Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)

Sequential Classification-Based Optimization for Direct Policy Search*

Yi-Qi Hu, Hong Qian, Yang Yu
National Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, 210023, China
Collaborative Innovation Center of Novel Software Technology and Industrialization, Nanjing, 210023, China
yuy@nju.edu.cn

Abstract

Direct policy search often results in high-quality policies in complex reinforcement learning problems, which employs some optimization algorithms to search the parameters of the policy for maximizing the its total reward. Classification-based optimization is a recently developed framework for derivative-free optimization, which has shown to be effective and efficient for non-convex optimization problems with many local optima, and may provide a power optimization tool for direct policy search. However, this framework requires to sample a batch of solutions for every update of the search model, while in reinforcement learning, the environment often offers only sequential policy evaluation. Thus the classification-based optimization may not efficient for direct policy search, where solutions have to be sampled sequentially. In this paper, we adapt the classification-based optimization for sequential sampled solutions by forming the sample batch via reusing historical solutions. Experiments on a helicopter hovering task and controlling tasks in OpenAI Gym show that the new algorithm significantly improve the performance from several state-of-the-art derivative-free optimization approaches.

Introduction

With the goal of learning an optimal policy from autonomous interactions with the environment, reinforcement learning is in the core of artificial intelligence. Previous studies (Wang et al. 2007; El-Fakdi, Carreras, and Palomeras 2005) show that direct policy search, which employs some optimization algorithms to find parameters of the policy to maximize the received total reward, can have better performance than traditional reinforcement learning approaches. The policy optimization task is usually quite complex, involving many local optima, for which derivative-free optimization can be useful. Many derivative-free optimization methods are model-based, and mainly consist of a cycle of two steps: sampling solutions from the current model, and updating the model from the sampled solutions along with their evaluation values. Through the cycle, they are expected to iteratively find better solutions. Typical derivative-free optimization algorithms include evolutionary algorithms (Hansen, Müller, and Koumoutsakos 2003; Golberg 1989; Larranaga and Lozano 2002; Neumann and Wegener 2007) and similar methods such as cross-entropy methods (de Boer et al. 2005), however, these methods have little theoretical support, except very few of them (Qian, Yu, and Zhou 2015a; 2015b; Yu, Yao, and Zhou 2012). Recently emerged optimistic optimization methods (Munos 2011; 2014) and Bayesian optimization methods (Bull 2011; de Freitas et al. 2012) are better theoretically supported, but often suffer from poor scalability.

Classification-based optimization is a recently developed theoretical framework of model-based derivative-free optimization method. Its implementation, the RACOS algorithm (Yu, Qian, and Hu 2016) has shown outstanding performance in various applications (Yu, Qian, and Hu 2016; Qian, Hu, and Yu 2016; Qian and Yu 2016), and may provide a powerful optimization tool for direct policy search. However, we have noticed an inefficient component of RACOS: like other batch-mode algorithms, it updates its model only after sampling and evaluating a batch of solutions, even if the model samples out unnecessary bad solutions. Meanwhile, in direct policy search, policies often have to be evaluated sequentially but not in parallel, such as evaluating policise on a single physical robot. Therefore, the original batch-mode RACOS may not efficient for direct policy search.

In this paper, we propose the sequential classification-based optimization, which updates the model after sampling every solution by reusing historical samples. We first analyze when the sequential-mode need fewer samples than the batch-mode. Then we design a new algorithm named Sequential RACOS (SRACOS), with which three replacing strategies for maintaining the historical samples are proposed. We empirically compare SRACOS with RACOS and several other model-based optimization algorithms on two synthetic testing functions, and apply them to direct policy search for 9 controlling tasks, where an artificial neural network is used as the policy and optimized. Experiment results show that SRACOS can significantly improve the performance of RACOS as well as the other state-of-the-art derivative-free optimization methods.

The rest four sections presents the background, new algorithm, experiments, and conclusion, respectively.

*This research was supported by the NSFC (61375061, 61333014), JiangsuSF (BK20160066), Foundation for the Author of National Excellent Doctoral Dissertation of China (201451), and 2015 Microsoft Research Asia Collaborative Research Program.

Background

Reinforcement learning can often be described by the Markov decision process (MDP) presented as a tuple $\langle S, A, P_{sa}, R\rangle$, where S is the state space, A is the action space and P_{sa} is the transition function for taking action a in state s, and $R : S \to \mathbb{R}$ is the reward function. The dynamic of MDP is as follows: the environment initializes a state s_0; we choose an action $a_0 \in A$ based on s_0; according to $P_{s_0a_0}$, the environment turns to a new state s_1 with probability; we then pick up a new action a_1 in s_1, and so on. The sequence of $s_0, s_1, \ldots$ represents a trajectory. A mapping $\pi : S \to A$ is a policy which determines the actions based on current state. The goal of reinforcement learning is to search for a policy that maximizes the long-term cumulated reward.

