Genetic Programming and Evolvable Machines (2019) 20:479–501
https://doi.org/10.1007/s10710-019-09357-1

A covariance matrix adaptation evolution strategy in reproducing kernel Hilbert space

Viet-Hung Dang[1] · Ngo Anh Vien[2] · TaeChoong Chung[3]

Received: 22 August 2018 / Revised: 15 May 2019 / Published online: 19 June 2019

Abstract

The covariance matrix adaptation evolution strategy (CMA-ES) is an efficient derivative-free optimization algorithm. It optimizes a black-box objective function over a well-defined parameter space in which feature functions are often defined manually. Therefore, the performance of those techniques strongly depends on the quality of the chosen features or the underlying parametric function space. Hence, enabling CMA-ES to optimize on a more complex and general function class has long been desired. In this paper, we consider *modeling* the input spaces in black-box optimization *non-parametrically* in reproducing kernel Hilbert spaces (RKHS). This modeling leads to a *functional optimisation* problem whose domain is a RKHS function space that enables optimisation in a very rich function class. We propose CMA-ES-RKHS, a generalized CMA-ES framework that is able to carry out black-box functional optimisation in RKHS. A search distribution on non-parametric function spaces, represented as a Gaussian process, is adapted by updating both its mean function and covariance operator. Adaptive and sparse representation of the mean function and the covariance operator can be retained for efficient computation in the updates and evaluations of CMA-ES-RKHS by resorting to sparsification. We will also show how to apply our new black-box framework to search for an optimum policy in reinforcement learning in which policies are represented as functions in a RKHS. CMA-ES-RKHS is evaluated on two functional optimization problems and two bench-marking reinforcement learning domains.

Keywords Covariance matrix adaptation-evolution strategies (CMA-ES) · Functional optimization · Policy search · Reinforcement learning · Robot learning · Kernel methods · Reproducing kernel Hilbert space

Area Editor: U.-M. O'Reilly.

✉ Ngo Anh Vien
v.ngo@qub.ac.uk

[1] Department of Computer Science, Duy Tan University, Da Nang, Vietnam

[2] School of EEECS, Queen's University Belfast, Belfast, UK

[3] Department of Computer Engineering, Kyung Hee University, Seoul, Korea

1 Introduction

The covariance matrix adaptation evolutionary strategy (CMA-ES) is a derivative-free method [12] which is a practical optimization tool for continuous optimization problems. It is a general optimization framework that possesses many appealing characteristics, e.g. derivative-free, covariant, off-the-shelf, scalable etc. It is especially useful on problems that are non-convex, non-separable, ill-conditioned, multi-modal, and with noisy evaluations. As the name indicates, CMA-ES belongs to Evolution Strategy (ES), hence it also works by operating three major steps: *recombination*, *mutation* and *selection*. In the context of robotics, CMA-ES has been widely used in many tasks: biped locomotion [43], whole-body locomotion optimization [8, 9], swimming by [34], skill learning via reinforcement learning [14, 15, 31], inverse reinforcement learning [5, 28], etc. Applying CMA-ES requires explicitly a finite-dimensional search space on which solution candidates live. In many domains, e.g. robotics, an optimization objective is often defined as a cost function of another parametric solution function. For instance, it might be an overall cost function depending on a robot controller, e.g. applications CMA-ES in robot skill learning [31] and policy search [15], or a loss function in the context of inverse optimal control [5, 28], etc.. A robot controller is usually a parametric function of predefined feature maps, e.g. radial basis function (RBF) features. Though showing a lot of remarkable successes, such parametric approaches as well as their black-box solver, the parametric CMA-ES, could not model a very rich and flexible solution space, e.g. a complex behavior space. The performance of CMA-ES is affected not only by a suboptimal choice of features, but also the amount of used features. Therefore, integrating the ability of adaptive feature selection or non-parametric techniques into CMA-ES has been desired and not yet investigated, though this integration have been seen very often in other machine learning algorithms.

In this work, we propose CMA-ES-RKHS that enables functional optimization over a non-parametric solution space. Specifically, we assume that the solution (domain) space is a reproducing kernel Hilbert space (RKHS). Each solution candidate is a function in RKHS. Modeling the solution space this way, CMA-ES-RKHS is able to not only inherit full characteristics from CMA-ES, but enjoy other appealing properties of kernel methods as well. *Firstly*, CMA-ES-RKHS is able to optimize a functional objective whose domain consists of functions in a RKHS with an embedding kernel. That means the solution space does not need to depend on a set of predefined features that often has limited complexity, therefore our method can have richer representation power. *Secondly*, by modeling the solution space in RKHS, all update steps in CMA-ES-RKHS are handled analytically. We show that updated mean functionals, other intermediate terms, evolution path functionals or conjugate evolution path functionals are functions in the underlying RKHS. Moreover, the updated covariance is also an operator on the underlying RKHS. In other words, there is no additionally introduced computation when compared to the standard CMA-ES method. *Thirdly*, via sparsification techniques, a very complex search space can be represented compactly due to the

reproducing property of the RKHS. However we can still achieve a solution of guaranteed quality which is at least comparable to that of the standard CMA-ES method. Though the use of sparsification might lead to representation with only a finite number of features, it allows adaptive feature selection and offers compact representation that reduce the computation cost.

By employing CMA-ES-RKHS we propose a non-parametric direct policy search technique in which the policy space in RL is an RKHS. As demonstrated by [2, 19, 36, 38, 40], the policy in RKHS is shown to be powerful and flexible in solving complex RL tasks. However their approaches use functional gradient ascent to update the policy functional that might have a problem with step-sizes or local optimality. It means the optimized policy does not fully exploit the flexible power of modeling in RKHS, which might consist of any complex policies including the global optimum one. As a global optimization method, our policy search via CMA-ES-RKHS is expected to search for such globally optimal policies. This paper is an extension from our work published [39].

Recently, there have been similar efforts in proposing new machine learning frameworks for functional optimization, such as functional regression by [16], representing motion trajectories in RKHS by [6, 21], or finding geodesic shortest paths in physical systems by [17]. We demonstrate the proposed CMA-ES-RKHS and direct policy search via CMA-ES-RKHS in four experiments. The first two synthetic domains demonstrate the advantages of functional optimization in RKHS. The last two experiments are two RL domains, inverted pendulum and double-pendulum swing-up. The experiments will evaluate how CMA-ES-RKHS can do adaptive feature selection in a non-parametric way.

Our paper is organized as follows. The Background section will briefly discuss necessary preliminaries that will be used in the rest of the paper. Section 3 will start with a problem statement of functional optimization and then we will present our proposed framework, CMA-ES-RKHS. Section 4 will describe an application of CMA-ES-RKHS to a policy search problem in reinforcement learning. Section 5 will present experiment results on numerical and RL domains. Finally, we will conclude the paper with some remarks and future research directions.

2 Background

We briefly present a background of the Covariance Matrix Adaptation - Evolution Strategy (CMA-ES), its application for direct policy search reinforcement learning, and the recently introduced policy search in RKHS.

2.1 Covariance matrix adaptation-evolution strategy

CMA-ES is a global optimisation method introduced by [12]. It works by forming a parametric distribution over the solution space, e.g. the space of policy parameter in policy search, or the space of parameters of the loss function in inverse optimal control, etc. It iteratively samples a population of solution candidates from a

parametrized search distribution. These candidates are then evaluated by a black-box function. Tuples of candidate-evaluation make up a dataset in order for CMA-ES to update the search distribution, i.e. its mean and its covariance matrix. Specifically, a cost function $f : \mathfrak{R}^n \mapsto \mathfrak{R}$ is parametrized by a parameter space $\theta \in \mathfrak{R}^n$, $f(\theta)$. The target is to find an optimum parameter θ^* such that $f(\theta^*)$ is minimum. It is common that a CMA-ES algorithm maintains a multivariate Gaussian distribution over the solution space as $\theta \sim \mathcal{N}(\theta; \mathbf{m}, \mathbf{C})$, where $\mathbf{m}$ is a n-dim mean vector and $\mathbf{C}$ is a $n \times n$ covariance matrix. At each iteration k, it generates the kth population of λ offsprings from the kth distribution as $\theta_i \sim \mathcal{N}(\theta; \mathbf{m}_k, \mathbf{C}_k)$, $i = 1, \cdots, \lambda$, where $\mathbf{m}_k, \mathbf{C}_k$ denote the mean vector and the covariance matrix at iteration k (after $\mathbf{m}$ and $\mathbf{C}$ got updates k times). Then, the offsprings are sorted ascendingly according to their evaluations $f(\theta_i)$. Only the first $\mu\,(< \lambda)$ best candidates are selected for use in updates of $\mathbf{m}_k$ and $\mathbf{C}_k$. Another parameter is the global step-size $\sigma \in \mathfrak{R}$ that controls the convergence rate of the covariance matrix update. The parameter σ is defined as a global standard deviation. Hence, a full set of parameters in CMA-ES is $\{\mathbf{m}, \mathbf{C}, \sigma\}$.

