Front. Comput. Sci., 2021, 15(6): 156336
https://doi.org/10.1007/s11704-020-0241-4

REVIEW ARTICLE

Derivative-free reinforcement learning: a review

Hong QIAN, Yang YU (✉)

National Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China

Abstract Reinforcement learning is about learning agent models that make the best sequential decisions in unknown environments. In an unknown environment, the agent needs to explore the environment while exploiting the collected information, which usually forms a sophisticated problem to solve. Derivative-free optimization, meanwhile, is capable of solving sophisticated problems. It commonly uses a sampling-and-updating framework to iteratively improve the solution, where exploration and exploitation are also needed to be well balanced. Therefore, derivative-free optimization deals with a similar core issue as reinforcement learning, and has been introduced in reinforcement learning approaches, under the names of *learning classifier systems* and *neuroevolution/evolutionary reinforcement learning*. Although such methods have been developed for decades, recently, derivative-free reinforcement learning exhibits attracting increasing attention. However, recent survey on this topic is still lacking. In this article, we summarize methods of derivative-free reinforcement learning to date, and organize the methods in aspects including parameter updating, model selection, exploration, and parallel/distributed methods. Moreover, we discuss some current limitations and possible future directions, hoping that this article could bring more attentions to this topic and serve as a catalyst for developing novel and efficient approaches.

Keywords reinforcement learning, derivative-free optimization, neuroevolution reinforcement learning, neural architecture search

1 Introduction

Reinforcement learning [1,2] aims to enable agents to automatically learn the policy with the maximum long-term reward via interactions with environments. It has been listed as one of the four research directions of machine learning by Professor T. G. Dietterich [3]. In recent years, with the fusion of deep learning and reinforcement learning, deep reinforcement learning has made remarkable progress and attracted more and more attention from both the academic and industrial community. To name a few, the deep Q-network (DQN) [4] proposed by DeepMind reaches the human-level control in Atari games, AlphaGo [5] proposed also by DeepMind defeats the top human experts in the Go game, and AlphaZero [6] defeats a world champion program in the games of chess, shogi and Go without the domain knowledge except the game rules. Due to the strong ability of reinforcement learning, it has been applied in automatic control [7], automatic machine learning [8], computer vision [9], natural language processing [10], scheduling [11], finance [12], commodity search [13], and network communication [14], etc. In the field of cognition and neuroscience, reinforcement learning also has important research value [15, 16].

However, as reinforcement learning is being applied to more realistic problems, the complexity of finding out an optimal or satisfactory policy is also increasing. The optimization problems encountered in reinforcement learning, especially deep reinforcement learning, are often quite sophisticated. For such optimization problems, standard gradient-based optimization methods may suffer from the difficulties of stationary point issues (e.g., a plethora of saddle points or spurious local optima), bad condition number, or flatness in the activations that could lead to the gradient vanishing problem [17]. These difficulties cannot be neglected since they could result in the unsatisfactory performance of gradient-based optimization methods. Therefore, more effective policy learning methods that could make up for the shortcomings of gradient-based ones are quite appealing.

Derivative-free optimization [18–20], also termed as zeroth-order or black-box optimization, involves a kind of optimization algorithms that do not rely on the gradient information. Given a function f defined over a continuous, discrete, or mixed search space $\mathcal{X}$, it only relies on the objective function value (or fitness value) $f(x)$ on the sampled solution x. Since the conditions of using derivative-free algorithms are relaxed, they are easy to use and suitable for dealing with the sophisticated optimization tasks, e.g., non-convex and non-differentiable objective functions. Moreover, derivative-free optimization commonly uses a sampling-and-updating framework to iteratively improve the quality of solutions. One of the key issues is to balance the exploration and exploitation in the search space. This key issue is in a quite similar situation of reinforcement learning. Therefore, the fusion of derivative-free optimization and reinforcement learning, termed as *derivative-free reinforcement learning* in this paper, has many potentialities.

Derivative-free reinforcement learning has been developed for decades, under the names of *learning classifier systems* (e.g., [21]) and *neuroevolution/evolutionary reinforcement learning* (e.g., [22, 23]). Although it has some history, we have

Received May 31, 2020; accepted November 20, 2020

E-mail: yuy@nju.edu.cn

noticed that derivative-free reinforcement learning attracts increasing attentions recently. This article reviews some recent advances of derivative-free reinforcement learning in optimization, exploration and computation. In optimization, the article organizes the aspects into parameter updating and model selection. In exploration, the article presents the recently proposed derivative-free exploration methods in reinforcement learning. In computation, the article reviews the parallel and distributed derivative-free reinforcement learning approaches. We also discuss some limitations and potential future directions of derivative-free reinforcement learning.

The rest of this article is organized as follows. In Section 2, the article presents the background and organizes the aspects into introducing the basic concepts of reinforcement learning as well as derivative-free optimization. In Section 3, the article presents the relationship between reinforcement learning and derivative-free optimization from the aspects of optimization, exploration, and computation. This section also discusses why they should be considered together and how to combine them. The recent progress in derivative-free reinforcement learning from the aspect of optimization is reviewed in Sections 4 and 5. Specifically, in Section 4, the derivative-free model parameter updating in reinforcement learning is reviewed according to the different optimization methods. And in Section 5, the article presents the derivative-free model selection in reinforcement learning. In Section 6, the derivative-free exploration in reinforcement learning is reviewed. In Section 7, the article presents the parallel and distributed derivative-free reinforcement learning. At last, in Section 8, we conclude the paper, and discuss some current limitations and possible future concerns of this direction.

2 Background

This section introduces the background of reinforcement learning and derivative-free optimization. The fusion of them could be an effective way of tackling the hard policy search problems in reinforcement learning.

2.1 Reinforcement learning

Reinforcement learning (RL) [1, 2] is an important direction in machine learning [3]. It aims to enable an agent to automatically learn the policy with the maximum long-term reward via interactions with an environment. An illustration of interaction structure of reinforcement learning is shown in Fig. 1. When an agent is put in an unknown environment, it is told of the action space A in which there are actions it can choose to take, and the state space S in which its observation is contained in. Both S and A can be discrete or continuous. The agent has a policy π to determine which action $a \in A$ is chosen at a state $s \in S$. Generally, a deterministic policy π is a mapping from the state space S to the action space A, i.e., $a = \pi(s)$; while a stochastic policy π chooses the action a according to the probability distribution $\pi(a|s)$ at the state s with the constraints

$$\forall a \in A,\ \pi(a|s) \geqslant 0 \quad \text{and} \quad \sum_{a \in A} \pi(a|s) = 1. \tag{1}$$

When the agent takes one action a_t at a state s_t in time step t, the unknown environment typically responds by transiting to the next state s_{t+1} and feeds back a reward signal. The (unknown)

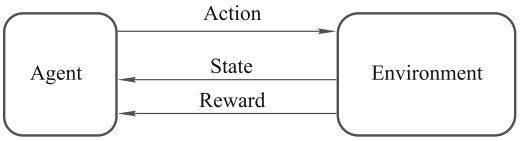

Fig. 1 Interaction structure of reinforcement learning

transition function can be represented as a distribution $P(s_{t+1}|s_t, a_t)$, and the (unknown) reward function can be represented as $r_t = R(s_t, a_t)$.

The agent explores the environment according to a policy π by interactions with the environment. Starting from the initial state $s_0 \sim \rho_0(\cdot)$, where $\rho_0(\cdot)$ is the start state distribution, the interaction trajectory τ (also frequently termed as episode or roll-out) is

$$\begin{aligned} &s_0, a_0 \sim \pi(\cdot|s_0), s_1 \sim P(\cdot|s_0, a_0), r_0 = R(s_0, a_0), \\ &a_1 \sim \pi(\cdot|s_1), \ldots. \end{aligned} \tag{2}$$

The quality of the policy π is then evaluated from interaction trajectories, as the expected sum up of the reward along the trajectories. Commonly, the expected total reward, or expected return, of the policy π starting from the state s is the expectation over the policy distribution and the transition distribution, e.g.,

$$V^{\pi}(s) = \mathbb{E}[\sum_{t=0}^{+\infty} \gamma^t r_t | s_0 = s], \tag{3}$$

where $\gamma \in [0, 1)$ is the discount factor.

Note that reinforcement learning setting shares the key components $< A, S, P, R, \gamma >$ with Markov decision process (MDP) [24], except that all the functions are known under the definition of an MDP. Also note that dynamic programming is a classical and effective method to solve the best policy in MDPs. There are thus two branches of approaches, model-based ones that recover the transition distribution and reward function to form up a complete MDP for using classical solvers, and model-free ones that learn the policy without recovering the MDP. In this paper, we mainly focus on model-free approaches. Nevertheless, model-assisted model-free reinforcement learning is a promising direction.