Direct policy search employs an optimization algorithm to search in the parameter space of the policy for maximizing the cumulated reward of the policy. For example, neural network is a commonly used model for policy, and its weights $w = \{w_1, w_2, \ldots, w_n\}$ are the parameters. The cumulated reward of executing a neural network policy, e.g., $f(w) = \sum_{i=1}^{T} R_i$ is the objective to be maximized. Treating the optimization of policy as a black-box optimization problem, derivative-free optimization methods have been used to directly optimize the parameters of the policy.

For black-box optimization, we consider general minimization problems in continuous domains. Let X denote a bounded solution space that is a compact subset of $\mathbb{R}^n$, and $f : X \to \mathbb{R}$ denote a minimization problem. Assume that there exist x^*, $\hat{x} \in X$ such that $f(x^*) = \min_{x\in X} f(x)$ and $f(\hat{x}) = \max_{x\in X} f(x)$. let $\mathcal{F}$ denote the set of all functions that satisfy this assumption. Given $f \in \mathcal{F}$, the *minimization problem* is to find a solution $x^* \in X$ s.t. $\forall x \in X : f(x^*) \leq f(x)$. For black-box optimization, given a solution x, only the objective function $f(x)$ is accessible for evaluating x. Other information of f such as the gradient is inaccessible.

RACOS is a recently proposed classification-based derivative-free optimization algorithm (Yu, Qian, and Hu 2016). Unlike other derivative-free optimization algorithms, the sampling region of RACOS is learned by a simple classifier. RACOS shows outstanding performance in empirical experiments and theoretical study. Its pseudo-code is presented in Algorithm 1. It starts from a set S_0 of solutions uniformly sampled from the solution space (line 1), where $\mathcal{U}_X$ denotes the uniform distribution over X. We will get a solution-value tuple set B_0 after querying objective function for each solution in S_0 (line 2). Lines 3 and 11 record the best-so-far solution. In line 5, the old tuple set B_{t-1} is split into the positive set B_t^+ consisting of the tuples of the best k solutions, and the negative set B_t^- consisting of the rest of tuples. Then a loop is followed to sample m solutions. In the t-th iteration (line 6 to 10), RACOS learns a hypothesis h_i by the learning algorithm $\mathcal{C}$, where h_i is an axis-parallel box that discriminates a randomly chosen positive solution from all negative solutions. The details of the learning algorithm can be find in (Yu, Qian, and Hu 2016). Then RACOS samples a solution x_i from the uniform distribution over D_{h_i} with λ probability or over X with $1-\lambda$ probability (line 8). Here, $D_h = \{x \in X | h(x) = +1\}$ is the region of positive solutions represented by h. Finally RACOS will return the best-so-far tuple $(\tilde{x}, \tilde{y})$.

Algorithm 1 RACOS (batch-mode)

Input:
f: Objective function to be minimized;
$\mathcal{C}$: A binary classification algorithm;
λ: Balancing parameter;
$T \in \mathbb{N}^+$: Number of iterations;
$m \in \mathbb{N}^+$: Sample size in each iteration;
$k \in \mathbb{N}^+ (\leq m)$: Number of positive samples;
`Sampling`: Sampling sub-procedure;
`Selection`: Decide the positive/negative solutions.

Procedure:
1: Collect $S_0 = \{x_1, \ldots, x_m\}$ by i.i.d. sampling from $\mathcal{U}_X$
2: $B_0 = \{(x_1, y_1), \ldots, (x_m, y_m)\}, \forall x_i \in S_0 : y_i = f(x_i)$
3: Let $(\tilde{x}, \tilde{y}) = \arg\min_{(x,y)\in B^+} y$
4: **for** $t = 1$ to T **do**
5: $(B_t^+, B_t^-) = \texttt{Selection}(B_{t-1}; k)$, $B_t = B_t^+$
6: **for** $i = 1$ to m **do**
7: $h_i = \mathcal{C}(B_t^+, B_t^-)$
8: $x_i = \begin{cases} \texttt{Sampling}(\mathcal{U}_{D_{h_i}}) & \text{w.p. } \lambda \\ \texttt{Sampling}(\mathcal{U}_X) & \text{w.p. } 1-\lambda \end{cases}$
9: $y_i = f(x_i)$ and let $B_t = B_t \cup \{(x_i, y_i)\}$
10: **end for**
11: $(\tilde{x}, \tilde{y}) = \arg\min_{(x,y)\in B_t \cup \{(\tilde{x},\tilde{y})\}} y$
12: **end for**
13: **return** $(\tilde{x}, \tilde{y})$

RACOS is a batch-mode algorithm: the model h_i depends on B_t^+ and B_t^- in line 7 of the algorithm, and in the loop of iteration (line 6 to 10), those two sets are unchanged. That is to say, the sampling regions are generated from the same distribution, even if this distribution is not good enough. Batch-mode sampling probably produces redundancy and is not suitable for sequential evaluating problem like direct policy search. So that we propose the sequential RACOS.