In Algorithm 1, we give a full summary of the CMA-ES algorithm. Algorithm 1 starts with an initialisation of the parameters in steps 1 and 2. As discussed above, the parameters $\{\mathbf{m}, \mathbf{C}, \sigma\}$ are updated iteratively as shown in steps from 4 to 13. Step 4 is *sampling* from a normal distribution with mean $\mathbf{m}$ and covariance $\mathbf{C}$. The *evaluations* via query to the black-box function $f(\cdot)$ are in step 5. The updated mean is a weighted sum of the best μ candidates as in step 7, in which the weights w_i are set to $1/\mu$ or to a better choice $\log(\mu/2) - \log(i)$. The notation $\mathbf{y}_{i:\lambda}$ means the best candidate out of $\mathbf{y}_i, \ldots, \mathbf{y}_\lambda$. The covariance matrix update in step 13 consists of three parts: (1) old information, (2) rank-1 update which computes the change of the mean over time as encoded in the *evolution path* $\mathbf{p}_c$, and (3) rank-μ update which takes into account the *good* variations in the last population. Step 10 devotes to the step-size control update that constrains the expected changes of the distribution. Thus, this step is based on the *conjugate evolution path* $\mathbf{p}_\sigma$. It aims to accelerate convergence to an optimum, and meanwhile prevents premature convergence. The other parameters: μ_w is the variance effective selection mass, c_1, c_c, c_σ are learning rates, and d_σ is a damping factor for σ. The setting of these parameters is well studied and discussed in-depth in [10]. The termination can be set in various ways depending on the contracting covariance, or fitness functions (e.g. if the fitness functions of the population do not change for some iterations).

Algorithm 1 The CMA-ES algorithm

1: Initialize $\mathbf{m} \in \Re^n$, $\sigma \in \Re^+$, λ, μ
2: Initialize $\mathbf{C} = \mathbf{I}, \mathbf{p}_c = \mathbf{0}, \mathbf{p}_\sigma = \mathbf{0}$
3: **while** not terminate **do**
4: Sampling: $\theta_i = \mathbf{m} + \sigma \mathbf{y}_i$, $\mathbf{y}_i \sim \mathcal{N}(\mathbf{0}, \mathbf{C}), i = 1, \ldots, \lambda$
5: Evaluating: $f(\theta_i), i = 1, \ldots, \lambda$
6: *// mean update*
7: $\mathbf{m} \leftarrow \mathbf{m} + \sigma \bar{\mathbf{y}}$, where $\bar{\mathbf{y}} = \sum_1^\mu w_i \mathbf{y}_{i:\lambda}$
8: *// step-size control update*
9: $\mathbf{p}_\sigma \leftarrow (1 - c_\sigma)\mathbf{p}_\sigma + \sqrt{c_\sigma(2 - c_\sigma)\mu_w}\mathbf{C}^{-\frac{1}{2}}\bar{\mathbf{y}}$
10: $\sigma \leftarrow \sigma \exp\left(\frac{c_\sigma}{d_\sigma}\left(\frac{\|\mathbf{p}_\sigma\|}{\mathbb{E}\|\mathcal{N}(\mathbf{0},\mathbf{I})\|} - 1\right)\right)$
11: *// covariance matrix update*
12: $\mathbf{p}_c \leftarrow (1 - c_c)\mathbf{p}_c + \sqrt{c_c(2 - c_c)\mu_w}\bar{\mathbf{y}}$
13: $\mathbf{C} \leftarrow (1 - c_1 - c_\mu)\mathbf{C} + c_1 \mathbf{p}_c \mathbf{p}_c^\top + c_\mu \sum_1^\mu w_i \mathbf{y}_{i:\lambda} \mathbf{y}_{i:\lambda}^\top$
14: **end while**

The updates of CMA-ES can alternatively be derived using the information-geometric concept of a natural gradient as shown in [11], which shares the same insight with the *natural evolution strategies* (NES) [44].

There are less efficient but practically fast techniques that can also adapt the covariance matrix: *estimation of distribution algorithms* (EDA) and *the cross-entropy method* (CEM). The major difference from CMA-ES is the choice of the reference mean value. EDA and CEM estimate the variance using the current population information, $\mathbf{y}_{1:\lambda}$, instead of exploiting old information as encoded in previous information of $\mathbf{C}$ and $\mathbf{p}_c$. Specifically, for the Gaussian search distribution, one can modify steps 9 and 13 in Algorithm 1 to receive new updates at iteration k for EDA and CEM [26, 27] as follows

$$\mathbf{m}^{(k)} = \frac{1}{\mu}\sum_{i=1}^{\mu}\theta_i,$$

$$\mathbf{C}_{\text{EDA}}^{(k)} = \frac{1}{\mu}\sum_{i=1}^{\mu}\left(\theta_i - \mathbf{m}^{(k)}\right)\left(\theta_i - \mathbf{m}^{(k)}\right)^\top,$$

$$\mathbf{C}_{\text{CEM}}^{(k)} = \frac{\mu}{\mu - 1}\mathbf{C}_{\text{EDA}}^{(k)}.$$

The difference between EDA and CEM is: EDA updates C to the empirical covariance matrix, meanwhile CEM updates C to the *unbiased* empirical covariance matrix

2.2 Direct policy search in reinforcement learning

In this section, we describe the background of Markov decision process, reinforcement learning and policy search techniques.

2.2.1 Markov decision process and reinforcement learning

A Markov Decision Process (MDP) is a mathematical framework often used to formulate sequential decision making problems that is understood as an interaction process between an agent and an external environment. A MDP is defined as a 5-tuple $\{\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma\}$, where $\mathcal{S}$ is a state space of the environment. $\mathcal{A}$ is an action space that can be taken by the agent to interact with the environment. $\mathcal{T}$ is a transition function defined as $\mathcal{T}(s, a, s') = p(s'|s, a)$ that tells a probability of next state s' if at state s, action a is taken. $\mathcal{R}$ is a reward function, i.e. $\mathcal{R}(s, a)$, that returns a scalar value if the environment at state s and action a is taken. $\gamma \in (0, 1]$ is a discount factor. The agent's task is to find an optimal deterministic policy $\pi : \mathcal{S} \mapsto \mathcal{A}$, which is a mapping from states to actions, that maximizes an expected cumulative discounted return, denoted as $J(\pi)$ a function of π which is defined as

$$J(\pi) = \mathbb{E}_{\mathcal{T}}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right], \tag{1}$$

where the expectation is with respect to the stochasticity of the transitions $\mathcal{T}$. The term inside expectation is the infinite-horizon sum of discounted rewards. Alternatively, the policy π can be stochastic and written as $\pi : \mathcal{S} \times \mathcal{A} \mapsto [0, 1]$ that defines the probability of selecting actions at a state. Finding optimal policies MDP can be handled by the use of dynamic programming, e.g. value iteration or policy iteration algorithms Puterman [24].

Reinforcement learning is a family of learning algorithms in MDP in which $J(\pi)$ is optimised w.r.t π when the agent does not know the dynamics of the environment, i.e. unknown $\mathcal{T}$ and $\mathcal{R}$. In unknown environment, planning techniques like value iteration or policy iteration can not be directly used. There are many different types of RL algorithms, but can often be categorized into two fashion: value-based and policy-based methods [32]. Value-based methods consist of Q-learning, SARSA, etc. that optimizes the policy by estimating intermediate terms, called value funcions or Q-value functions. Policy-based methods (*direct policy search*) optimize $J(\pi)$ on the parametric policy space. The policy can be optimized using policy search techniques such as policy gradient [45], natural actor-critic [23], Bayesian policy gradient Ghavamzadeh et al. [7], Vien et al. [41], or black-box search such as CEM [20] and CMA-ES [15], etc. The first two techniques are local methods that can only look for locally optimal policies (more discussions over their differences can be found in [13]).

In the next sections, we first describe an application of CMA-ES for policy search as introduced in [15], then give a brief introduction of one policy gradient technique that models policies in reproducing kernel Hilbert space [19].