In model-free reinforcement learning, the policy can be derived from well learned value functions, or can be learned directly. In either ways, the general steps of the learning approaches are the same, i.e., iterating between exploring the environment and updating the policy. The exploration is typically implemented by executing the policy with added noise, such as ϵ-greedy and Gibbs sampling. In this way, the probability of executing every action and visiting every state is non-zero, and thus diverse trajectory data can be generated for the next step of policy update.

To learn the value function, the most popular way might be the temporal difference update [1], which is essentially an incremental update rule of arithmetic sum

$$V^{\pi}(s_t) \leftarrow V^{\pi}(s_t) + \delta \cdot (r_t + V^{\pi}(s_{t+1}) - V^{\pi}(s_t)), \tag{4}$$

where $\delta > 0$ denotes the step size. After obtaining the value function, the policy is derived simply as that taking the action of the largest value, i.e., $\pi(s) = \text{argmax}_a Q^{\pi}(s, a)$, where

$Q^\pi(s,a) = R(s,a) + \mathbb{E}_{s'\sim P(\cdot|s,a)}V^\pi(s')$. $V^\pi(s)$ and $Q^\pi(s,a)$ are called value functions.

The value function based methods may face the problem of policy degradation [25]. That is to say, a more accurate estimation of the value function may not guarantee a better policy. Another kind of approach is to learn the policy directly, i.e., policy search. In policy search, the policy model is firstly parameterized, such as softmax policy for discrete action space

$$\pi_\theta(a|s) = \frac{\exp\left(h_\theta(s,a)\right)}{\sum_{a'\in A}\exp\left(h_\theta(s,a')\right)}, \tag{5}$$

and Gaussian policy for continuous action space

$$\pi_\theta(a|s) = \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{(a-h_\theta(s))^2}{\sigma^2}\right), \tag{6}$$

where $h_\theta(s,a)$ in Eq. (5) and $h_\theta(s)$ in Eq. (6) with parameter θ can be linear, and are nowadays often (deep) neural network models. Note that this parametric policy model is fully differentiable with respect to (w.r.t.) θ, if h_θ is differentiable. Then, consider the objective of learning an optimal policy $\pi^* = \pi_{\theta^*}$ that can maximize the total reward $J(\theta)$, i.e.,

$$\theta^* = \mathrm{argmax}_{\theta\in\Theta}J(\theta), \tag{7}$$

where Θ denotes the parameter space. We suppose that both the environment transitions and the policy are stochastic. In episodic environments, $J(\theta)$ can be trajectory-wise total reward

$$J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[R(\tau)] = \int_{\mathcal{T}} p_\theta(\tau)R(\tau)\,\mathrm{d}\tau, \tag{8}$$

where $R(\tau) = \sum_{t=0}^{T-1} r_t$ is the total reward (or return) over a trajectory τ, $\mathcal{T}$ is a valid trajectory space, and $p_\theta(\tau) = \rho_0(s_0)\prod_{t=0}^{T-1}P(s_{t+1}|s_t,a_t)\pi_\theta(a_t|s_t)$ is the probability of generating a T-step trajectory τ according to π_θ in the environment. In continuing environments, $J(\theta)$ can be average reward per time-step for one-step MDPs

$$J(\theta) = \int_S d^{\pi_\theta}(s)\int_A \pi_\theta(a|s)R(s,a)\,\mathrm{d}a\,\mathrm{d}s, \tag{9}$$

where $d^{\pi_\theta}(s)$ is the stationary probability of visiting the state s following the policy π_θ. We can find that the total reward objective $J(\theta)$ is also differentiable w.r.t. θ. A straightforward idea of solving the parameter is to follow the gradient $\nabla_\theta J(\theta)$, which is generally called as policy gradient method.

Affected by the deep learning, flexible and capable deep neural network models are introduced in reinforcement learning. It can be noticed that, although the objective $J(\theta)$ is differentiable, the objective function is not simple, particularly when deep neural network models are employed. How to best optimize the objective function is still an open problem with continual progress.

2.2 Derivative-free optimization

Optimization, $x^* = \mathrm{argmax}_{x\in\mathcal{X}}f(x)$ as a general representative, plays a fundamental role in machine learning. With the rapid development of machine learning, it has deeper applications in a broader and more complex scenario. Due to the increasing complexity of machine learning application problems, optimization methods for complex and hard problems are attracting more and more attention from researchers. For complex optimization problems (e.g., non-convex, non-smooth, non-differential, discontinuous, and NP-hard, etc.), gradient-based methods may fall into the local optima or even lose their power, and result in the unsatisfactory performance.

In a search space, the objective of optimization is to find the solution with the extreme function value. A general principle of optimization is simply that, given the currently accessed solutions, which can be randomly sampled at the initialization, find the next solution with better function value. For example, gradient ascent method finds the next solution follows the gradient direction of the objective function. This procedure requires firstly that the objective function is differentiable, and more importantly that there are few local optima and saddle points, so that the gradient direction is informative and can lead to better solutions.

Derivative-free optimization (DFO) [18–20], also termed as zeroth-order or black-box optimization, finds the next solutions in another way. It covers many families of optimization algorithms that do not rely on the gradient. It only relies on the function values (or fitness values) $f(x)$ on the sampled solution x. Most derivative-free optimization algorithms share a common structure. They firstly initialize from some random solutions in the search space. From the accessed solutions, derivative-free optimization algorithms build a model, either explicitly or implicitly, about the underlying objective function. The model could imply an area that contains some potential better solutions. They then sample new solutions from that model and update the model. Derivative-free optimization methods repeat this sampling-and-updating procedure to iteratively improve the quality of solutions. To sum up, the general framework of derivative-free optimization methods involve some key steps below:

1) Randomly sample solutions;
2) Evaluate the objective function value of the sampled solutions;
3) Update the model from the sampled solutions and their function values;
4) Sample new solutions according to the model with a designed mechanism;
5) Repeat from step 2 until some termination conditions are satisfied;
6) Return the best found solution and its function value.

The termination conditions usually include that the function evaluation budget is exhausted or the goal optimal function value has been reached. Representative derivative-free algorithms include evolutionary algorithms [26, 27], Bayesian optimization [28], cross-entropy method [29], deterministic or stochastic optimistic optimization [30], and classification-based optimization [31], etc. So far, the substantial progress has been made in both the theoretical analysis tools and theoretical guarantees of derivative-free optimization [30–40]. Some fundamental and crucial concerns in theory, such as the global con-

vergence rate and the function class that can be optimized efficiently, are disclosed progressively.

In order to take a deeper insight of the key steps 3 and 4 in the above procedure, we present a snapshot of a canonical genetic algorithm [26,41] as an example. Genetic algorithms belong to the family of evolutionary algorithms. Consider maximizing a pseudo-Boolean function $\text{argmax}_{x\in\{0,1\}^d} f(x)$. Here the solutions are represented as bit strings of length d. A canonical genetic algorithm maintains a population which includes n solutions, and the model in each iteration/generation is represented by the population (i.e., n solutions in each population). In each iteration, n children (or offspring) are produced on the basis of n parent solutions via the designed mechanism called crossover operator from the parents, e.g., single-point crossover which is shown below. We set $d = 5$ for illustration.

$$\begin{aligned} x_1^{\text{parent}} &= (0, \underline{1}, 0, 0, 1) \\ x_2^{\text{parent}} &= (1, \underline{0}, 1, 0, 0) \end{aligned} \xrightarrow{\text{crossover}} \begin{aligned} x_1^{\text{child}} &= (0, \underline{1}, 1, 0, 0), \\ x_2^{\text{child}} &= (1, \underline{0}, 0, 0, 1). \end{aligned} \tag{10}$$

In the above single-point crossover, a point (or bit) on both parents' bit strings is picked randomly, and bits to the right of that point are swapped between the two parents, which results in two children. Besides, mutation is another variation operator in the designed mechanism. If mutation takes place, it can be applied to each child that has been produced by crossover. One way of realizing mutation could be flipping each bit in a solution x^{child} with some probability and producing a new solution $x_{\text{new}}^{\text{child}}$, which is shown below. We set $d = 5$ for illustration.

$$x^{\text{child}} = (0, 1, 1, \underline{0}, 0) \xrightarrow{\text{mutation}} x_{\text{new}}^{\text{child}} = (0, 1, 1, \underline{1}, 0). \tag{11}$$

After the crossover and mutation operators (step 4), the objective function values of these newly produced solutions are evaluated (step 2). Then, the model, which is represented by the solutions, is updated via selecting the best n solutions from both the parents and children to form the next generation of population (step 3).

From the above algorithmic procedure, some key characteristics of derivative-free optimization algorithms can be observed. Firstly, they can be utilized as long as the quality of solutions can be evaluated. Secondly, the designed mechanisms for sampling solutions and rules for updating model always consider the balance between exploration and exploitation. In optimization, exploration intuitively means gathering more information about objective functions and reducing some uncertainty, while exploitation means choosing the best solutions under current information. In the instance of canonical genetic algorithm, exploration is realized via crossover and mutation operators. The combination of exploration and exploitation could help optimization procedures maintain global and local search, and jump out of the local optima in order to find out the (approximately) global optima with high probability. Thirdly, many derivative-free optimization algorithms are population-based. A population of solutions is maintained and improved iteratively, and the algorithms could share and leverage the information across a population. These key characteristics make derivative-free optimization methods have a low barrier to use as well as the ability of search globally.