Sequential RACOS

Algorithm

The idea of converting RACOS to be sequential is straightforward: update the model (B^+, B^-) from the batch formed by reusing historical solutions after each sampling. Therefore it is important to find a way to maintain historical solutions. The sequential algorithm SRACOS is presented in Algorithm 2, where we omit the inherited input from Algorithm 1.

With the same as Algorithm 1, $\texttt{Sampling}(\mathcal{S})$ is used to sample a solution from the given distribution $\mathcal{S}$. $\texttt{Selection}(B; k)$ splits the solution-value tuple set B into a positive set and a negative set, where the positive set contains k tuples of best-so-far solutions. $\texttt{Replace}(a, A, \text{'s'})$ is replacing sub-procedure for SRACOS means using a to replace a tuple from set A with some strategy 's', and the return values are a replaced tuple and the updated tuple set.

Algorithm 2 Sequential RACOS (SRACOS)

Input: (extra input than RACOS)
$N \in \mathbb{N}^+$: Budget;
$r = m + k$;
`Replace`: Replacing sub-procedure.

Procedure:
1: Collect $S = \{x_1, \ldots, x_r\}$ by i.i.d. sampling from $\mathcal{U}_X$
2: $B = \{(x_1, y_1), \ldots, (x_r, y_r)\}, \forall x_i \in S : y_i = f(x_i)$
3: $(B^+, B^-) = \texttt{Selection}(B; k)$
4: Let $(\tilde{x}, \tilde{y})$ = argmin$_{(x,y)\in B^+} y$
5: **for** $t = r + 1$ to N **do**
6: $\quad h = \mathcal{C}(B^+, B^-)$
7: $\quad x = \begin{cases} \texttt{Sampling}(\mathcal{U}_{D_h}) & \text{w.p. } \lambda \\ \texttt{Sampling}(\mathcal{U}_X) & \text{w.p. } 1-\lambda \end{cases}$
8: $\quad y = f(x)$
9: $\quad [(x', y'), B^+]$ = `Replace`$((x, y), B^+,$ 'strategy_P')
10: $\quad [, B^-]$ = `Replace`$((x', y'), B^-,$ 'strategy_N')
11: $\quad (\tilde{x}, \tilde{y}) = \arg\min_{(x,y)\in B^+ \cup \{(\tilde{x},\tilde{y})\}} y$
12: **end for**
13: **return** $(\tilde{x}, \tilde{y})$

Three replacing strategies that we are considering are as follows: replacing the tuple with worst evaluation value in A (WR-); replacing a tuple in A randomly (RR-) and replacing the tuple which has largest margin from the best-so-far solution (LM-).

In Algorithm 2, after initialization, SRACOS will get two tuple sets B^+ and B^-, denoting the positive solution-value tuple set and the negative one. The method of generating a new solution (lines 7 to 8) is the same as RACOS. After getting a new tuple (x, y), SRACOS updates B^+ and B^- immediately. When updating B^+ (line 9), we use the 'strategy_P'. Because B^+ must contain the best-so-far solutions, 'strategy_P' can only be 'WR-', i.e., a solution with the worst evaluation value is removed from $B^+ \cup \{(x, y)\}$ and the rest of set is the new B^+. The removed tuple (x', y') is used to update B^- using 'strategy_N' in line 10. 'strategy_N' can be any one of three strategies mentioned above. In experiments section, we will prove that selection of 'strategy_N' has no influence on convergence rate of SRACOS empirically. In the end, SRACOS returns the best solution-value tuple $(\tilde{x}, \tilde{y})$.

Query complexity

For a subset $D \subseteq X$, let $\#D = \int_{x\in X} \mathbb{I}[x \in D]\,\mathrm{d}x$, where $\mathbb{I}[\cdot]$ is the indicator function. Define $|D| = \#D/\#X$. Let $D_\alpha = \{x \in X \,|\, f(x) \le \alpha\}$, and $D_\epsilon = \{x \in X \,|\, f(x) - f(x^*) \le \epsilon\}$ for $\epsilon > 0$. A hypothesis is a mapping $h\colon X \to \{-1, +1\}$. Let $\mathcal{H} \subseteq \{h\colon X \to \{-1, +1\}\}$ be a hypothesis space. Let $D_h = \{x \in X \,|\, h(x) = +1\}$ for hypothesis $h \in \mathcal{H}$, i.e., the positive class region represented by h. Denote $\mathcal{U}_{D_h}$ the uniform distribution over D_h, respectively, and denote $\mathcal{T}_h$ the distribution defined on D_h induced by h. Let $\mathcal{S}_t = \lambda\mathcal{U}_{D_{h_t}} + (1-\lambda)\mathcal{U}_X$ be the sampling distribution in iteration t, $\mathcal{D}_t$ be the distribution under which the classifier is trained in iteration t, $R_{\mathcal{D}_t}$ denote the generalization error of $h_t \in \mathcal{H}$ under the distribution $\mathcal{D}_t$, D_{KL} denote the Kullback-Leibler (KL) divergence between two probability distributions, and N denote the number of iterations.