2.2.2 Direct policy search via CMA-ES

As direct policy search would directly search on the space of policies using stochastic optimization, one natural application of CMA-ES for direct policy search is to construct a search distribution on the policy space. Assuming that the policy space is parameterized by a function space $\mathcal{H}$, in which each function $h \in \mathcal{H}$ is defined as $h : \mathcal{S} \mapsto \mathfrak{R}$. Specifically, each policy is written as a deterministic distribution as

$$\mathbf{a} = h(\mathbf{s};\theta), \tag{2}$$

where $\mathbf{a} \in \mathcal{A}$ is an action, $\mathbf{s} \in \mathcal{S}$ is a state. For parametric policy approaches, the function space $\mathcal{H}$ is parametrized by a parameter space $\theta \in \mathfrak{R}^n$. For example, h might be a neural network (where n is the number of trainable parameters in the network), or be a linear function of predefined features (where n is the number of features) as

$$h(\mathbf{s}) = \sum_{j=1}^{n} \theta^j \phi_j(\mathbf{s}), \tag{3}$$

where θ^j denotes the jth coordinate of θ, and $\phi_j(s)$ is the jth feature. Therefore, the objective in Eq. 1 as a function of policy can be re-written as a function of parameters θ. The authors in [15] propose to use the CMA-ES algorithm in Algorithm 1 to optimize a cumulative reward objective function of θ,

$$J(\theta) = \mathbb{E}_{\pi,\mathcal{T}}\left(\sum_{t=0}^{\infty} \gamma^t r_t\right), \tag{4}$$

where the expectation is with respect to the stochasticity of the stochastic policy $\pi(s, a;\theta)$ and the dynamics $\mathcal{T}$. For each candidate θ_i, its evaluation in Step 5 in Algorithm 1 is $J(\theta_i)$. Each $J(\theta_i)$ is computed using Monte-Carlo simulations. Specifically, a set of N trajectories are sampled by executing the policy $\pi(\theta_i)$, then the evaluation is approximated as $J(\theta_i) \approx \frac{1}{N}\sum_{j=1}^{N} R(\xi_j)$, where $R(\xi_j)$ is a return of trajectory j, ξ_j. One advantage of CMA-ES is the ability to adapt the covariance matrix to control exploration, therefore it is considered to be more robust to noise and that is typically the case in robotics. The application of CMA-ES for direct policy search has shown many remarkable results in robotics Deisenrot et al. [4].

2.2.3 Policy gradient in reproducing kernel Hilbert space

An alternative direct policy search is policy gradient which uses gradient ascent to find a local optimal policy π. In this section, we discuss how to use functional gradient ascent that would optimise the objective $J(h)$ directly on a function space of h, instead of a parameterised function space through θ. We start by describing

how to represent a function space in a non-parametric way through a reproducing kernel Hilbert space.

A reproducing kernel Hilbert space is a Hilbert space of functions in which the evaluation functional is a continuous linear functional. Denote $\mathcal{S}$ a set (e.g. a state space in MDP) and $\mathcal{H}$ a Hilbert space of functions on $\mathcal{S}$. An evaluation functional is defined as a linear functional that has an evaluation at each point $\mathbf{s}$ as

$$L_{\mathbf{s}} : f \mapsto f(\mathbf{s}), \quad \forall f \in \mathcal{H}.$$

Essentially, the above definition means the linear functional $L_{\mathbf{s}}$ evaluates f at a $\mathbf{s}$ to receive $L_{\mathbf{s}}(f) = f(\mathbf{s})$.

According to the Riesz representation theorem [18], for all $\mathbf{s}' \in \mathcal{S}$ there exists a unique evaluation functional $K(\mathbf{s}, \cdot) \in \mathcal{H}$ satisfying the reproducing property:

$$K(\mathbf{s}, \mathbf{s}') = \langle K(\mathbf{s}, \cdot), K(\cdot, \mathbf{s}') \rangle,$$

where $\langle \cdot, \cdot \rangle$ denotes an inner product. The above result also means that we are evaluating the function $K(\cdot, \mathbf{s}')$ using the evaluation functional $K(\mathbf{s}, \cdot)$, and vice versa. We call the Hilbert space $\mathcal{H}$ with a reproducing kernel K, a reproducing kernel Hilbert space $\mathcal{H}_K$ [29].

Recently, there have been efforts in integrating kernel methods with policy search in reinforcement learning. The work by [2, 19] suggest to represent policies in reproducing kernel Hilbert spaces in order to enjoy rich representation and flexible complexity, in which each function $h(\cdot)$ in the definition of policies in Eq. 2 is a vector-valued function in RKHS $\mathcal{H}_K$, and defined as

$$h(\cdot) = \sum_i K(\mathbf{s}_i, \cdot)\alpha_i, \quad \text{where } \alpha_i \in \mathcal{A}, \tag{5}$$

and K is a vector-valued kernel, $K : \mathcal{S} \times \mathcal{S} \mapsto \mathcal{L}(\mathcal{A})$ [22], in which $\mathcal{L}(\mathcal{A})$ is the space of linear operators on $\mathcal{A}$ (an action space). In policy gradient, we often resort to stochastic policies in order to make gradients exist and promote exploration in RL, therefore Gaussian policies are commonly chosen and written as

$$\pi(\mathbf{s}, \mathbf{a}; h) = p(\mathbf{a}|\mathbf{s}; h) \sim \mathcal{N}(h(\mathbf{s}); \Sigma), \tag{6}$$

where Σ is a covariance matrix that dictates the level of exploration. In all experiments, we use a diagonal matrix $\Sigma = \sigma^2\mathbf{I}$, where σ starts with a large value (when exploration dominates) and decreases gradually (until exploitation dominates) Lever and Stafford [19]. A more sophisticated strategy can be used to adapt Σ, e.g. functional gradient Vien et al. [40].

Using this functional policy parametrization, we can analytically derive the functional gradient $\nabla_h J(\pi_h)$ of the objective $J(\pi_h)$ w.r.t h by using the Fréchet derivative [18], and compute its approximate value given a set of sampled trajectories $\{\xi_i\}_{i=1}^N$ of length H at most,

$$\begin{aligned}\nabla_h J(\pi_h) &= \nabla_h \mathbb{E}\left\{\sum_t \gamma^t r_t\right\} \\ &= \nabla_h \int R(\xi) P(\xi;h) d\xi \\ &= \int \left[R(\xi)\nabla_h \log P(\xi;h)\right] P(\xi;h) d\xi \\ &\approx \frac{1}{N}\sum_{i=1}^{N} \nabla_h \log P(\xi_i;h) R(\xi_i) \\ &= \frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{H} Q(\mathbf{s}_t, \mathbf{a}_t) K(\mathbf{s}_t, \cdot)\Sigma^{-1}(\mathbf{a}_t - h(\mathbf{s}_t)),\end{aligned} \tag{7}$$

where $R(\xi) = \sum_t \gamma^t r_t$ is the accumulated reward of the trajectory ξ, and $P(\xi_i)$ is the probability distribution of trajectories. In the third step, we use the log trick $\nabla_h P(\xi;h) = [\nabla_h \log P(\xi;h)]P(\xi;h)$. The Monte-Carlo estimation step means that we sample N trajectories $\{\xi_i\}_{i=1}^N$ from the distribution $P(\xi;h)$. $Q(\mathbf{s}_t, \mathbf{a}_t)$ is a sample of the Q-value function of $(\mathbf{s}_t, \mathbf{a}_t)$ (this equality results from the Policy Gradient theorem [33]). Thus, the derivation in Eq. 7 results in $\nabla_h J(\pi_h)$ as a function in $\mathcal{H}_K$ as defined in Eq. 5. In addition, the functional gradient update at iteration k can be written as

$$h_{k+1} \leftarrow h_k + \alpha \nabla_h J(\pi_{h_k}), \tag{8}$$

where α is a step-size. As the representation of each policy h_{k+1} becomes more complex after each iteration [19] (the number of linear evaluation functionals $K(s_i, \cdot)$ involved in the representation of h_{k+1}), a sparsification technique is often used to achieve a sparse and adaptive policy. A compatible kernel for Q-functions $Q^\pi(\mathbf{s}, \mathbf{a})$ can simply be derived based on kernel K, that is K_h as

$$K_h\big((\mathbf{s}, \mathbf{a}), (\mathbf{s}', \mathbf{a}')\big) = \Big(K(\mathbf{s}, \mathbf{s}')\Sigma^{-1}(\mathbf{a} - h(\mathbf{s}))\Big)^\top \Sigma^{-1}(\mathbf{a}' - h(\mathbf{s}')). \tag{9}$$

Kernel regression methods can be used to approximate Q easily via kernel matching pursuit [42], kernelized Least Square Temporal Difference Q-learning (k-LSTDQ) [46], etc. This framework is called the compatible RKHS Actor-Critic framework (RKHS-AC) by [19]. Though RKHS-AC has very excellent policy modeling, it would only result in locally optimal policies. A global policy search technique like CMA-ES which may be considered as a powerful off-the-shelf tool but has many drawbacks on its current form. For many applications of CMA-ES, a parametric objective function is often designed, which requires a set of predefined features. Hence, the choice of features plays a very important role. Recently, the authors in [40] propose a natural actor-critic in RKHS algorithm by computing the Fisher

information operator, which is analogue to the Fisher information matrix for parametric policies.

3 CMA-ES in reproducing kernel Hilbert space

3.1 Problem statement

We consider a black-box functional optimisation problem [37] that finds the maximum of an unknown functional $f : \mathcal{H} \mapsto \mathfrak{R}$, where $\mathcal{H} = \{h : \mathcal{X} \mapsto \mathcal{Y}\}$ is a separable Hilbert space of vector-valued functions h with a domain $\mathcal{X}$ and output space $\mathcal{Y}$. For each queried function $h \in \mathcal{H}$, an evaluation $y = f(h)$ is returned. The goal is to find an optimal function $h^* = \arg\max_{h \in \mathcal{H}} f(h)$ such that the functional $f(h^*)$ is maximum with least queries, i.e. $f(h^*) \geq f(h), \forall h \in \mathcal{H}$. For example, we might want to find trajectories of minimum cost for a mobile robot to move from one location to a desired destination. A trajectory is represented by a function $h : [0, 1] \mapsto \mathcal{C}$, that maps time $t \in [0, 1]$ to robot configuration $h(t) \in \mathcal{C}$ where $\mathcal{C}$ denotes the configuration space of a robot. A configuration is the positions of all robot points as represented relatively in a fixed coordinate system. The cost functional $f(h)$ maps each trajectory h to a scalar cost in $\mathfrak{R}$. The cost might measure the efficiency of the trajectory (*energy efficiency*) and the proximity to obstacles (*obstacle avoidance*).