Since the conditions of using derivative-free optimization methods are relaxed, they are easy to use and general in the continuous, discrete, or mixed search space. Their ability could guarantee the effectiveness of global optimization. Therefore, derivative-free optimization methods are suitable for dealing with the complex and hard optimization problems. They have been applied in the complex learning tasks and achieved impressive empirical results, such as policy search in reinforcement learning [42–45], automatic machine learning and hyper-parameter tuning [46–49], objective detection in computer vision [50], subset selection [51, 52], and security games [53], etc.

3 When RL meets DFO

After a brief recap of reinforcement learning (RL) and derivative-free optimization (DFO), this section explains and emphasizes that the fusion of them, termed as *derivative-free reinforcement learning*, is necessary and attractive. We explain it from the aspects of optimization, exploration, and computation, respectively.

Optimization A learning task typically involves three components, and they are representation, evaluation, and optimization [54]. The ability of optimization method has a significant impact on the complexity of model representation and the type of evaluation function that we can choose for learning. Nowadays, in reinforcement learning, the policy or value function models are often represented by deep neural networks, i.e., deep reinforcement learning. When injecting this sophisticated deep representation into the evaluation function in RL (e.g., Eq. (8) or Eq. (9) in Section 2.1), the resulting optimization problems (e.g., Eq. (7) in Section 2.1) could be non-convex.

For such optimization problems, standard gradient-based methods may suffer from the difficulties of stationary point issues (e.g., a plethora of saddle points or spurious local optima), bad condition number, or flatness in the activations that could lead to the gradient vanishing problem [17]. And these difficulties could result in the unsatisfactory results.

At the same time, derivative-free methods that conduct optimization from samples provide another way of policy learning, and can be complementary with gradient-based ones in reinforcement learning. One straightforward way of applying derivative-free optimization methods is to define the search space as the policy parameter space and the objective function as the expected long-term reward. Namely, $\mathcal{X} \overset{\text{def}}{=} \Theta$ and $f(x) \overset{\text{def}}{=} J(\theta)$. For policy learning, derivative-free optimization methods have their own merits of being able to search parameters globally and being easy to train. They do not perform gradient back-propagation, do not care whether rewards are sparse or dense, do not care the length of time horizons, and do not need value function approximation [45]. In Section 4 and 5, we will discuss the recent advances dedicated to the derivative-free model parameter updating as well as model selection in reinforcement learning, respectively.

Exploration Almost all reinforcement learning methods share the exploration-learning framework [55]. Namely, an agent explores and interacts with an unknown environment to learn the optimal policy that maximizes the total reward from the exploration samples. Generally speaking, the explo-

ration samples involve states, state transitions, actions and rewards. From the exploration samples, the quality of a policy can be evaluated by rewards, and the learning step updates the policy or value function models from the evaluations. This exploration-learning procedure is repeated until some termination conditions are met. Exploration is necessary in reinforcement learning. Because achieving the best total reward on the current trajectory samples is not the ultimate goal, and the agent should visit the states that have not been visited before so as to collect better trajectory samples. This means that the agent should not follow the current policy tightly, and thus the exploration mechanisms need to encourage the agent to deviate from the previous trajectory paths properly.

Most existing exploration mechanism mainly suffer from being memoryless and blind, e.g., action space noise or parameter space noise [56], or being difficult to use in real state/action spaces, e.g., curiosity-driven exploration [57]. On the other hand, many mainstream policy gradient methods, such as truncated natural policy gradient (TNPG) [58] and trust region policy optimization (TRPO) [59], seldom touch the exploration.

Meanwhile, derivative-free reinforcement learning is naturally equipped with the exploration strategies. Because in the search process of derivative-free optimization methods, the designed mechanisms for sampling solutions and rules for updating model always consider the exploration. Therefore, derivative-free optimization methods can take part of the duty of exploration for reinforcement learning when updating the policy or value function models from samples. Recently, some problem-dependent derivative-free exploration methods that could improve the sample efficiency have been proposed. In Section 6, we will discuss these works dedicated to the derivative-free exploration in reinforcement learning.

Computation Although derivative-free methods could bring some good news to reinforcement learning with respect to optimization and exploration, they mostly suffer from low convergence rate. Derivative-free optimization methods often require to sample a large amount of solutions before convergence, even if the objective function is convex or smooth [35, 37, 60]. And the issue of slow convergence becomes more serious as the dimensionality of a search space increases [37, 61]. Obviously, this issue will block the further application of derivative-free methods to reinforcement learning. They sample a lot of policy parameters before finding out an optimal or satisfactory one, and the quality of policy parameters is evaluated by the trajectory samples. This makes reinforcement learning more sample inefficient.

Fortunately, many derivative-free optimization methods are population-based. That is to say, a population of solutions is maintained and improved iteratively. This characteristic makes them highly parallel. Thus, derivative-free optimization methods can be accelerated by parallel or distributed computation, which alleviates their slow convergence. Furthermore, for parallel/distributed derivative-free methods, the data communication cost is lower compared with gradient-based ones, since only scalars (fitness values) instead of gradient vectors or Hessian matrices need to be conveyed. This merit further compensates for the low convergence rate partly. In Section 7, we will discuss the recent works dedicated to the parallel/distributed derivative-free reinforcement learning.

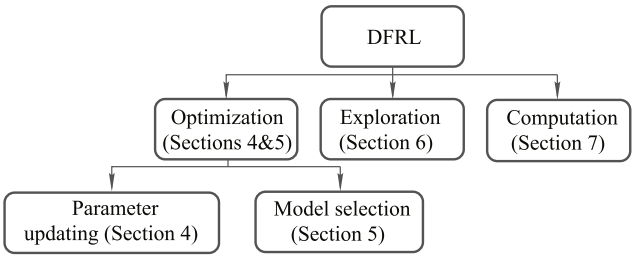

Fig. 2 The organization of the works reviewed in the article

To sum up, derivative-free reinforcement learning could hopefully result in more effective and powerful algorithms for complex control tasks, more sample efficient exploration in environments, and more time efficient global optimization for better policy. The organization of the following reviewed works is depicted in Fig. 2. We should stress that derivative-free optimization methods are not proposed to replace gradient-based ones, e.g., policy gradient algorithms, in reinforcement learning, and they are complementary with each other. In fact, some impressive works that will be discussed in this article are hybrids of derivative-free and gradient-based ones.

4 Derivative-free model parameter updating in reinforcement learning

The generic framework of using derivative-free optimization algorithms to update the model parameters in reinforcement learning is quite straightforward. After parameterizing the policy or value function models, the quality of a parameter θ is evaluated via the total reward $J(\theta)$ provided by an environment. For policy search methods in deep reinforcement learning, the parameters θ of a policy model π_θ are the weights of a deep neural network. Here the policy model can be softmax policy for discrete action space or Gaussian policy for continuous action space, and this section only considers that the architecture/topology of a (deep) neural network is fixed. Derivative-free algorithms regard θ and $J(\theta)$ as solutions and objective function values (or fitness values) respectively, and search the optimal solution θ^*. They sample different policy parameters, and learn where to sample in the next iteration.

The works dedicated to applying derivative-free methods to optimize the weights of neural networks have been developed for decades, e.g., neuroevolution. In neuroevolution, evolutionary algorithms, inspired from natural evolution, can not only optimize the weights, but also search the topology of neural networks (discussed in Section 5). In each generation of a neuroevolutionary algorithm, each neural network in the population is evaluated by the task, and the best ones are selected. The crossover and mutation operators are used to reproduce the new networks from the selected ones in order to form a new population, and the process iterates. A comprehensive discussion of neuroevolution can be found from the last century review paper [62] or the recent one [63].

The methods of applying neuroevolution for reinforcement learning tasks have been developed for decades. Recently, more and more works have shown that, compared with gradient-based methods, neuroevolution is competitive for not only policy search in RL [45, 64], but also supervised learning tasks

[65,66]. This confirms the power of neuroevolution and renews the increasing interests in it. In addition to the success of neuroevolution RL [45, 67–70], other derivative-free optimization methods for RL could also be promising. For instance, stochastic zeroth-order search could rival the gradient-based methods for the static linear policy optimization on the MuJoCo locomotion tasks [71]. And its convergence behavior is analyzed in [72].

In this section, we mainly review the recent progress in derivative-free model parameter updating in reinforcement learning, and the involved works include but are not only restricted to neuroevolution RL. For the early works on evolutionary RL, the survey of them can be found in [22, 23].