The complexity of an algorithm is measured by the (ϵ,δ)-Query Complexity as Definition 1 (Yu and Qian 2014; Yu, Qian, and Hu 2016). It counts the total number of calls to the objective function by an algorithm before it finds a solution that reaches the approximation level ϵ, with high probability.

DEFINITION 1 ((ϵ,δ)-Query Complexity)
Given $f \in \mathcal{F}$, an algorithm $\mathcal{A}$, $0 < \delta < 1$ and $\epsilon > 0$, the (ϵ,δ)-query complexity is the number of calls to f such that, with probability at least $1-\delta$, $\mathcal{A}$ finds at least one solution $\tilde{x} \in X \subseteq \mathbb{R}^n$ satisfying $f(\tilde{x}) - f(x^*) \le \epsilon$, where $f(x^*) = \min_{x\in X} f(x)$.

Under the conditions of error-target θ-dependence and γ-shrinking rate (Yu, Qian, and Hu 2016), we derive a general upper bound of the query complexity of sequential classification-based optimization algorithms as Theorem 1.

THEOREM 1
Given $f \in \mathcal{F}$, $0 < \delta < 1$ and $\epsilon > 0$, if a sequential classification-based optimization algorithm has error-target θ-dependence and γ-shrinking rate, then its (ϵ,δ)-query complexity is upper bounded by

$$O\left(\max\left\{\frac{1}{|D_\epsilon|}\left((1-\lambda)+\frac{\lambda}{\gamma(N-r)}\sum_{t=r+1}^{N}\Phi_t\right)^{-1}\ln\frac{1}{\delta}, N\right\}\right),$$

where $\Phi_t = \left(1 - R_{\mathcal{D}_t} - \#X\sqrt{\frac{1}{2}D_{KL}(\mathcal{D}_t\|\mathcal{U}_X)} - \theta\right) \cdot |D_{\alpha_t}|^{-1}$ *and $\#X$ is the volume of X.*

We can observe from Theorem 1 that the smaller θ, γ, $R_{\mathcal{D}_t}$ and $D_{KL}(\mathcal{D}_t\|\mathcal{U}_X)$, the better the query complexity. Note that the training data in SRACOS reuses samples from historical models, $\mathcal{D}_t$ is a combination of the uniform sampling, and the sampling from a set of models determined by the strategy of data reuse. Due to the shifted training distribution, the generalization error $R_{\mathcal{D}_t}$ may be worse than that in the batch-mode, meanwhile, the $D_{KL}(\mathcal{D}_t\|\mathcal{U}_X)$ can be better than that in the batch-mode as the training distribution is more spreaded.

We can have a comparison of the query complexity bound of the sequential classification-based algorithm of Theorem 1 with the bound of the batch-mode algorithm in (Yu, Qian, and Hu 2016). Let $\mathcal{D}_t^S$ and $\mathcal{D}_t^B$ denote the distribution under which the classifier is trained in iteration t of the sequential-mode and the batch-mode, respectively, and also denote $R_{\mathcal{D}_t^S}$ and $R_{\mathcal{D}_t^B}$ as the generalization error of them, respectively. The comparison result of the query complexity upper bound is shown in Theorem 2.

THEOREM 2
Ignoring the constant factor and fixing θ and γ, the sequential classification-based optimization algorithm can have a better (or worse) query complexity upper bound than the batch-mode if for any iteration t

$$R_{\mathcal{D}_t^S} < (\text{or } >)\frac{1}{1-\lambda}R_{\mathcal{D}_t^B} - \#X\sqrt{\frac{1}{2}D_{KL}(\mathcal{D}_t^S\|\mathcal{U}_X)}.$$

Theorem 2 discloses that the spreading training distribution $\mathcal{D}_t^S$ of the sequential algorithm makes room for its worse generalization error of the classifier. If the room is large enough, the query complexity of the sequential algorithm can be better than the batch-mode. The proofs of Theorem 1 and 2 are both presented in the appendix[1] due to the space limitation.

Experiments

We empirically compare SRACOS with RACOS (code from https://github.com/eyounx/RACOS), as well as other state-of-the-art derivative-free optimization algorithms including CMA-ES (Hansen, Müller, and Koumoutsakos 2003) (code from https://pypi.python.org/pypi/cma), differential evolution algorithm (Storn and Price 1997) (DE, code implemented in SciPy https://www.scipy.org/), cross-entropy method (CE, implemented by the authors in python according to https://en.wikipedia.org/wiki/Cross-entropy_method) and a Bayesian optimization method with exponential convergence (Kawaguchi, Kaelbling, and Lozano-Pérez 2015) (IMGPO, code from http://lis.csail.mit.edu/code/imgpo.html). All algorithms are used with their default parameters.