3.2 The CMA-ES-RKHS framework

We propose a new general-purpose CMA-ES-RKHS framework, a generalized CMA-ES variant in RKHS, that solves the above problem. We explicitly assume the function space $\mathcal{H}$ as a reproducing kernel Hilbert space (RKHS) associated with a kernel K. Each $h \in \mathcal{H}$ is defined as a mapping from an arbitrary space $\mathcal{X}$ to $\mathcal{Y}$, $h : \mathcal{X} \mapsto \mathcal{Y}$. The function space $\mathcal{H}$ may be a vector-valued RKHS [22], denoted as $\mathcal{H}_K$, with the kernel $K : \mathcal{X} \times \mathcal{X} \mapsto \mathcal{L}(\mathcal{Y})$, where $\mathcal{L}(\mathcal{Y})$ is the space of linear operators on $\mathcal{Y}$. For example, when $\mathcal{X} = \mathfrak{R}^n$ the simplest choice of K might be $K(x, x') = \kappa(x, x')\mathrm{I}_n$, where I_n is an $n \times n$ identity matrix, and κ is a scalar-valued kernel [29]. Each function $h \in \mathcal{H}$ is then represented approximately as a linear span of finite elements $\{x_i, y_i\}$ as

$$h(\cdot) = \sum_i K(x_i, \cdot) y_i. \tag{10}$$

Now we define a search distribution over $\mathcal{H}$. A direct extension of parametric CMA-ES is to use a Gaussian process over the solution function h, $h \sim \mathcal{GP}(m, \sigma^2 C)$, where m is a mean function in $\mathcal{H}_K$, C is a covariance operator on $\mathcal{H}_K$, and σ is a scalar global step-size. We discuss now how to update the functionals $\{m, C\}$ and the parameter σ in our CMA-ES-RKHS framework, which is also summarised in Algorithm 2. Basically, one might think of the difference between CMA-ES versus CMA-ES-RKHS as: (1) the parameterization is a finite parameter space of $\theta \in \mathfrak{R}^n$ versus a potentially infinite space of $h \in$ RKHS, (2) the search distribution is a

multi-variate Normal distribution versus a Gaussian process, (3) each sample from the search distribution is a finite parameter vector θ_i versus a function h_i. We now describe the CMA-ES-RKHS framework in more detail.

Algorithm 2 The CMA-ES-RKHS algorithm

1: Initialize $m \in \mathcal{H}_K$, $\sigma \in \Re^+$, λ, μ
2: Initialize $C = \delta(\cdot,\cdot), p_c \in \mathcal{H}_K, p_\sigma \in \mathcal{H}_K$
3: **while** (not terminate) **do**
4: *Sampling*: $\tilde{g}_i \sim \mathcal{GP}(0, C), i = 1, \ldots, \lambda$
5: Via kernel regression: for each $\tilde{g}_i$, fit a function $g_i \in \mathcal{H}_K$
6: Set samples: $h_i = m + \sigma^{(t)} g_i$
7: *Evaluating*: $f_i = f(h_i), i = 1, \ldots, \lambda$
8: *// mean update*
9: $m \leftarrow m + \sigma\bar{g}$, where $\bar{g} = \sum_1^\mu w_i g_{i:\lambda}$
10: *// step-size control update*
11: $p_\sigma \leftarrow (1 - c_\sigma)p_\sigma + \sqrt{c_\sigma(2 - c_\sigma)\mu_w} C^{-\frac{1}{2}} \bar{g}$
12: $\sigma \leftarrow \sigma \exp\left(\frac{c_\sigma}{d_\sigma}\left(\frac{\|p_\sigma\|}{\mathbb{E}\|\mathcal{GP}(0,\delta(\cdot,\cdot))\|} - 1\right)\right)$
13: *// covariance matrix update*
14: $p_c \leftarrow (1 - c_c)p_c + \sqrt{c_c(2 - c_c)\mu_w}\bar{g}$
15: $C \leftarrow (1 - c_1 - c_\mu)C + c_1 p_c \otimes p_c + c_\mu \sum_1^\mu w_i g_{i:\lambda} \otimes g_{i:\lambda}$
16: Sparsify m, C and derive $C^{-\frac{1}{2}}$
17: **end while**

3.2.1 Mean function update in RKHS

Assuming that at iteration k, we can sample a set of λ functions $\tilde{g}_i \sim \mathbb{GP}(0, C)$ (Step 4), where $\mathbb{GP}(0, C)$ is a Gaussian process with mean function 0 and covariance C. Many techniques for sampling from a Gaussian process are basically described in [25]. It is commonly known that a sample from a Gaussian process is not in $\mathcal{H}_K$ with probability of 1, as discussed in detail by [1]. For any sampling techniques of a Gaussian process, we receive $\tilde{g}_i$ in a form of data tuples $(x^{(i)}, y^{(i)})$. In order to receive a function in $\mathcal{H}_K$, we approximate $\tilde{g}_i$ by a function $g_i \in \mathcal{H}_K$. One technique allows us to do this is kernel ridge regression whose kernel is the covariance operator $C(\cdot,\cdot)$ (Step 5). Hence, in our framework each function $\tilde{g}_i$ is approximated by a function $g_i \in \mathcal{H}_K$. As a result, a new function candidate sampled from the function distribution is $h_i = m + \sigma g_i$. The new mean function is updated as (Step 9),

$$m = m + \sigma \sum_{i=1}^{\mu} w_i g_{i:\lambda} \in \mathcal{H}_K, \tag{11}$$

where the normalized weights w_i satisfy

$$\sum_{i=1}^{\mu} w_i = 1, \quad w_1 \geq w_2 \geq \cdots \geq w_\mu > 0.$$

As a result, after the update the new functional mean is an element in $\mathcal{H}_K$. We denote $\bar{g}$ as

$$\bar{g} = \sum_{i=1}^{\mu} w_i g_{i:\lambda}.$$

There are a number of settings for w that might inherit from that of CMA-ES, for example: $w_i = 1/\mu$, $w_i \propto \mu - i + 1$; or a better choice $w_i = \log(\mu + \frac{1}{2}) - \log(i)$. In our experiment, we implement the latter as it performs consistently better in all domains.

3.2.2 Covariance operator update

The covariance operator update (Steps 14–15) is based on the best selected candidate functions in terms of their evaluations $f(h_i)$. Hence an empirical estimate of the covariance operator C on $\mathcal{H}_K$, called *rank-μ update*, is

$$C = (1 - c_\mu)C + c_\mu \sum_{i=1}^{\mu} w_i g_{i:\lambda} \otimes g_{i:\lambda},$$

where c_μ is a learning rate of rank-μ, and $\otimes$ denotes an outer product. Similar to parametric CMA-ES, we also consider the change of the mean function over time by estimating an evolution path function p_c as (Step 14),

$$p_c = (1 - c_c)p_c + \sqrt{c_c(2 - c_c)\mu_w}\bar{g} \in \mathcal{H}_K, \tag{12}$$

where μ_w is a variance-effectiveness constant, and c_c is the backward time horizon for the evolution path function p_c. This is low-pass filtered of chosen steps $\bar{g}$. As p_c is just a linear combination of functions in $\mathcal{H}_K$, therefore p_c is also an element in RKHS $\mathcal{H}_K$. As a result, a complete update of the covariance operator that combines both rank-1 and rank-μ is computed as (Step 15),

$$C = (1 - c_\mu - c_1)C + c_1 p_c p_c^\top + c_\mu \sum_{i=1}^{\mu} w_i g_{i:\lambda} \otimes g_{i:\lambda}, \tag{13}$$

where and c_1, c_μ are learning rates of rank-1 and rank-μ respectively. This reduces to a rank-1 update if $c_1 = 1, c_\mu = 0$. Similarly, the update becomes a rank-μ update when $c_1 = 0, c_\mu = 1$.