4.1 Evolution strategies based model parameter updating

Evolution strategies (ESs) [73] belong to the family of evolutionary algorithms inspired from natural evolution. In ESs, a parameterized search distribution (e.g., Gaussian or Cauchy distribution) is maintained, and solutions are sampled from this search distribution and then are evaluated by an objective function. The search distribution is iteratively updated according to the evaluated solutions. Mutation is a commonly used variation operator in ESs, and it perturbs solutions in order to generate the new ones. Mutation introduces the diversity of solutions in the population, and may alleviate the problem that optimization algorithms are trapped into the local optima. Due to the different algorithmic implementations, ESs have many variants, and one of the most popular among them might be the covariance matrix adaptation evolution strategy (CMA-ES) [27, 74, 75]. In CMA-ES, the search distribution is a multivariate Gaussian, and its covariance matrix adapts over time. The mutation ellipsoids of CMA-ES are not restricted to be axis-parallel, and the overall step size is controlled with the cumulative step size adaptation.

Heidrich-Meisner and Igel [76, 77] proposed to optimize the parameterized policy in RL via CMA-ES. They consider to apply CMA-ES because it is robust (ranking policies based), can detect the correlations among parameters, and can infer the search direction from the scalar signals in RL. In [76], CMA-ES is used to optimize linear policies (e.g., $\pi_\theta(s) = \theta^\top s$ for the deterministic policies) on the double cart-pole balancing task. Empirical results show that, compared with the episodic natural actor-critic algorithm (NAC) [78] which is a gradient-based one, CMA-ES is more robust with respect to noise, policy initialization, and hyper-parameters. On the other hand, NAC could surpass CMA-ES with respect to learning speed under appropriate policy initialization and hyper-parameters. In [77], CMA-ES is used to search neural network policies on the Markovian and non-Markovian variants of the pole balancing task. Compared with the policy gradient methods, value function based methods, random search, and several neuroevolution methods, overall, the empirical performance of CMA-ES is superior.

Also, Heidrich-Meisner and Igel [79] enhanced CMA-ES for direct parameterized policy search with an adaptive uncertainty handling mechanism. On the basis of this mechanism, the resulting Race-CMA-ES can dynamically adapt the number of roll-outs for evaluating the parameterized policies such that the ranking of solutions is just reliable enough to push the learning process. Empirical results on the mountain car and swimmer control tasks with linear policies show that, compared with CMA-ES and NAC [78], Race-CMA-ES can accelerate the learning process and improve the algorithmic robustness. Stulp and Sigaud [80] were the first to make the relationship between CMA-ES and the cross-entropy (CE) method [29] explicit under the view of probability-weighted averaging. The efficacy of CE for playing Tetris has been studied in [81]. They claim that CE can be regarded as a special case of CMA-ES through setting some CMA-ES parameters to extreme values. Besides, they also inject the covariance matrix adaptation into the policy improvement with path integrals, and result in PI^2-CMA. PI^2-CMA shares the similar way of performing the parameter updating with CMA-ES and CE, but has the more consistent convergence behavior under varying initial conditions.

Wierstra et al. [82, 83] presented the natural evolution strategies (NESs) where the natural gradient was used to update a parameterized search distribution in the direction of higher expected fitness. The effectiveness of NESs is empirically verified on a neuroevolutionary control policy design for the non-Markovian double pole balancing task. And the result shows that NESs possess the merits of alleviating oscillations and premature convergence.

Salimans et al. [45] from OpenAI proposed to use a simple evolution strategy, based on a simplification of NESs, to directly optimize the weights θ of policy neural networks. After initializing the policy parameters θ_0, OpenAI ES stochastically mutates the parameters θ of the policy with Gaussian distribution, and evaluates the resulting mutated parameters via the total reward/return $J(\cdot)$. Then, it combines these returns and updates the parameters θ. This process iterates till the termination condition is met. Let $\alpha > 0$ denote the learning rate, and σ denote the noise standard deviation. The core steps are summarized as follows:

1) Randomly sample $\epsilon_1, \ldots, \epsilon_n$ from the standard Gaussian distribution $\mathcal{N}(0, I)$;
2) Compute returns $J_i = J(\theta_t + \sigma\epsilon_i)$ for $i = 1, \ldots, n$;
3) Update parameters $\theta_{t+1} \leftarrow \theta_t + \alpha \frac{1}{n\sigma} \sum_{i=1}^{n} J_i \epsilon_i$;
4) Repeat from step 1 till the termination condition is met.

Furthermore, some techniques are integrated to facilitate the success of OpenAI ES for learning policy neural networks. To name a few, virtual batch normalization [84] is adopted to enhance the exploration ability, antithetic sampling [85] (also known as mirrored sampling [86]) is adopted to reduce variance, and fitness shaping [83] is adopted to decrease the trend of trapping into the local optima in the early training phase. This simple and generic OpenAI ES is very easy to parallelize and only takes low communication cost, which will be discussed in Section 7. The effectiveness of OpenAI ES is empirically investigated on some simulated robotics tasks in MuJoCo and Atari games in OpenAI Gym [87–89]. And the results surprisingly show that OpenAI ES is comparable to some state-of-the-art gradient-based algorithms, e.g., trust region policy optimization (TRPO) [59] and asynchronous advantage actor-critic (A3C) [90], on both simple and hard environments, but Ope-

nAI ES needs more samples. Notably, this work [45] seems to renew the increasing interests in (deep) neuroevolution RL.

Encouraged and inspired by [45], new ideas, discussions and approaches are blooming out. Lehman et al. [91] claimed that the simple OpenAI ES in [45] was more than just a finite-difference approximation of the reward gradient. Because OpenAI ES optimizes the average return of the whole solutions in the population instead of a single solution, it searches parameters that are robust to perturbation in the parameter space and yields more stable policies. Chrabaszcz et al. [70] compared a simper and basic canonical ES algorithm with OpenAI ES [45]. The empirical results on a subset of 8 Atari games [88, 89] show that this simpler canonical ES is able to match or even surpass the performance of OpenAI ES. This work [70], on the one hand, further confirms that the power of ES-based model parameter updating for policy search may rival that of the gradient-based algorithms. On the other hand, it indicates that the ES-based RL algorithms could be further ameliorated in many aspects by integrating the recent advances made in the field of ES.

Choromanski et al. [92] enhanced ES-based RL model parameter updating by structured evolution and compact policy networks. The resulting algorithm needs less policy neural network parameters, and thus could speed up training and inference. Chen et al. [93] improved ES for training policy neural networks in terms of the mutation strength adaptation and principal search direction update rules. And the number of elitists adaptation and restart procedure are also integrated to tackle the local optima. The proposed algorithm is of linear time complexity and low space complexity, and thus possesses the merit of scalability. Choromanski et al. [94] applied the ideas of active subspaces [95], which is one of the popular approaches to dimensionality reduction, to yield the sample-efficient and scalable ES for policy optimization in RL. Liu et al. [96] boosted the sample efficiency of ES for RL by reusing the existing sampled data, and proposed the trust region evolution strategies (TRES) algorithm. TRES realizes sample reuse for multiple epochs of updates in the way of iteratively optimizing a surrogate objective function. Tang et al. [97] proposed a variance reduction technique for ES-based RL model parameter updating. It utilizes the underlying MDP structure of RL through problem re-parameterization and control variable construction. Fuks et al. [98] proposed a technique called progressive episode lengths (PEL), and injected it into a canonical ES [70] to result in PEL-ES. Inspired from transfer learning and curriculum learning, PEL-ES firstly lets an agent play the easy and short tasks, and then applies the gathered knowledge to further deal with the harder and longer tasks. Compared with the canonical ES [70], PEL-ES is superior in optimization speed, total rewards, and stability.

Houthooft et al. [99] proposed a meta-learning method for policy learning across different tasks called evolved policy gradient (EPG). EPG encodes the prior knowledge implicitly through a parametrized loss function, and parametrization is realized by temporal convolutions over the agent's experience. The agent can use the learned loss function to learn on a new task quickly. Thus, the main procedure is to evolve a parametrized and differentiable loss function. The agent optimizes its policy to minimize this loss function so as to gain high returns. To implement this procedure, EPG involves two optimization loops:

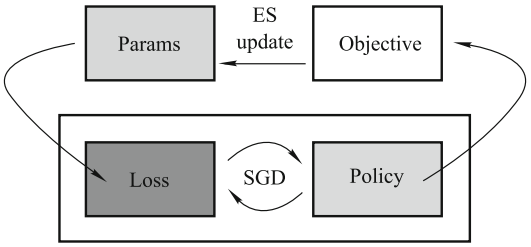

Fig. 3 Figure from [99] that illustrates the high-level procedure of EPG

- In the inner loop, the agent learns to solve a task through minimizing a loss function provided by the outer loop.
- In the outer loop, the parameters of a loss function are optimized in order to maximize the returns gained after the inner loop.

