schaal, S. (2008b). Reinforcement learning of motor skills with policy gradients. *Neural Networks*, 21(4):682–697.

Precup, D., Sutton, R. S., and Singh, S. P. (2000). Eligibility traces for off-policy policy evaluation. In *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML '00, pages 759–766, San Francisco, CA, USA. Morgan

Kaufmann Publishers Inc.

Scholkopf, B. and Smola, A. J. (2001). *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond.* MIT Press, Cambridge, MA, USA.

Schulman, J., Levine, S., Abbeel, P., Jordan, M. I., and Moritz, P. (2015). Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pages 1889–1897.

Shelton, C. R. (2001). Policy improvement for pomdps using normalized importance sampling. In *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence*, UAI'01, pages 496–503, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Sutton, R. S., McAllester, D. A., Singh, S. P., Mansour, Y., et al. (1999). Policy gradient methods for reinforcement learning with function approximation. In *NIPS*, volume 99, pages 1057–1063. MIT Press.

Tsitsiklis, J. N. and Van Roy, B. (1997). An analysis of temporal-difference learning with function approximation. *IEEE transactions on automatic control*, 42(5):674–690.

Vien, N. A., Englert, P., and Toussaint, M. (2016). Policy search in reproducing kernel hilbert space. In *Proceedings of The 25th International Joint Conference on Artificial Intelligence.*

Vincent, P. and Bengio, Y. (2002). Kernel matching pursuit. *Machine Learning*, 48(1-3):165–187.

Wawrzyński, P. (2009). Real-time reinforcement learning by sequential actor–critics and experience replay. *Neural Networks*, 22(10):1484–1497.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256.

Xu, X., Hu, D., and Lu, X. (2007). Kernel-based least squares policy iteration for reinforcement learning. *IEEE Transactions on Neural Networks*, 18(4):973–992.

Zhao, T., Hachiya, H., Tangkaratt, V., Morimoto, J., and Sugiyama, M. (2013). Efficient sample reuse in policy gradients with parameter-based exploration. *Neural computation*, 25(6):1512–1547.