3.2.3 Step-size update

The global step-size σ is adapted through the computation of a conjugate evolution path function p_σ as (Step 11),

$$p_\sigma = (1 - c_\sigma)p_\sigma + \sqrt{c_\sigma(2 - c_\sigma)\mu_w}\mathrm{C}^{-\frac{1}{2}}\bar{g}, \tag{14}$$

where c_σ is a backward time horizon for the conjugate evolution path function p_σ. According to the bounded inverse theorem in functional analysis [3], C as computed in Eq. 13 is a linear operator in the RKHS $\mathcal{H}_K$, hence it has a bounded inverse C^{-1}. Therefore, p_σ is updated in a way that renders it an element in $\mathcal{H}_K$. The volume and the correlation of the selected steps are compared to the expected value of the standard Gaussian process with a Dirac kernel. The fact that the former is larger than the latter makes σ increased, otherwise decreased. The update formula of σ (Step 12) is

$$\sigma = \sigma \exp\left(\frac{c_\sigma}{d_\sigma}\left(\frac{\|p_\sigma\|}{\mathbb{E}\|\mathcal{GP}(0, \delta(\cdot,\cdot))\|} - 1\right)\right), \tag{15}$$

where $\|\cdot\|$ is the L^2-norm, and d_σ is a damping factor for σ. The term $\mathbb{E}\|\mathcal{GP}(0, \delta_x)\|$ is the expectation of all L^2-norms of functions sampled from $\mathcal{GP}(0, \delta_x)$. This term can be computed in advance using Monte-Carlo simulations as

$$\mathbb{E}\|\mathcal{GP}(0, \delta(\cdot,\cdot))\|_{\mathcal{H}_K} \approx \frac{1}{N}\sum_{i=1}^{N}\langle g_i(\cdot), g_i(\cdot)\rangle_{\mathcal{H}_K},$$

where $g_i(\cdot)$ is a function in $\mathcal{H}_K$ approximated (via kernel ridge regression) from a sample $\tilde{g}_i$ drawn from $\mathcal{GP}(0, \delta(\cdot,\cdot))$.

3.2.4 Sparsification and adaptive representation

We now discuss implementation concerns of the CMA-ES-RKHS algorithm. Firstly, the most critical one is the representation issue of mean functions m and covariance operators C. Secondly, it follows with discussions of parameter setting in CMA-ES-RKHS. Thirdly, we discuss how to deal with the update rule in Eq. 14 that involves finding the inverse operator $C^{-\frac{1}{2}}$.

Sparsification (Step 16): The updates of the mean function in Eq. 11 and covariance operator in Eq. 13 make their representation complexity increase linearly on the number of iterations. Though we receive an adaptive, flexible and complex policy, this would result in an expensive evaluation cost, e.g. computing $m(x)$ or $C(h, h')$ when needed, where $x \in \mathcal{X}$ and $m, h, h' \in \mathcal{H}_K$. Sparsification is a technique that is able to keep these kernel-based representation sparse and approximately accurate. Assuming that after each iteration, m and C are re-written in the forms of

$$m = \sum_{i}^{N_1}\beta_i K(x_i, \cdot), \quad C = \sum_{i}^{N_2}\lambda_i h_i \otimes h_i, \tag{16}$$

where N_1, N_2 are the numbers of functions in the representation of m, C respectively, $x_i \in \mathcal{X}, h_i \in \mathcal{H}_K$, and $\beta_i, \lambda_i \in \mathcal{Y}$. Assuming that a sparsification algorithm would sparsify m and C to become

$$m = \sum_{i}^{n_1} \tilde{\beta}_i K(\tilde{x}_i, \cdot), \quad C = \sum_{i}^{n_2} \tilde{\lambda}_i \tilde{h}_i \otimes \tilde{h}_i,$$

where $\tilde{x}_i \in \mathcal{X}, \tilde{h}_i \in \mathcal{H}_K$ and satisfying that $n_1 \ll N_1, n_2 \ll N_2$. In our CMA-ES-RKHS framework, we resort to two different sparsification techniques separately for m and C. We propose to use the kernel matching pursuit algorithm [42] to sparsify m. Its idea is to add kernel functions $K(x_i, \cdot)$ as features sequentially and greedily that maximally reduce the current approximation error. A tolerance constant is used to check the error reduction level before adding a new kernel feature. The use of tolerance also helps achieving more compact representation.

In general, we can use the kernel matching pursuit algorithm [42] to sparsify C. However, we aim to look for a method that will both sparsify C and together compute the inverse square root operator $C^{-\frac{1}{2}}$, because $C^{-\frac{1}{2}}$ is also used in Step 11 in Algorithm 2. Therefore, we propose to use the *kernel PCA* method (kPCA) from [30] for achieving efficiently and fast both a sparse and compact covariance operator and its inverse square root operator. Specifically, we rewrite C in Eq. 16 as

$$C = H\Lambda H^\top = \Phi\Phi^\top,$$

where H is a matrix whose ith column is h_i, a $N_2 \times N_2$ diagonal matrix $\Lambda = \text{diag}(\lambda_i)$, and $\Phi = H\Lambda^{\frac{1}{2}}$. Via kPCA, C can be decomposed through a decomposition of the $N_2 \times N_2$ Gram matrix $G = \Phi^\top\Phi$, in which $G(i,j) = \sqrt{\lambda_i\lambda_j}\langle h_i, h_j\rangle_{\mathcal{H}_K}$. If a singular value decomposition (SVD) of G is $G = UDU^\top$, the decomposition of C via kPCA is $C = VDV^\top$, where $V = \Phi UD^{-\frac{1}{2}}$ are orthonormal eigenfunctions of C, hence $V = H\Lambda^{\frac{1}{2}}UD^{-\frac{1}{2}}$. One could easily show that each eigenfunction of C is a linear span of $\{h_i\}_{i=1}^{N_2}$,

$$v_i = \sum_i w_{ij} h_j(\cdot),$$

where w_{ij} is an element of a matrix $W = \Lambda^{\frac{1}{2}}UD^{-\frac{1}{2}}$. Hence v_i is also an element in $\mathcal{H}_K$.

From the decomposition of C via kPCA, we are now able to sparsify C by choosing only a small set of eigenfunctions of principal eigenvalues. Moreover, from the decomposition of C, the inverse square root operator $C^{-\frac{1}{2}}$ is derived as

$$C^{-\frac{1}{2}} = VD^{-\frac{1}{2}}V^\top, \tag{17}$$

which is also a linear operator in RKHS $\mathcal{H}_K$.

Parameter setting The mean function is represented by n_1 adaptive kernel features, hence it has n_1 pivotal parameters. In CMA-ES, parameter setting is based on the number of free parameters, the dimensionality of the search space. Though it is not precise, we use the same setting of CMA-ES for CMA-ES-RKHS's parameters, i.e. the parameters: $c_1, c_c, c_\mu, c_\sigma, d_\sigma$ based on n_1. We call n_1 the *effective dimensionality* on our CMA-ES-RKHS.

3.3 EDA and CEM in RKHS

For a short notice, the derivations of CMA-ES-RKHS can easily be transferred to derive EDA–RKHS and CEM–RKHS algorithms which are EDA and CEM in RKHS. Specifically, the updates at iteration k for EDA–RKHS and CEM–RKHS change step 9 and 15 in Algorithm 2 as follows

$$\begin{aligned} m^{(k)} &= \frac{1}{\mu}\sum_{i=1}^{\mu} h_i \in \mathcal{H}_K, \\ C^{(k)}_{\text{EDA–RKHS}} &= \frac{1}{\mu}\sum_{i=1}^{\mu}(h_i - m^{(k)}) \otimes (h_i - m^{(k)}), \\ C^{(k)}_{\text{CEM–RKHS}} &= \frac{\mu}{\mu - 1} C^{(k)}_{\text{EDA–RKHS}}. \end{aligned}$$

Though the updates of $C^{(k)}_{\text{EDA–RKHS}}$ and $C^{(k)}_{\text{CEM–RKHS}}$ are simpler than C of CMA-ES-RKHS, they result in similar covariance operators on $\mathcal{H}_K$. Hence the implementation technique of EDA–RKHS and CEM–RKHS is similar to that of CMA-ES-RKHS as discussed above. Therefore, we will not put them in comparisons due to their weaker performance when compared to CMA-ES-RKHS.

4 Direct policy search via CMA-ES-RKHS

In RL literature, there is recent effort to model policies as functions in RKHS [2, 19, 40]. Such RKHS policy gradient approaches suffer from a problem of step-size and local optima. Though one of the extended work, RKHS EM-based policy search (RKHS-PoWER) by [40], would overcome this issue, it can only converge to local optima. We propose a new black-box direct policy search in RKHS that is based on CMA-ES-RKHS (the extension to EDA–RKHS and CEM–RKHS is similar). We use deterministic policies where each policy is a function in RKHS with a kernel K as first introduced in Eq. 5. Different from the standard CMA-ES based direct policy search [15], our non-parametric modeling enables optimisation in a very rich policy space and allows to learn more complex policies that is able to avoid local optima, enjoy adaptive and compact representation and do not depend on pre-defined features.