The inner loop is optimized by the stochastic gradient descent (SGD) method. For the outer loop, since the returns are not explicit functions of the loss function parameters, ES can be applied to optimize the parameters of this loss function. Figure 3 illustrates the main procedure of EPG. Empirical results on several continuous control tasks in MuJoCo [87, 89] show that, compared with proximal policy optimization (PPO) [100] that is an off-the-shelf policy gradient algorithm, EPG can generate a learned loss function which trains the agent faster. Besides, EPG also exhibits generalization properties for out-of-distribution test time tasks that surpass other meta-learning algorithms RL^2 [101] and MAML [102]. Notably, in EPG, ES is used to optimize the parameters of a loss function instead of policy parameters. And the success of EPG largely owes to the hybrid of derivative-free and gradient-based algorithms. The similar scenario also happens in [103, 104], where CMA-ES and gradient-based algorithms are mixed for policy transfer from simulation to reality.

4.2 Genetic algorithms based model parameter updating

Similar to evolution strategies (ESs), genetic algorithms (GAs) [41] also belong to the family of evolutionary algorithms inspired from natural evolution. Genetic algorithms maintain a population of solutions. Via the operators of mutation, crossover (also called recombination) and selection, a population of solutions can be evolved. One of the main differences between GAs and ESs is that the crossover operator is often used in GAs. Crossover combines two parents to generate new offspring, and the newly generated solutions are usually mutated before adding them to the population. Notably, the crossover operator in GAs could further improve the diversity of solutions in the population.

To verify whether GAs can also be effectively applied to the RL tasks, Such et al. [64] used a very simple GA to update the weights of deep policy neural networks. It iteratively maintains a population of parameter vectors θ, and the mutation operator is realized by the additive Gaussian noise to θ. The elitism technique is employed during evolution. For simplicity, it does not consider crossover. Due to the huge amount of parameters in

deep neural networks and the relatively unsatisfied scalability of GA, the work proposes an approach to storing large parameter vectors θ compactly via representing each θ as an initialization seed plus the list of random seeds. This compact representation approach is reversible (i.e., each parameter vector θ can be reconstructed from it), and substantially improves the scalability and efficiency of deep GA. The policy neural networks with more than four million free parameters can be successfully evolved. Besides, novelty search (NS) [105] is integrated into this deep GA to avoid the local optima and encourage exploration. And a distributed version of the deep GA on many CPUs across many machines is implemented for acceleration. The experimental results on the Atari and MuJoCo humanoid locomotion tasks [87–89] indicate that the deep GA is effective and competitive. Overall, it performs roughly as well as DQN [4], A3C [90], and OpenAI ES [45].

Gangwani and Peng [106] proposed a genetic policy optimization (GPO) method for sample-efficient deep policy optimization. GPO involves crossover, mutation, and selection. For crossover, instead of directly exchanging the parameter representations of two parents (parameter crossover) that may destroy the hierarchical relationship of the neural networks and lead to a catastrophic drop in performance, GPO employs imitation learning to realize policy crossover in the state space. The state-space crossover operator combines two parent policies π_x and π_y to produce an offspring (or child) policy π_c that shares the same network architecture as parents via two steps, as shown in Fig. 4. Firstly, it trains a two-level policy π_H, and

$$\begin{aligned}\pi_H(a|s) = \pi_S(\text{parent} = x|s)\,\pi_x(a|s)+\\ \pi_S(\text{parent} = y|s)\,\pi_y(a|s),\end{aligned} \tag{12}$$

where π_S is a binary policy that trains from the parents' trajectories. Given a state s, π_H selects between the parent policies π_x and π_y, and then outputs the action distribution of the selected parent. Secondly, the trajectories generated from the expert policy π_H is used as the supervised data, and the offspring policy π_c is trained from the supervised data by imitation learning. To avert the compounding errors introduced by the state distribution mismatch between expert and offspring, the dataset aggregation (DAgger) algorithm [107] is used for imitation learning. This sample-efficient crossover can distill the knowledge from parents to produce a child that aims to imitate its best parent in generating similar state visiting distributions. For mutation, instead of utilizing the commonly-used Gaussian perturbation, GPO mutates the policy network weights by randomly rolling out trajectories and performing several iterations of a policy gradient algorithm using these roll-out trajectories. GPO chooses PPO [100] and advantage actor-critic (A2C) algorithm [1, 90] to fulfill the policy gradient mutation. This mutation operator possesses the merits of high efficiency, sufficient genetic diversity, and good exploration of state space. For selection, GPO takes both performance and diversity into consideration. The experimental results on some continuous control tasks in MuJoCo [87, 89] show that, compared with PPO and A2C which are state-of-the-art policy gradient methods, GPO could be superior in terms of episode reward and sample efficiency.

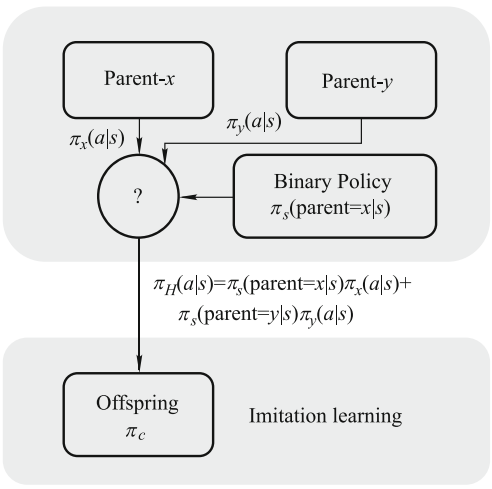

Fig. 4 An illustration of combining the parent policies to produce an offspring policy in GPO [106]

Bodnar et al. [108] also made progress in developing the variation operators in GAs tailored to policy optimization. They observe that, when the direct genetic encoding for deep neural networks (i.e., the weights of a network are recorded as a list of real numbers) meets the traditional variation operators in GAs, the negative side-effects may occur in RL since deep neural networks are sensitive to the small variation of weights. For instance, the recently proposed evolutionary reinforcement learning (ERL) framework [109], which combines both of them, has the risks of catastrophic forgetting and destructive behaviors, and does not fully address the scalability problem of GAs for RL. Therefore, they propose the learning-based Q-filtered distillation crossover, the safe mutation [110] based proximal mutation , and then integrate them into ERL [109] in a hierarchical manner to result in the proximal distilled evolutionary reinforcement learning (PDERL) framework. PDERL could compensate for the simplicity of the direct genetic encoding, satisfy the functional requirements of the genetic variation operators when applied to the directly encoded deep neural networks, and prevent the catastrophic forgetting of parental behaviors. Compared with PPO [100] and twin delayed deep deterministic policy gradients (TD3) algorithm [111], as well as ERL, PDERL is superior when evaluated on five robot locomotion tasks in OpenAI Gym [87–89]. Notably, in GPO and PDERL, the gradient-based update is injected into GAs to enhance the genetic variation operators. The success of them implies that, by fusing derivative-free and gradient-based algorithms in a clever way, the strengths of both sides could be absorbed, and more powerful policy search algorithms would be expected.

4.3 Bayesian optimization based model parameter updating

Bayesian optimization (BO) [28] is also a kind of widely-used black-box derivative-free approaches aimed at optimizing the difficult functions with relatively few evaluations. BO constructs a probabilistic model for the function being optimized, and then exploits this model to make decisions about the next solution point of the function to evaluate. The newly evaluated solution and the previously evaluated ones are all used to update the probabilistic model so as to improve its accuracy, and this procedure iterates to sample the solutions with increasing quality. There are two ingredients that need to be specified in

BO, i.e., the probabilistic prior over functions that expresses the assumptions of the function being optimized, and the acquisition function that determines the next solution to evaluate. For the probabilistic prior, BO usually uses the Gaussian process (GP) [112] prior because of its flexibility and analytical tractability. GP defines a distribution over functions specified by its mean function and covariance function. The function being optimized is assumed to be drawn from a GP prior, and this prior as well as the sampled data induce a posterior over functions. The acquisition function $a(\cdot)$, that is constructed from the model posterior, determines which solution should be evaluated next via a proxy optimization $\theta_{\text{next}} = \text{argmax}_\theta a(\theta)$. The popular choices of acquisition function are probability of improvement [113], expected improvement [114], GP upper confidence bound [115, 116] and so on. As BO can exploit the prior information about the expected return and utilize this knowledge to select new policies to execute, more and more works focus on applying BO to RL, and early works on this direction is already reviewed in [117].

Wilson et al. [118] proposed a novel Gaussian process covariance function to measure the similarity between policies using the trajectory data generated from policy executions. Compared with the traditional covariance functions that relate the policy parameters, the proposed covariance function that relates the policy behavior is more reasonable and effective. Furthermore, they also introduce a novel Gaussian process mean function that exploits the learned transition and reward functions to approximate the landscape of the expected return. The developed Bayesian optimization approach tailored to reinforcement learning could recover from model inaccuracies when good transition and reward models cannot be learned. Empirical results on a set of classic control benchmarks verify its effectiveness and efficiency.