We firstly investigate properties of SRACOS on two synthetic testing functions, and then we apply to several reinforcement learning tasks including helicopter hovering control task (Kim et al. 2003) and controlling tasks in OpenAI Gym, an open source environment for reinforcement learning (http://gym.openai.com).

[1]The appendix presenting all the proofs can be found in the homepage of the author http://cs.nju.edu.cn/yuy.

On synthetic functions

We choose two benchmark testing functions: the convex Sphere function defined as $f(x) = \sum_{i=1}^{n}(x_i - 0.2)^2$, and the highly non-convex Ackley function defined as

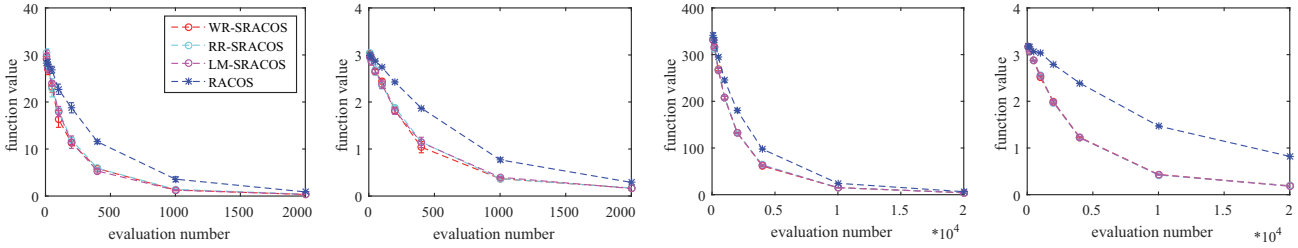
$$f(x) = -20e^{(-\frac{1}{5}\sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i-0.2)^2})} - e^{\frac{1}{n}\sum_{i=1}^{n}\cos 2\pi(x_i-0.2)} + e + 20.$$

The functions are minimized within the solution space $X = [-1, 1]^n$, of which the minimum value is 0. Note that we shift the optimal solution by 0.2 to avoid possible optimization bias to the origin point.

To study the convergence speed against the number of function evaluations, we choose $n = 100$ and $1,000$, and set the total number of function evaluations to be $20n$ for all algorithms and objective functions. Each algorithm is repeated 15 times independently, and the mean value and standard deviation of the top-5 results are reported.

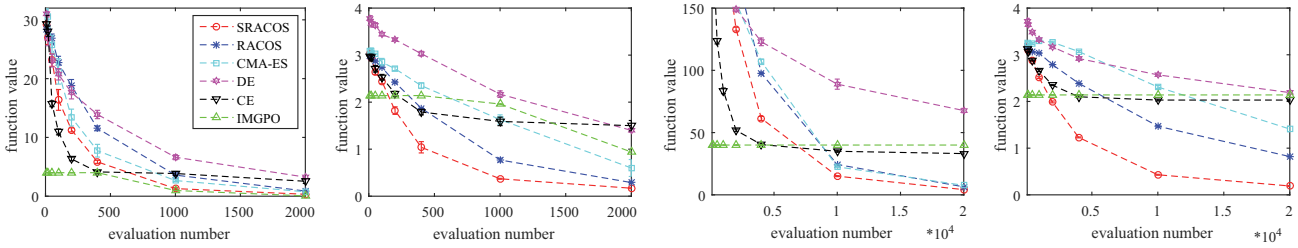
First of all, we investigate how 'strategy_N' influences convergence of SRACOS. The convergence speeds of SRACOS with 'WR-', 'RR-', 'LM-' strategies and RACOS are compared. From Figure 1, the curves of SRACOS with the three strategies are almost overlapped, indicating that the difference among the replacing strategies can be ignored. In the rest of the experiments, we will choose WR-SRCOS as representation of SRACOS.

We then compare SRACOS with other derivative-free algorithms in convergence rate. The results are plotted in Figure 2. Firstly, we compare SRACOS with RACOS: it is clear that SRACOS is consistently better than RACOS in all situations. In low dimension convex problem (i.e., $n = 100$, Sphere), the Bayesian optimization method IMGPO shows the highest convergence speed, but IMGPO is hard to handle



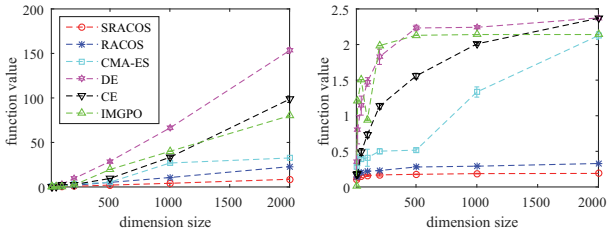
(a) Sphere, $n = 100$ (b) Ackley, $n = 100$ (c) Sphere, $n = 1000$ (d) Ackley, $n = 1000$

Figure 1: Comparison on the impact of the three replacing strategy in SRACOS on Sphere and Ackley function with dimension size $n = 100$ and $n = 1000$.