Adaptive CMA-ES direct policy search There is a naive way that modifies the parametric CMA-ES direct policy search [15] to become adaptive in selecting features. This naively proposed modification is used as a base-line to compare to CMA-ES-RKHS policy search. Assuming that we use a controller that is a linear span of RBF features $\phi_i(\mathbf{s})$ in which $\mathbf{c}_i$ is a center,

$$h(\mathbf{s}) = \sum_{i=1}^{n} w_i \phi_i(\mathbf{s}). \tag{18}$$

where each $\phi_i(\mathbf{s}) = K(\mathbf{s}, \mathbf{c}_i)$ Thus, h becomes a parametric function whose parameters are $\mathbf{w} = \{w_i\}_{i=1}^n$. This renders policies $\pi(a|\mathbf{s};w)$ a standard parametric policy. Therefore standard policy search algorithms like CMA-ES and policy gradient can be applied straightforwardly. Applying the CMA-ES based direct policy search method [15], the parameter space would be $\mathbf{w} = \{w_i\} \in \mathfrak{R}^n$. We make a slight change to assume that the parameter space is $\{w_i, \mathbf{c}_i\}$, called *adaptive CMA-ES direct policy search* (CMA-ES-A). This adaptive algorithm would search for both optimum weights and optimum RBF features. A clear problem of CMA-ES-A is in determining the scaling of these parameters. However it is in principle overcome by the RKHS norm on functions which is similar to the setting in our CMA-ES-RKHS algorithm.

5 Experiments

In this section, we present experiment results to evaluate and compare our proposed framework to other state-of-the-art approaches. We first evaluate the advantages and general optimisation applications of CMA-ES-RKHS on two simple functional optimisation problems: 1-D and 2-D function spaces. We compare the behavior of CMA-ES-RKHS with other three other methods: the standard CMA-ES, the adaptive CMA-ES version (CMA-ES-A), and the functional gradient techniques. The next experiments are two RL tasks: inverted pendulum and double-pendulum. We compare our direct policy search via CMA-ES-RKHS to the standard CMA-ES policy search, the adaptive CMA-ES policy search, a parametric actor-critic, and the actor-critic in RKHS (RKHS-AC) methods. In all experiments, we use Gaussian *Radial Basis Function* (RBF) as an embedding kernel of the RKHS in which the bandwidths are set using *median-trick*. In other words, the bandwidth is set to the median (of all pair-wise distances between sampled data points) divided by the number of features used in each function. These experiments aim to evaluate the proposed CMA-ES-RKHS for: (1) the quality of the returned compact solution function, (2) the flexibility and power of the proposed method in capturing a complex solution function that can not be found easily by existing methods, (3) the applicability in practice, i.e. for direct policy search in reinforcement learning.

5.1 Synthetic domains

We design two unknown 1-dimension (1-D) and 2-dimension (2-D) functions f^*. Each function is a mixture of two (multivariate in the case of 2-D) Gaussians, respectively. All optimizers are tasked to find a function $h : \mathcal{X} \mapsto \mathfrak{R}$, where $h \in \mathcal{H}_K$ that minimizes the objective function as a square distance to the ground-truth. The objective is written as

$$J(h) = \int_{x_0}^{x_T} \left(f^*(x) - h(x)\right)^2 dx, \tag{19}$$

where $x \in \mathfrak{R}^k$, $k = 1, 2$ correspondingly to the 1-D or 2-D domain. We limit the domain $\mathcal{X}$ from x_0 to x_T for 1-D task and in the box $[x_0, x_T] \times [x_0, x_T]$ for 2-D task. This task is a simplified version of many similar problems in machine learning and robotics, e.g. regularized risk functional ($J(h)$ is a least-square cost and $(x, f^*(x))$ are data samples) [29], trajectory optimisation (where $f^*(x)$ is a reference trajectory w.r.t time $x \in [x_0, x_T]$) [35], path planning or trajectory optimisation in RKHS [21], loss minimization inverse optimal control (where $f^*(x)$ is a demonstration trajectory) [5], etc. For example, in the case of path planning a robot must find a shortest path from one location to a destination while avoiding collisions with obstacles. The cost function in Eq. 19 is computing an Euclidean distance between the current configuration $h(x)$ and the destination configuration represented as $f^*(x)$, together with a distance to the closest obstacle. However, most of the above work must rely on discretization and parametric modelling.

Functional gradient: Using functional gradient requires to know J and have access to the ground-truth function f^* (CMA-ES-RKHS only accesses evaluations $J(h)$) from which we are able to use discretization to approximate the objective J in Eq. 19 as

$$J(h) \approx \frac{1}{T} \sum_{k=0}^{T} \big(f^*(x_k) - h(x_k)\big)^2. \tag{20}$$

The functional gradient can be computed analytically as:

$$\nabla_h J(h) = \sum_{k=0}^{T} 2\big(h(x_k) - f^*(x_k)\big) K(t_k, \cdot).$$

Thus, a functional gradient update is $h \leftarrow h + \alpha \nabla_h J(h)$. A sparsification technique [42] can be used to achieve a compact representation of h which renders the functional gradient approach an adaptive method too. That means the representation of h will be adapted to best approximate f^*. Hence, discretization is required to be fine enough (T is large enough, we used $T \gg N$) to guarantee accurate approximation.

CMA-ES: We assume that a parametric representation of h as a linear expansion of N features: $h(x) = \sum_{k=1}^{N} w_k \phi_k(x) = \mathbf{w}^\top \boldsymbol{\phi}(x)$. We use RBF features $\phi_k(x) = \exp(-\|x - \mathbf{c}_t\|^2/\sigma^2)$ in which N centers $\mathbf{c}_t$ are regular intervals in the domain of $\mathbf{s}$. Hence we apply CMA-ES to optimise J in a parameter space $\mathbf{w} \in \mathfrak{R}^N$. CMA-ES-A would optimize over a search space of $\{\mathbf{w}, \{\mathbf{c}_t\}_{t=1}^N\}$.

Results: For all optimizers, we use the same number N of features in CMA-ES and CMA-ES-A, and centers after sparsification in CMA-ES-RKHS and functional gradient methods. We use $N = 10$ for 1-D task, and $N = 100$ for 2-D task. As mentioned in parameter setting section, we use a standard way of CMA-ES to initialize other parameters in CMA-ES-RKHS. We set the same N to the effective dimensionality in CMA-ES-RKHS. The results are averaged over 15 runs and report the averaged squared error w.r.t the number of evaluations, i.e queries to the objective function.

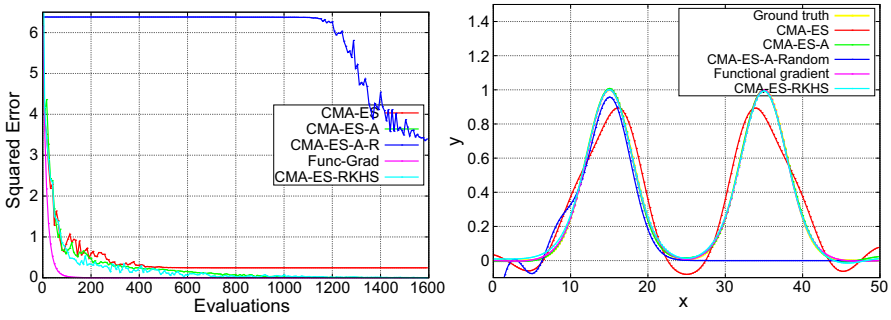

Fig. 1 1D synthetic domain: (left) squared error, (right) solution functions

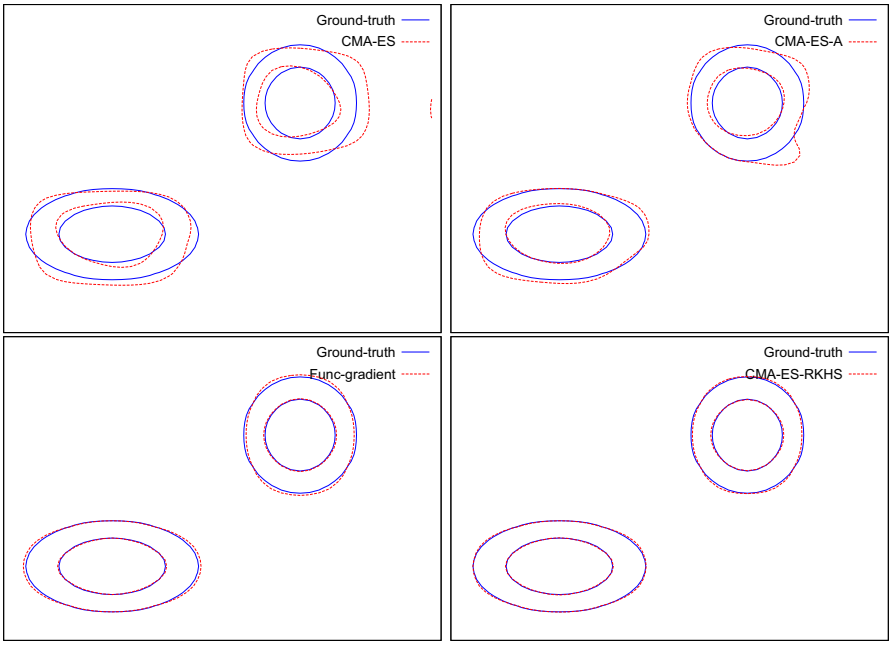

Fig. 2 2D synthetic domain: contours of levels equivalent to the first and second deviations