Calandra et al. [119] applied Bayesian optimization to design the gaits and the corresponding control policies of a bipedal robot. Three popular acquisition functions, as well as the effect of fixed versus automatic hyper-parameter selection, are analyzed therein. In [120], they additionally formalize the problem of automatic gait optimization, discuss some widely-used optimization methods, and extensively evaluate Bayesian optimization on both simulated tasks and real robots. The evaluation demonstrates that BO is very suitable for robotic applications since it could search a good set of gait parameters with only a few amount of experimental trials. By comparing the different variants of BO algorithms, they observe that the GP upper confidence bound acquisition function performs the best among them.

Marco et al. [121] proposed a Bayesian optimization algorithm that can adaptively select among multiple information sources with different accuracies and evaluation costs, e.g., experiments on a real-world robot and simulators. This BO algorithm exploits the prior knowledge from simulations via maximizing the information gain from each experiment, and automatically integrates the cheap but inaccurate information from simulations with the accurate but expensive real-world experiments in a cost-effective way. The empirical results show that using the prior model information from simulators can reduce the amount of data required to find out the desirable control policies. Letham and Bakshy [122] augmented the on-line experiments with the off-line simulator observations to tune the live machine learning systems on the basis of the multi-task Bayesian optimization [123]. The empirical study on the live machine learning systems indicates that, by directly utilizing a simple and biased off-line simulator together with a small number of on-line experiments, the proposed method can accurately predict the on-line outcomes and achieve the substantial gains. And the empirical result is consistent with the theoretical findings of the multi-task Gaussian process generalization.

The work termed as the Bayesian functional optimization (BFO) [124] extends the BO methods to the functional policy representations, and its motivation is similar to [125]. BFO models the function space as a reproducing kernel Hilbert space (RKHS) [112], and introduces an efficient update of the functional GP as well as a simple optimization of the acquisition functional. BFO bypasses the problem of manually selecting the features used in the function approximations to define the parameter space, and thus relaxes the performance reliance on the selected parameter space. The experimental result on the RL task whose policies are modeled in RKHS shows that BFO is able to compactly represent the complex solution functions.

Eriksson et al. [126] proposed the trust region Bayesian optimization (TuRBO) method in order to tackle the high-dimensional RL problems. They notice that the difficulty of high-dimensional optimization in RL comes from the plentiful local optima and the heterogeneity of the objective function (e.g., the sparse rewards in RL may lead to the objective function being nearly constant in a large region of the search space). This difficulty makes the task of learning a global surrogate model challenging. Thus, TuRBO maintains a collection of simultaneous local probabilistic models. Each local surrogate model shares the same benefits with Bayesian probabilistic modeling, and at the same time enables the heterogeneous modeling of the objective function. In TuRBO, a multi-armed bandit strategy is adopted to globally allocate the samples among the trust regions. The empirical results show that TuRBO outperforms the compared BO, EAs, and stochastic optimization on the RL benchmarks.

4.4 Classification based model parameter updating

The classification-based derivative-free optimization methods [31, 127–129] are recently proposed for non-convex functions with sample-complexity guarantees. The researchers notice that many derivative-free optimization methods are model-based, e.g., BO employs GP to model the function. The model-based optimization approaches learn a model from the evaluated solutions, and the model is then utilized to guide the sampling of solutions in the next iteration. The classification-based derivative-free optimization methods use a particular type of model, classification model from machine learning, to model the objective function. A classification model learns to classify the solutions in the search space into two categories, the good/positive and the bad/negative ones, according to the quality of the sampled solutions. The learned classifier partitions the search space into the good and bad regions. And then the solutions are sampled from the good regions with high probability.

Yu et al. [31] attempted to answer the crucial questions

of classification-based derivative-free optimization, including which factors of a classifier effect the optimization performance, and which function class can be efficiently solved. They identify the critical factors, the error-target dependence and the shrinking rate, and propose the randomized coordinate shrinking (RACOS) classification algorithm to efficiently learn the classifier for both continuous and discrete search space. Given a set of good/positive and bad/negative solutions, RACOS learns an axis-aligned hyper-rectangle to cover all the positive but no negative solutions, and at the same time, the learning process is randomized and the hyper-rectangle is highly shrunk to meet the critical factors disclosed. However, RACOS needs to sample a batch of solutions in each iteration to update the classification model, while in RL, the environment often only offers the sequential policy evaluation. This means RACOS cannot be directly applied to policy search, where solutions are sampled sequentially. To address this issue, Hu et al. [44] further proposed a sequential version of RACOS called SRACOS, which is tailored to direct policy search in RL. SRACOS adapts the classification-based optimization for sequential sampled solutions by forming the sample batch via reusing the historical solutions. In each iteration, it only samples one solution, replaces a historical solution with the newly sampled one, and then updates the classifier as RACOS. They introduce three replacing strategies for maintaining the historical solutions, i.e., replacing the worst one, replacing by a randomized mechanism, and replacing the one that has the largest margin from the best-so-far solution. The empirical effectiveness is verified on the helicopter hovering task and controlling tasks in OpenAI Gym [87–89], and the neural network is used to represent the policy. The results indicate that SRACOS can surpass CMA-ES, CE, BO with respect to the total reward.

5 Derivative-free model selection in reinforcement learning

Deep reinforcement learning uses the deep neural networks to model the policy or the value function. The final determination of model depends on two aspects: the weights of connections in a neural network and the topology/architecture of a neural network. In the fourth section, we mainly review the cases when the topology of a neural network is fixed. Namely, the number of hidden layers, the number of hidden layer neurons, the edge set of connected neurons and the like are all fixed in advance. Under the condition of fixed neural network topology, the derivative-free methods are used to optimize the weights of connections to search for the optimal policy. However, relying on the fixed topology representation imposes some limitations, since it requires the user to correctly specify a good topology representation in advance. Unfortunately, in most tasks, the user cannot correctly guess the appropriate topology representation. Selecting an overly simple topology representation could result in the insufficient model expression ability. Even if the optimization algorithm can find out the optimal solution under this simple representation, the policy corresponding to the found optimal solution may still be unsatisfactory. On the other hand, selecting an overly complex topology representation could significantly improves the expressive power of the model, but the huge search space makes the optimization algorithm inefficient, and thus it is difficult to find out the optimal policy. Therefore, many works are devoted to developing the methods that can automatically finding out the appropriate topology representation of neural network. And the review of neuroevolution for learning a neural network architecture can be found from [62, 63].

This section mainly reviews the methods for optimizing neural network topologies using derivative-free optimization, where neural networks can represent the policy or the value function in reinforcement learning. The network topology and connection weights are variable at the same time, which further improves the expressiveness and flexibility of the reinforcement learning model. However, it challenges the derivative-free optimization algorithms. The search space becomes larger and more complex, and the evaluation of model quality becomes more time consuming and labor intensive.

The earliest and simplest work to optimize the neural network topologies using evolutionary algorithms might be traced back to the structured genetic algorithm (sGA) [130]. sGA uses a two-part representation to describe each neural network. The first part represents the connectivity of the neural network in the form of a binary matrix. In this binary matrix, the rows and columns correspond to the neuron nodes in the network, and the value of each element in the matrix indicates whether there is an edge connecting a given pair of nodes. The second part represents the weight of each edge in the neural network. Via evolving these binary matrices with the connection weights, sGA can automatically discover the appropriate the network topology. However, sGA still has some restrictions. When a new topology is introduced (e.g., adding a new edge), the quality of the corresponding solution may be poor due to the fact that the weight associated with the new topology has not been optimized. Even if the topology ultimately corresponds to a better policy.

Stanley and Miikkulainen [131, 132] proposed the neuroevolution of augmenting topologies (NEAT), which is a well-known approach of topology and weight evolving artificial neural networks (TWEANNs), and achieved a significant performance gains in RL. To represent networks of different topologies, NEAT uses a flexible genetic coding. Each network is described by an edge list, and each edge describes the connection between two neuron nodes and the weight. In NEAT, the architecture is evolved in a way of incremental growth from minimal structure. New architecture is added incrementally as structural mutations occur, and only those architectures survive that are found to be useful. During mutation, one can add new nodes or new connections to the network, as shown in Fig. 5. To avoid the catastrophic crossovers, NEAT introduces the innovation numbers to record and track the historical origin of

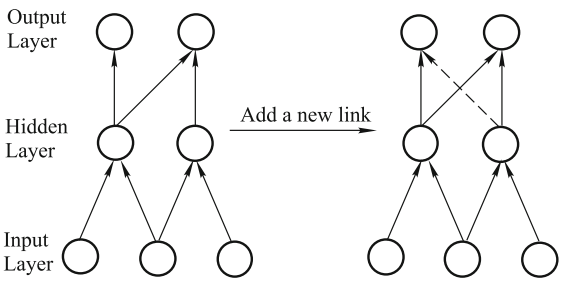

Fig. 5 A new link (edge) is added between two existing nodes in NEAT [131, 132]

each individual. Whenever a new individual emerges through mutation, it receives a unique innovation number that belongs to itself. Therefore, the innovation numbers can be regarded as the chronology of all individuals produced during the evolutionary process. Experimental result on the pole-balancing benchmark shows that NEAT can be faster and better than the compared fixed-topology methods therein.