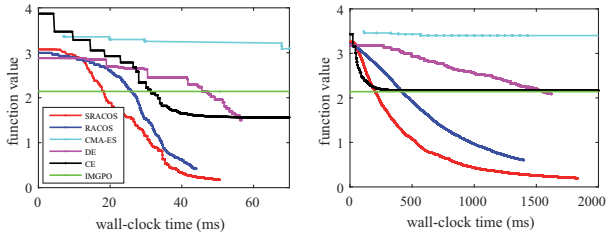
(a) Sphere, $n = 100$ (b) Ackley, $n = 100$ (c) Sphere, $n = 1000$ (d) Ackley, $n = 1000$

Figure 2: Comparison of the convergence speed on Sphere and Ackley functions with dimension size $n = 100$ and $n = 1000$.



(a) Sphere (b) Ackley

Figure 3: Comparison the scalability with $20n$ evaluations on Sphere and Ackley functions.



(a) Ackley, $n = 100$ (b) Ackley, $n = 1,000$

Figure 4: Comparison on the convergence speed against the wall-clock time, on $n = 100$ and $n = 1,000$ Ackley functions with $20n$ evaluations.

high dimension problems (i.e., $n = 1,000$, Sphere) or problems with many local optima (i.e., Ackley). While SRACOS shows the best convergence rate in those situations.

To study the scalability w.r.t. the solution space dimension n, we choose n as 10, 20, 50, 100, 200, 500, $1,000$, $2,000$, set the maximum number of function evaluations to be $20n$, and set m to be 5, 5, 10, 10, 20, 20, 40, 40 corresponding to each n setting for all algorithms. Figure 3 shows that the curves of SRACOS has the lowest growing rate as the dimensionality increasing. It indicates that SRACOS has the best scalability among the compared algorithms. Specially, we can also observed that RACOS already has better scalability than other compared algorithms, while SRACOS improves the scalability of RACOS even further.

Furthermore, we consider the actual computation costs and compare the convergence speed against the wall-clock time. Figure 4 presents the results. It can be observed that SRACOS takes more computation time than RACOS (i.e., longer curve), but even with the same time, SRACOS achieves better solution except the very beginning.

On reinforcement learning for controlling tasks

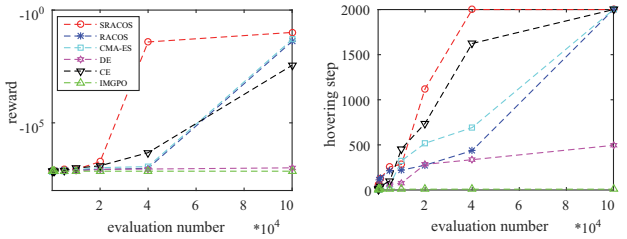
In this section, we apply the optimization for direct policy search. In the following experiments, the policy is a feed forward fully connected neural network. We will employ the optimization to find the best weights of the neural network that maximizing the cumulated reward.

Helicopter hovering control task Helicopter flight is regarded to be a challenging control problem. In (Kim et al. 2003), they successfully applied reinforcement learning to design of controller for helicopter hovering task and implemented the helicopter hovering simulation environment. In this environment helicopter is required to hover in a limited space. If the helicopter moves out of the boundary within limited steps, we regard that helicopter has crashed and the policy will get a very bad score.

Previously, neural network policy model is considered suitable for this task (Rogier and Whiteson 2011). A full-connected neural network without any hidden nodes is employed as the policy model. The system will fly helicopter under policy's control up to 2,000 steps. The sum of rewards in these steps is used as the evaluation of the policy. Set $w \in [-10, 10]^n$. On this task, the state space has 13 dimensions and action space has 4 dimensions, so there are 52 neural network weights that is also the dimension number of the search space. Helicopter hovering control task can be presented as: $\text{argmax}_{w\in[-10,10]^n} f(w)$. When using SRACOS to optimize w, a derivative-free optimization algorithm generates a w, then system will fly the helicopter with w as the parameters of the policy model. After getting $f(w)$, the algorithm will update its model and then generate a new w and so on.

We compare SRACOS with RACOS, CMA-ES, DE, CE, IMGPO mentioned above. The number of evaluations is set as 10^5 for all algorithms. Each algorithm is repeated 15 times independently. Figure 5 shows the performance of the best result, and Table 1 reports the average performance of the top-5 highest reward.

On helicopter hovering control task, the reward of policy should be as large as possible. If reward is larger than -2×10^6, it indicates that helicopter has not crashed within 2,000 steps. Table 1 shows the top-5 average performance on the rewards and the hovering steps, and the success rates



(a) Reward (b) Hovering steps

Figure 5: Comparison of the convergence speed on (a) the reward and (b) the hovering steps.

Table 1: Average of the top-5 rewards and the hovering steps, and the success rate of the 15 repeats. The values in bold mean best in each item.