We report the squared error J and the solution function for the 1-D task in Fig. 1. The results for the 2-D task are reported in Figs. 2 and 3. We create two versions for CMA-ES-A, one with good initialization (initial values of x_t are centers for CMA-ES) and one with random initialization, called CMA-ES-A-R. In Fig. 1, the performance of CMA-ES-A-R is not good in terms of error. As demonstrated on the right picture, it can detect only one mode of the optimal function. Hence we stop reporting results from CMA-ES-A-R in other domains. One remarkable note is that CMA-ES initialization does not consist of two correct modes in its set of centers, hence it

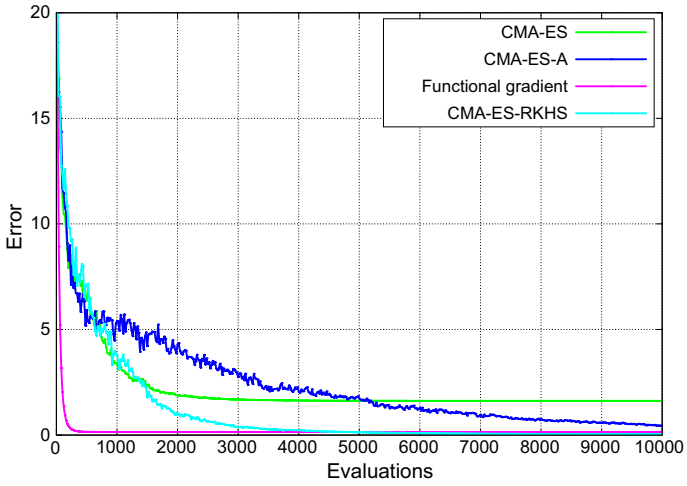

Fig. 3 Comparison results for 2D synthetic domain

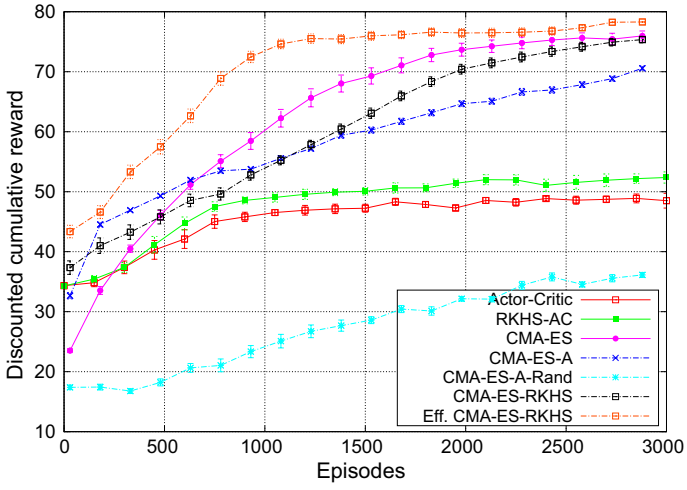

Fig. 4 Comparison results for the inverted pendulum domain

gives poor approximation error. With adaptive ability, CMA-ES-A and CMA-ES-RKHS are able to estimate the true modes correctly.

In the larger domain (2-D), CMA-ES-A performs much worse than our method. This is explained by the way our method approaches from a principled way, i.e kernel methods, for the scaling of parameters. The functional gradient method performs very well which also confirms that it can be very competitive when gradient information is known (in this case the form of $J(h)$ is known). Fig. 2 shows very

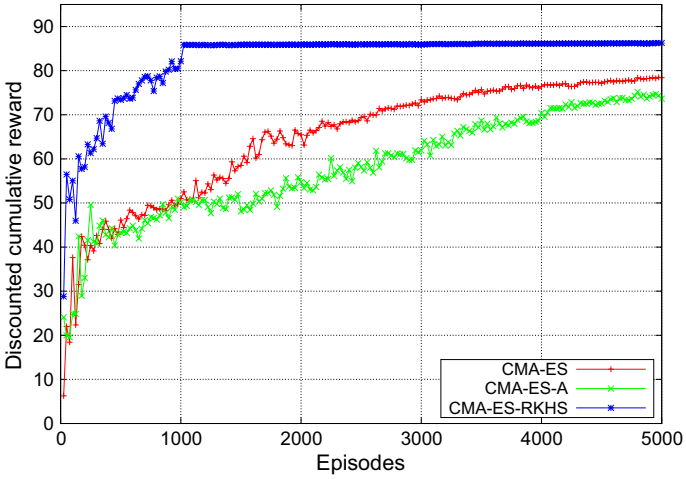

Fig. 5 Comparison results the double-link domain

interesting results where other methods like CMA-ES and CMA-ES-A are still struggling around the optimal regions to find the correct shapes of the Gaussians.

5.2 Inverted pendulum

We use the same setting of the inverted pendulum domain as in [19]. This problem has an 1-dim action space $[-3, 3]$, a state space $s = (\theta, \omega)$, where $\theta \in [-\pi, \pi]$ is angular position and $\omega \in [-4\pi, 4\pi]$ is angular velocity. The system always starts at $s_0 = (-\pi, 0)$ (downward position). The reward function is $r(s, a) = \exp(-0.5\theta^2)$ that requires to bring the pole to the upright position and keep it balanced there. The dynamics of the system is $\theta' = \theta + 0.05\omega + \epsilon$; $\omega' = \omega + 0.05a + 2\epsilon$, ϵ is a small Gaussian noise $\mathcal{N}(0, 0.02^2)$. We use $N = 50$ centers or features for all algorithms. We set $\gamma = 0.99$ and a horizon $H = 400$. Each policy evaluation $J(h)$ of policy h is averaged over 5 episodes.

The results of mean performance and its 95% confidence are computed over 15 runs and reported in Fig. 4. In this task, CMA-ES performs better than CMA-ES-A and CMA-ES-RKHS. CMA-ES-A has a much bigger search space comparing to that of CMA-ES, $3N$ versus N parameters. We conjecture that CMA-ES-RKHS performs worse because we use N as the effective dimensionality to set its parameters. Theoretically, CMA-ES-RKHS optimizes over a potentially infinite dimensional space. We tried to increase its effective dimensionality to $2N$, called Eff. CMA-ES-RKHS. This modification improves the performance significantly. CMA-ES-A-R performs slowly but keeps improving constantly. Local direct policy search algorithms, AC and RKHS-AC, do not perform comparably to the other global direct methods. This experiment shows that CMA-ES-RKHS is able to avoid local optima.

5.3 Double pendulum

This problem consists of two links and two under-actuated joints. The system state is 4-dimensional of joint position and velocities $\mathbf{s} = \{\theta_1, \dot{\theta}_1, \theta_2, \dot{\theta}_2\}$. Actions are motor torques $\mathbf{a} = [u_1, u_2]$, which are limited in $[-5N, 5N]$. The dynamics is simulated using second-order Runge-Kutta. We use a low sampling frequency of $50Hz$ at which torques could be applied. The start state is $\{0, -\pi, 0, 0\}$. The reward function is $r(\mathbf{s}) = \exp(-\|\mathbf{s} - \mathbf{s}^*\|_W)$, where $\mathbf{s}^* = \{0, 0, 0, 0\}$ and $W = \text{diag}(0.25, 0.0025, 0.25, 0.0025)$. Each episode is simulated in $6s$, which is equivalent to a horizon of 300 steps. Each policy evaluation $J(h)$ of policy h is averaged over 5 episodes. We use $N = 256$ features or centers, and $\gamma = 0.99$. The optimal policy returns 88. We only compare between global policy search methods via CMA-ES, CMA-ES-A, and CMA-ES-RKHS.

In this complex task, CMA-ES-RKHS has clearly outperformed other methods as seen in Fig. 5. Due to more expensive computation, we report an averaged performance over only three runs. CMA-ES performs worse because it still depends on fixed and pre-defined features, therefore in a more complex task it can only find a policy whose performance is up to the power and quality of the selected features. CMA-ES-A uses a non-principled scaling on its parameter space which inherently consists of two parts: the weights $\{\mathbf{w}\}$ and the state information $\{\mathbf{s}_i\}$. This scaling does not capture correctly distances between points on the parameter space, hence it leads to a non-optimal solution.

6 Conclusion

This paper proposes a CMA-ES-RKHS framework that generalises CMA-ES to handle functional optimisation in which the search is handled over a function space. The fact that the function space is modeled in reproducing kernel Hilbert space results in analytic update rules for CMA-ES-RKHS. On the other hand, the solution function attains compactness and flexibility characteristics. We apply CMA-ES-RKHS for direct policy search in which the policy is modeled in RKHS. Our experiments show that both CMA-ES-RKHS and direct policy search via CMA-ES-RKHS are able to represent a complex solution function compactly and adaptively. The result shows many interesting aspects and results of CMA-ES-RKHS: (1) explicitly handling functional optimisation in principle; (2) overcoming the issue of hand-designed feature in many practical applications of CMA-ES. Though offering many advantages, CMA-ES-RKHS is a kernel method therefore it also suffers from the problem of expensive computation. A study to investigate the way how to scale it will be a very promising research direction. There are also a number of other potential future research directions. A thorough study into hyperparameters of CMA-ES-RKHS will definitely be important to CMS-ES-RKHS. Moreover, more theoretical work and practical applications of CMA-ES-RKHS would be a very interesting future research direction.