Taylor et al. [42] conducted a detailed empirical comparison between NEAT and Sarsa [133] in the Keepaway RL benchmark based on robot soccer. Sarsa is a kind of temporal difference [1] methods that learn a value function to estimate the expected total reward for taking a particular action given a state. The results show that NEAT can learn better policies than Sarsa, but NEAT requires more fitness evaluations to achieve so. The results on two variations of Keepaway show that Sarsa can learn better policies when the task is fully observable, and NEAT can learn faster when the fitness function is deterministic. Whiteson and Stone [134, 135] enhanced the sample efficiency of evolutionary value function approximation via combining NEAT and a temporal difference method. The proposed algorithm can automatically discover the appropriate topologies for pre-trained neural network function approximators, and exploit the off-policy nature of a temporal difference method.

Kohl and Miikkulainen [136] noticed that NEAT could perform poorly on the fractured problems, where the correct action varies discontinuously as the agent moves from state to state. They introduce a method to measure the degree of fracture by utilizing the concept of function variation, and propose RBF-NEAT and Cascade-NEAT to improve the performance on the fractured problems through biasing or constraining the search for network topologies towards the local solutions. Gauci and Stanley [137] proposed the hypercube-based neuroevolution of augmenting topologies (HyperNEAT) that is able to exploit the geometric regularities in the two-dimensional game screen. Hausknecht et al. [138] introduced a HyperNEAT-based general game playing (HyperNEAT-GGP) method to play Atari games. HyperNEAT-GGP reduces the learning complexity from the raw game screen by using a game-independent visual processing hierarchy aimed to identify the objects and entities on the screen. And the identified ones are input of HyperNEAT. The effectiveness of HyperNEAT-GGP is verified on Asterix and Freeway.

Ebrahimi et al. [139] introduced an approach to learn the control policy for the autonomous driving task aimed to minimize crashes and safety violations during training. The proposed approach learns to generate an optimal network topology from demonstrations by utilizing a new reward function that simultaneously optimizes model size and accuracy. They use a recurrent neural network to sequentially generate the description of layers of an architecture from a given design space, which is inspired by [140]. The variable length architectures are searched by an enhanced evolution strategy with a modification in noise generation. Finally, by combining this derivative-free policy search with demonstrations, the proposed approach can learn a policy that adapts to the new environment based on the rewards in the target domain. The experimental result shows that, when the agent learns to drive in a real simulation environment, this approach can learn more safely than the compared baseline method and has fewer cumulative crash times in the life cycle of the agent. Gaier and Ha [141] attempted to answer the question of how important are the weight parameters of a neural network compared to its architecture. To deemphasize the importance of weights in the neural networks, they assign a single shared weight parameter to every network connection from a uniform random distribution, and only search for the architectures that perform well on a wide range of weights without explicit weight training. The proposed search method is inspired by NEAT [131, 132]. The empirical result shows that this method is able to find the minimal neural network architectures that perform well on a set of continuous control tasks only with a random weight parameter.

6 Derivative-free exploration in reinforcement learning

In most cases, RL algorithms share the exploration-learning framework [55]. An agent explores and interacts with an unknown environment to learn the optimal policy that maximizes the total reward from the exploration samples. The exploration samples involve states, state transitions, actions and rewards. Exploration is necessary in RL. Because achieving the best total reward on the current trajectory samples is not the ultimate goal, and the agent should visit the states that have not been visited before so as to collect better trajectory samples. This means that the agent should not follow the current policy tightly, and thus the exploration mechanisms need to encourage the agent to deviate from the previous trajectory paths properly and drive it to the regions with uncertainty. Derivative-free RL is naturally equipped with the exploration strategies. In the search process of derivative-free optimization methods, the designed mechanisms for sampling solutions and rules for updating model always consider the exploration. To name a few, the mutation and crossover operators in GAs help to explore the solution space, the GP upper confidence bound in BO uses variance to drive the search direction towards the regions with uncertainty, and the classification-based optimization algorithms usually adopt the global sampling to realize the exploration. Thus, derivative-free optimization methods can take part of the duty of exploration for RL when updating the policy or value function models from samples. Recently, some problem-dependent derivative-free exploration methods that could improve the sample efficiency have been proposed.

Conti et al. [142] proposed the NSR-ES method, which attempted to enhance the ability of exploration in evolution strategies for deep RL. NSR-ES uses the novelty search (NS) [105] to explore. NS encourages policies to engage in different behaviors than those previously seen. The encouragement of different behaviors is realized through computing the novelty of the current policy with respect to the previously generated policies and then pushing the population distribution towards the regions in parameter space with high novelty. NS is hybridized with ES to improve its performance on sparse or deceptive deep reinforcement learning tasks. The proposed NSR-ES algorithm has been tested in the MuJoCo and Atari [87–89], and the overall performance is superior to the classic evolution strategies. Lehman et al. [110] noticed that the simple mutation operators, e.g., Gaussian mutation, may lead to the catastrophic consequences on the

behavior of an agent in deep RL due to the drastic differences in the sensitivity of parameters. Therefore, they propose a set of safe mutation operators that facilitate exploration without dramatically varying the network behaviors. The safe mutation operators scale the degree of mutation of each individual weight according to the sensitivity of the outputs of network to that weight. And it is realized by computing the gradient of outputs with respect to the weights.

Chen and Yu [143] considered the exploration as an extra optimization problem in reinforcement learning, and realized the exploration component guided by a state-of-the-art classification-based derivative-free optimization algorithm, rather than directly applying derivative-free optimization to both exploration and learning. The proposed method searches for the high-quality exploration policy globally by setting the performance of the sampled trajectories as the fitness value of that exploration policy. The target policy is optimized by the deep deterministic policy gradient (DDPG) algorithm [144] on those explored trajectories. This method could overcome the sample inefficiency in derivative-free optimization and the exploration locality in gradient-based one. Vemula et al. [145] conducted a theoretical study on when exploring in the parameter space is better/worse than exploring in the action space by the black-box optimizer. They reveal that, when the dimensionality of action space and the horizon length are small, as well as the dimensionality of parameter space is large, exploration in the action space is preferred. Otherwise, when the horizon length is long and the dimensionality of policy parameter space is low, exploration in the parameter space is preferred.

Khadka and Tumer [109] proposed the evolutionary reinforcement learning (ERL) method. The core idea behind ERL is to combine the genetic algorithm (GA) and DDPG algorithm [144]. The algorithmic flow is illustrated in Fig. 6. The algorithm can be divided into the genetic algorithm module and the reinforcement learning module. In the genetic algorithm module, ERL generates n actors for parallel sampling, accumulates the cumulative reward of the sampled trajectories as the fitness of these n actors, and then constructs n new actors according to the mutation process of the genetic algorithm for the next sampling. In the reinforcement learning module, ERL collects samples of n actors into the experience replay buffer, uses the policy of reinforcement learning algorithm itself to perform additional sampling, and adds the sampling results to the experience replay buffer. Then, according to the DDPG algorithm which is a gradient-based one, a mini-batch sample with a fixed number of rounds is selected from the experience replay buffer to optimize the policy. At last, in order to inject the gradient information of policy into the genetic algorithm process, ERL periodically uses the policy to replace the policy with the worst fitness value among n actors before the genetic algorithm performs further sampling. ERL is tested on multiple environments in MuJoCo [87, 89], and the result shows that ERL is superior to the compared derivative-free algorithms and gradient-based ones with respect to both algorithmic effectiveness and sample efficiency. For instance, as shown in Fig. 7, we can observe that DDPG can easily solve the standard inverted double pendulum task, while fails on the hard one. Both tasks are similar for the evolutionary algorithm (EA). ERL is able to inherit the merits