Algorithms	Reward	Hovering step	Success rate
SRACOS	$\mathbf{-9.72 \times 10^5 \pm 2.17 \times 10^6}$	**1,837±364**	**4/15**
RACOS	$-3.18 \times 10^6 \pm 3.34 \times 10^6$	1,477±535	2/15
CMA-ES	$-5.29 \times 10^6 \pm 4.88 \times 10^6$	1,280±673	2/15
DE	$-1.02 \times 10^7 \pm 5.92 \times 10^5$	453±74	0/15
CE	$-5.48 \times 10^6 \pm 3.35 \times 10^6$	1,121±525	1/15
IMGPO	$-1.18 \times 10^7 \pm 2.66 \times 10^5$	256±31	0/15

Table 2: The mean reward and the standard deviation of the best found policy by each algorithm. The numbers in bold mean the best cumulated reward in each row. The mark ↓ means the reward is the smaller the better, and ↑ means the larger the better.

Task	SRACOS	RACOS	CMA-ES	DE	CE	IMGPO
Acrobot↓	**156.60**±18.48	169.70±14.15	181.10±42.66	161.10±45.91	534.00±774.69	1545.00±736.14
MountainCar↓	**132.40**±39.60	141.50±0.97	190.60±26.89	153.00±48.44	3048.90±4796.7	5171.40±5090.29
HalfCheetah↑	**36719.90**±8288.84	27961.18±7493.08	20191.83±984.95	17250.21±305.01	14714.05±5169.94	10355.83±93.16
Humanoid↑	**502.57**±88.03	398.03±19.23	357.09±124.77	428.97±67.89	423.58±27.88	209.75±3.16
Swimmer↑	**3692.65**±7.89	3495.16±72.75	3202.33±11.98	3096.44±20.08	3002.26±46.14	270.73±3.27
Ant↑	**2114.14**±2290.57	1215.28±1487.81	63.66±12.00	653.56±969.84	722.88±531.73	42.52±3.57
Hopper↑	**10818.98**±501.11	9892.70±417.85	9986.81±0.96	9931.70±1.35	5149.48±5006.35	136.28±23.04
LunarLander↑	**238.14**±15.61	193.45±35.62	132.62±35.18	125.00±93.86	92.45±110.81	64.29±27.32

Table 3: Parameters of the Gym tasks, including the dimensionally of the state space d_{State}, the number of actions, the layers and the nodes of the feed-forward neural networks, the number of weights, and the horizon steps.

Task name	d_{State}	#Actions	NN nodes	#Weights	Horizon
Acrobot	6	1	5, 3	48	2,000
MountainCar	2	1	5	15	10,000
HalfCheetah	17	6	10	230	10,000
Humanoid	376	17	25	9825	50,000
Swimmer	8	2	5, 3	61	10,000
Ant	111	8	15	1785	10,000
Hopper	11	3	9, 5	159	10,000
LunarLander	8	1	5, 3	58	10,000

(hovering for 2,000 steps is a successful try) of the 15 repeats. The numbers in bold mean best performance in each item. It indicates that SRACOS has best performance in every item. We can get same conclusions from (a) and (b) in Figure 5: the policies generated by SRACOS, RACOS, CMA-ES and CE can reach maximum step within 10^5 evaluations. But the helicopter can't hover more than 500 steps with the policies from DE and IMGPO. And it only costs SRACOS about $40,000$ evaluations to reach the maximum step, faster than RACOS, CMA-ES and CE.

Gym tasks In the OpenAI Gym environment, we use eight existing controlling tasks, 'Acrobot', 'MountainCar', 'HalfCheetah', 'Humanoid', 'Swimmer', 'Ant', 'Hopper' and 'LunarLander' to test the algorithms. We also apply neural network as policy. The task information and neural network structures are showed in Table 3. For example, on 'Acrobot': $|S| = 6$, $|A| = 1$, the neural network has two hidden layers with 5 and 3 neurons each, $|w| = 48$ and the maximum number of steps is 2,000. We will give a summary of each task and the details can be found in homepage of OpenAI Gym . In 'Acrobot', system includes two joints and two links, where the joint between the two links is actuated. Initially, the links are hanging downwards and the goal of this task is to swing the end of the low link up to a given height. In 'MountainCar', a car is positioned in a valley between two mountains and wants to drive up the mountain on the right by building up momentum. 'HalfCheetah', 'Humanoid', 'Swimmer', 'Ant' and 'Hopper' are simulation tasks. In those tasks, policy control simulated objects to achieve a goal. For example, in 'HalfCheetah', policy should control a cheetah with half body running forward as fast as possible. 'LunarLander' is a video game to control a lander to land on the surface of moon safely. The tasks of 'Acrobot' and 'MountainCar' are finding policies with smallest step number when goals are met. The tasks except for 'Acrobot' and 'MountainCar' are finding policies to control object getting score from environment as high as possible. Therefore, in Table2, rows of 'Acrobot' and 'MountainCar' are step numbers, the smaller the better. The other rows are scores from environment, the larger the better. All algorithms use at most 2,000 evaluations for each task.