Acknowledgements This research is funded by Vietnam National Foundation for Science and Technology Development (NAFOSTED) under Grant No. 102.05-2016.18; and by the Basic Science Research Program through National Research Foundation of Korea (NRF) of the Ministry of Education, Science, and Technology under Grant No. NRF-2017R1D1A1B04036354 and Kyung Hee University under Grant No. KHU-20160601.

References

1. R.J. Adler, *The Geometry of Random Fields* (Wiley, New York, 1981)
2. J.A.D. Bagnell, J. Schneider, Policy search in reproducing kernel Hilbert space. Technical Report CMU-RI-TR-03-45, Robotics Institute, Pittsburgh, PA (2003)
3. J.B. Conway, *A Course in Functional Analysis*, vol. 96 (Springer, Berlin, 2013)
4. M.P. Deisenroth, G. Neumann, J. Peters et al., A survey on policy search for robotics. Found. Trends® Robot. **2**(1–2), 1–142 (2013)
5. A. Doerr, N.D. Ratliff, J. Bohg, M. Toussaint, S. Schaal, Direct loss minimization inverse optimal control, in *Robotics: Science and Systems XI*, Sapienza University of Rome, Rome, Italy, July 13–17 (2015)
6. J. Dong, M. Mukadam, F. Dellaert, B. Boots, Motion planning as probabilistic inference using Gaussian processes and factor graphs, in *Robotics: Science and Systems XII*, University of Michigan, Ann Arbor, Michigan, USA, June 18–June 22 (2016)
7. M. Ghavamzadeh, Y. Engel, M. Valko, Bayesian policy gradient and actor-critic algorithms. J. Mach. Learn. Res. **17**(1), 2319–2371 (2016)
8. S. Ha, C.K. Liu, Iterative training of dynamic skills inspired by human coaching techniques. ACM Trans. Graph. **34**(1), :1–1:11 (2014)
9. S. Ha, C.K. Liu, Evolutionary optimization for parameterized whole-body dynamic motor skills, in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1390–1397 (2016)
10. N. Hansen, The CMA evolution strategy: a tutorial. CoRR arXiv:1604.00772 (2016)
11. N. Hansen, A. Auger, Principled design of continuous stochastic search: from theory to practice, in *Theory and principled methods for the design of metaheuristics*, ed. by Y. Borenstein, A. Moraglio, pp. 145–180 (Springer, Berlin, 2014)
12. N. Hansen, S.D. Müller, P. Koumoutsakos, Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). Evol. Comput. **11**(1), 1–18 (2003)
13. V. Heidrich-Meisner, C. Igel, Similarities and differences between policy gradient methods and evolution strategies, in *The European Symposium on Artificial Neural Networks (ESANN)*, pp. 149–154 (2008)
14. V. Heidrich-Meisner, C. Igel, Hoeffding and Bernstein races for selecting policies in evolutionary direct policy search, in *Proceedings of the International Conference on Machine Learning, ICML*, pp. 401–408 (2009)
15. V. Heidrich-Meisner, C. Igel, Neuroevolution strategies for episodic reinforcement learning. J. Algorithms **64**(4), 152–168 (2009)
16. H. Kadri, E. Duflos, P. Preux, S. Canu, A. Rakotomamonjy, J. Audiffren, Operator-valued kernels for learning from functional response data. J. Mach. Learn. Res. **16**, 1–54 (2015)
17. M.F. Kasim, P.A. Norreys, Infinite dimensional optimistic optimisation with applications on physical systems. arXiv preprint arXiv:1611.05845 (2016)
18. E. Kreyszig, *Introductory Functional Analysis with Applications* (Wiley, New York, 1989)
19. G. Lever, R. Stafford, Modelling policies in mdps in reproducing kernel Hilbert space, in *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2015*, San Diego, California, USA, May 9–12 (2015)
20. S. Mannor, R.Y. Rubinstein, Y. Gat, The cross entropy method for fast policy search, in *Machine Learning, Proceedings of the Twentieth International Conference (ICML)*, pp. 512–519 (2003)
21. Z. Marinho, B. Boots, A.D. Dragan, A. Byravan, G.J. Gordon, S. Srinivasa, Functional gradient motion planning in reproducing kernel hilbert spaces, in *Robotics: Science and Systems XII*, University of Michigan, Ann Arbor, Michigan, USA, June 18–22 (2016)

22. C.A. Micchelli, M. Pontil, On learning vector-valued functions. Neural Comput. **17**(1), 177–204 (2005)
23. J. Peters, S. Schaal, Natural actor-critic. Neurocomputing **71**(7), 1180–1190 (2008)
24. M.L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming* (Wiley, New York, 2014)
25. C.E. Rasmussen, C.K.I. Williams, *Gaussian Processes for Machine Learning* (The MIT Press, Cambridge, 2006)
26. R. Rubinstein, The cross-entropy method for combinatorial and continuous optimization. Methodol. Comput. Appl. Probab. **1**(2), 127–190 (1999)
27. R.Y. Rubinstein, D.P. Kroese, *The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation and Machine Learning* (Springer, Berlin, 2013)
28. E.A. Rückert, G. Neumann, M. Toussaint, W. Maass, Learned graphical models for probabilistic planning provide a new class of movement primitives. Front. Comput. Neurosci. **6**, 97 (2013)
29. B. Schölkopf, A.J. Smola, *Learning with Kernels Support Vector Machines, Regularization, Optimization, and Beyond. Adaptive Computation and Machine Learning Series* (MIT Press, Cambridge, 2002)
30. B. Schölkopf, A.J. Smola, K. Müller, Kernel principal component analysis, in *The International Conference on Artificial Neural Networks (ICANN)*, pp. 583–588 (1997)
31. F. Stulp, O. Sigaud, Path integral policy improvement with covariance matrix adaptation, in *Proceedings of the 29th International Conference on Machine Learning, ICML 2012*, Edinburgh, Scotland, UK, June 26–July 1 (2012)
32. R. Sutton, A. Barto, *Reinforcement Learning: An Introduction* (Cambridge University Press, Cambridge, 1998)
33. R.S. Sutton, D.A. McAllester, S.P. Singh, Y. Mansour, Policy gradient methods for reinforcement learning with function approximation, in *Conference on Neural Information Processing Systems (NIPS)*, pp. 1057–1063 (1999)
34. J. Tan, Y. Gu, G. Turk, C.K. Liu, Articulated swimming creatures. ACM Trans. Graph. **30**(4), 58 (2011)
35. M. Toussaint, Newton methods for k-order Markov constrained motion problems. CoRR arXiv :abs/1407.0414 (2014)
36. L.P. Tuyen, N.A. Vien, T. Chung, A deep hierarchical reinforcement learning algorithm in partially observable markov decision processes. IEEE Access **6**, 49089–49102 (2018)
37. M. Ulbrich, Optimization methods in Banach spaces, in *Optimization with PDE Constraints*, ed. by Y. Borenstein, A. Moraglio, pp. 97–156 (Springer, Berlin, 2009)
38. H. van Hoof, J. Peters, G. Neumann, Learning of non-parametric control policies with high-dimensional state features, in *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2015*, San Diego, California, USA, May 9–12 (2015)
39. N.A. Vien, V.-H. Dang, T. Chung, A covariance matrix adaptation evolution strategy for direct policy search in reproducing kernel Hilbert space, in *The 9th Asian Conference on Machine Learning (ACML)* (2017)
40. N.A. Vien, P. Englert, M. Toussaint, Policy search in reproducing kernel Hilbert space, in *International Joint Conferences on Artificial Intelligence (IJCAI)*, pp. 2089–2096 (2016)
41. N.A. Vien, H. Yu, T. Chung, Hessian matrix distribution for bayesian policy gradient reinforcement learning. Inf. Sci. **181**(9), 1671–1685 (2011)
42. P. Vincent, Y. Bengio, Kernel matching pursuit. Mach. Learn. **48**(1–3), 165–187 (2002)
43. J.M. Wang, S.R. Hamner, S.L. Delp, V. Koltun, Optimizing locomotion controllers using biologically-based actuators and objectives. ACM Trans. Graph. **31**(4), 25:1–25:11 (2012)
44. D. Wierstra, T. Schaul, T. Glasmachers, Y. Sun, J. Peters, J. Schmidhuber, Natural evolution strategies. J. Mach. Learn. Res. **15**(1), 949–980 (2014)
45. R.J. Williams, Simple statistical gradient-following algorithms for connectionist reinforcement learning. Mach. Learn. **8**(3–4), 229–256 (1992)
46. X. Xu, D. Hu, X. Lu, Kernel-based least squares policy iteration for reinforcement learning. IEEE Trans. Neural Netw. **18**(4), 973–992 (2007)