We independently execute each method 15 times respectively on every task. And for each running, the result (policy) will be tested independently for 10 times to calculate the average cumulated reward (i.e., 20,000 total samples of trajectories in policy search). Finally, we pick up the policy with the best cumulated reward, and report the mean and standard deviation of the policy in Table 2. It can be observed that SRACOS obtained the best results on all of those tasks. Especially on complex tasks from HalfCheetah to LunarLander, SRACOS drastically improved the average reward.

Conclusion

In this paper, we propose the sequential classification-based derivative-free optimization method, SRACOS, for solving policy optimization problems in direct policy search, where solutions have to be sampled sequentially. We analyze the query complexity of SRACOS, and disclose the possibility that SRACOS can be more efficient than its batch-mode counterpart. In empirical analysis, we first study the properties of SRACOS on synthetic functions, showing its better convergence speed and stronger scalability than the other state-of-the-art derivative-free methods. On reinforcement learning tasks, SRACOS demonstrates significantly better performance than the other compared methods, showing that SRACOS is suitable for direct policy search with sequential sampling. Future work include designing better direct policy search methods using the power of SRACOS, and designing policy search methods for robust policies with more practical goals, such as high reward with small variance.

References

Bull, A. D. 2011. Convergence rates of efficient global optimization algorithms. *The Journal of Machine Learning Research* 12:2879–2904.

de Boer, P.; Kroese, D. P.; Mannor, S.; and Rubinstein, R. Y. 2005. A tutorial on the cross-entropy method. *Annals of Operations Research* 134(1):19–67.

de Freitas, N.; Smola, A. J.; ; and Zoghi, M. 2012. Exponential regret bounds for gaussian process bandits with deterministic observations. In *Proceedings of the 29th International Conference on Machine Learning*.

El-Fakdi, A.; Carreras, M.; and Palomeras, N. 2005. Direct policy search reinforcement learning for robot control. In *Conference on Artificial Intelligence Research and Development*, 9–16.

Golberg, D. E. 1989. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, Massachusetts: Addison-Wesley.

Hansen, N.; Müller, S. D.; and Koumoutsakos, P. 2003. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es). *Evolutionary Computation* 11(1):1–18.

Kawaguchi, K.; Kaelbling, L.-P.; and Lozano-Pérez, T. 2015. Bayesian optimization with exponential convergence. In *Advances in Neural Information Processing Systems 28*, 2791–2799.

Kim, H.; Jordan, M.; Sastry, S.; and Ng, A. 2003. Autonomous helicopter flight via reinforcement learning. *Advances in neural information processing systems*.

Larranaga, P., and Lozano, J. 2002. *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*. Boston: Massachusetts: Kluwer.

Munos, R. 2011. Optimistic optimization of a deterministic function without the knowledge of its smoothness. In *Advances in Neural Information Processing Systems*, 78–791.

Munos, R. 2014. From bandits to monte-carlo tree search: The optimistic principle applied to optimization and planning. *Foundations and Trends in Machine Learning* 7(1):1–130.

Neumann, F., and Wegener, I. 2007. Randomized local search, evolutionary algorithms, and the minimum spanning tree problem. *Theoretical Computer Science* 378(1):32–40.

Qian, H., and Yu, Y. 2016. On sampling-and-classification optimization in discrete domains. In *Proceedings of the 2016 IEEE Congress on Evolutionary Computation*.

Qian, H.; Hu, Y.-Q.; and Yu, Y. 2016. Derivative-free optimization of high-dimensional non-convex functions by sequential random embeddings. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence*.

Qian, C.; Yu, Y.; and Zhou, Z.-H. 2015a. Pareto ensemble pruning. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence*, 2935–2941.

Qian, C.; Yu, Y.; and Zhou, Z.-H. 2015b. Subset selection by pareto optimization. In *Advances in Neural Information Processing Systems*, 2935–2941.

Rogier, K., and Whiteson, S. 2011. Neuroevolutionary reinforcement learning for generalized control of simulated helicopters. *Evolutionary intelligence* 4(4):219–241.

Storn, R., and Price, K. 1997. Differential evolutiona simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization* 11(4):341–359.

Wang, X.-N.; Chen, W.; Zhang, M.; Xin, X.-U.; and HanGen, H.-E. 2007. A survey of direct policy search methods in reinforcement learning. *Caai Transactions on Intelligent Systems*.

Yu, Y., and Qian, H. 2014. The sampling-and-learning framework: A statistical view of evolutionary algorithm. In *Proceedings of the 2014 IEEE Congress on Evolutionary Computation*, 149–158.

Yu, Y.; Qian, H.; and Hu, Y.-Q. 2016. Derivative-free optimization via classification. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence*.

Yu, Y.; Yao, X.; and Zhou, Z.-H. 2012. On the approximation ability of evolutionary optimization with application to minimum set cover. *Artificial Intelligence* 180-181:20–33.