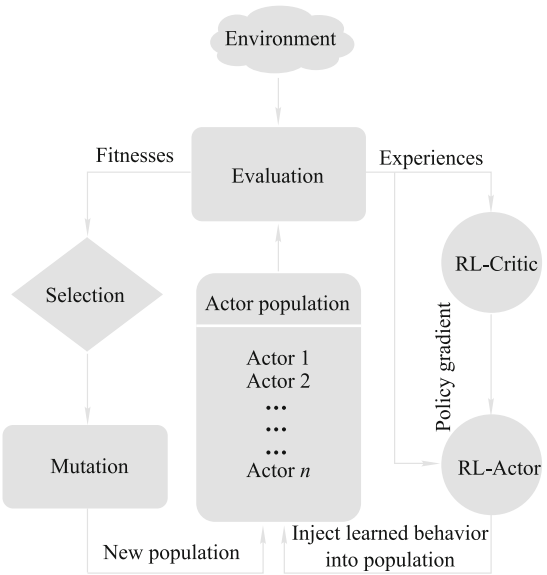

Fig. 6 Figure from [109] that illustrates the structure of ERL

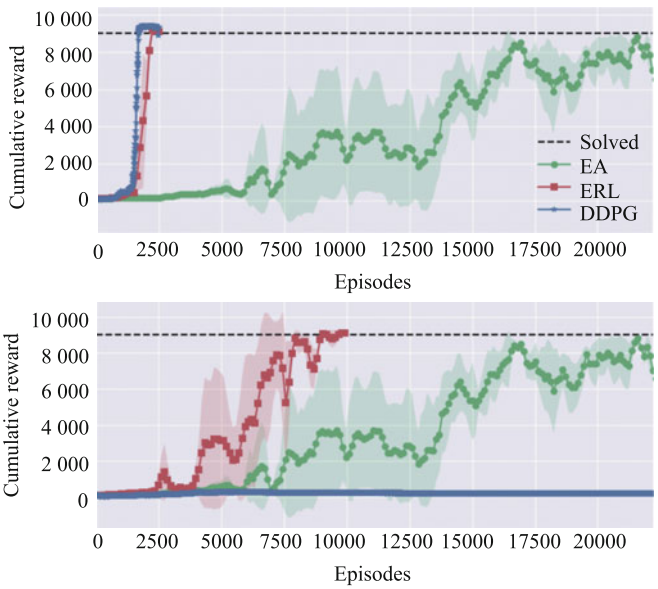

Fig. 7 Figure from [109] that compares the performance (measured by the cumulative reward) of the proposed ERL, evolutionary algorithm, and DDPG on the standard (upper sub-figure) and hard (lower sub-figure) inverted double pendulum task

of both DDPG and EA, and successfully solves both standard and hard tasks similar to EA while utilizing the gradients for better sample efficiency similar to DDPG.

Colas et al. [146] proposed the GEP-PG algorithm that splits reinforcement learning into two parts: global exploration and policy gradient. In the global exploration process (GEP), it uses a method similar to novelty search to generate a variety of exploration policies based on the policy behavior space. Specifically, the GEP process samples a batch of policy behavior features from the behavior space. The behavior space is an artificially defined mapping from the policy trajectory to a vector, which is used to represent the type of a policy. It then uses the neighborhood method to find the clustering center for each policy behavior. After that, each clustering center is perturbed to generate a new set of exploration policies. After the new ex-

ploration policy is sampled, the behavior features of the exploration policy are marked, and according to the newly added marks, the cluster centers are re-generated by the neighborhood method. This process is repeated so that the generated exploration policies can be placed in different goals as much as possible, which could ensure the diversity of the generated policies. In the policy gradient (PG) process, it combines with the deep deterministic policy gradient (DDPG) algorithm [144], and adds the samples collected by the GEP process to the experience replay buffer. PG updates the policy by the fixed mini-batch. The GEP-PG approach enhances the diversity of exploration policies and is tested on the continuous mountain car and half-cheetah environments in MuJoCo [87,89]. GEP-PG shows the better performance than the compared algorithms. Notably, some of the aforementioned methods, e.g., GEP-PG and ERL, are essentially a combination of derivative-free and gradient-based optimization. The derivative-free ones are used to better explore, and the gradient-based ones are used to better exploit.

7 Parallel and distributed derivative-free reinforcement learning

Although derivative-free methods could bring some good news to RL with respect to optimization and exploration, they mostly suffer from low convergence rate. Derivative-free optimization methods often require to sample a large amount of solutions before convergence, even if the objective function is convex or smooth [35,37,60]. And the issue of slow convergence becomes more serious as the dimensionality of a search space increases [37,61]. Obviously, this issue will block the further application of derivative-free methods to RL. Fortunately, many derivative-free optimization methods are population-based. A population of solutions is maintained and improved iteratively. This characteristic makes them highly parallel. Thus, derivative-free optimization methods can be accelerated by parallel or distributed computation, which alleviates their slow convergence. Furthermore, for parallel/distributed derivative-free methods, the data communication cost is lower compared with gradient-based ones, since only one-dimensional scalars (fitness values) instead of gradient vectors or Hessian matrices need to be conveyed. This merit could further compensates for the low convergence rate.

The ES method [45] mentioned in the fourth section is a typical parallel case of the derivative-free method. Moreover, by the virtue of high parallelization property of ES, a novel communication strategy based on common random numbers is introduced to further reduce the communication cost, and it can make the algorithm well scaled to a large number of parallel workers. It takes only 10 minutes to reach 6000 long-term reward in the 3D Humanoid environment with 1440 cores. And with the increasing of core amount, the consumption of training time shows a relatively stable linear decline. SRACOS [44] is a classification-based derivative-free optimization algorithm for direct policy search. Its distributed version ZOOpt [147], an open-source toolkit, is implemented by the Julia language. And the experimental result shows that the algorithm can support more than 100 processes for parallel computing. The parallel strategies of ES and SRACOS are similar. Both are based on a certain generation module to construct multiple policies simultaneously. ES is based on Gaussian perturbation for current parameters, and SRACOS is based on a randomized coordinate shrinking classifier. Policies are run and their fitness values are evaluated in parallel. The generation module is updated by these fitness values, and the next batch of policies is constructed.

Reinforcement learning algorithms sometimes are sensitive to the hyper-parameters, such as learning rate and entropy penalty coefficient. Finely tuned hyper-parameters can accelerate learning and improve the performance of policy. Previous hyper-parameter adjustment methods are parallel search algorithms, which use the performance of final policy under different hyper-parameters to adjust hyper-parameters. Such methods require a large amount of computational resources to simultaneously optimize multiple models. Population-based training (PBT) [148] is a framework to simultaneously optimize model parameters and adjust hyper-parameters with a derivative-free method. The method is shown in Algorithm 1. The step$(\theta \mid h)$ is a function to conduct model optimization according to the current hyper-parameters h and the model parameters θ. The optimization method depends on tasks (e.g., SGD for supervised learning and DDPG for reinforcement learning). The eval(h) is a function to realize performance evaluation. The evaluation method also depends on tasks (e.g., the loss for supervised learning and the total reward of a policy for reinforcement learning). If the model evaluation result p is better than a threshold, population-based evolution will start to adjust the hyper-parameters h. There are two main functions in the population-based evolution, i.e., exploit$(h,\theta,p,\mathcal{P})$ and explore$(h',\theta',\mathcal{P})$. The exploit$(h,\theta,p,\mathcal{P})$ function exploits the best hyper-parameters h and its model parameters θ to conduct evolution. For example, replacing the worst solution in $\mathcal{P}$ with the best one. The explore$(h',\theta',\mathcal{P})$ function explores the new unknown hyper-parameters h based on the current population $\mathcal{P}$. For example, perturbing h' with Gaussian noise to generate the new hyper-parameters h. After population evolution, the new hyper-parameters h as well as its model parameters θ will be added to $\mathcal{P}$. The main characteristic of PBT is that the process of parameter optimization and hyper-parameters adjustment is hybrid, which can not only reduce the computational

Algorithm 1 Population-based training (PBT) [148]

Procedure: TRAIN ($\mathcal{P}$)
initial population $\mathcal{P}$
for $(\theta,h,p,t) \in \mathcal{P}$ (asynchronously in parallel) **do**
  while not end of training **do**
    one step of optimization: $\theta \leftarrow \text{step}(\theta \mid h)$
    current model evaluation: $p \leftarrow \text{eval}(h)$
    if ready$(p,t,\mathcal{P})$ **then**
      use the rest of population to find better solution:
      $h',\theta' \leftarrow \text{exploit}(h,\theta,p,\mathcal{P})$
      if $\theta \neq \theta'$ **then**
        produce new h: $h,\theta \leftarrow \text{explore}(h',\theta',\mathcal{P})$
        new model evaluation: $p \leftarrow \text{eval}(\theta)$
      end if
    end if
    update $\mathcal{P}$ with new $(\theta,h,p,t+1)$
  end while
end for
return θ with the highest p in $\mathcal{P}$

cost but also help algorithm adjust hyper-parameters in dynamic environments. PBT has succeeded in RL when combining with A3C [90]. The empirical results in the DeepMind Lab 3D environment [149], Atari games [88, 89], and the StarCraft II environment tasks [150] show that PBT increases the final performance of the agents when trained with the same number of episodes.

Ray [151], a distributed framework proposed by the researchers from UC Berkeley, integrates the distributed implementation of the PBT algorithm. Online meta-learning by parallel algorithm competition (OMPAC) [152] is also a derivative-free distributed hybrid optimization algorithm in RL. This algorithm can be regarded as an improvement based on PBT. The main difference between OMPAC and PBT is the evolution mechanism. In OMPAC, all solutions are evaluated in a synchronized manner after a fixed number of samples, and OMPAC uses the stochastic universal sampling [153] to select solutions for continual learning. In PBT, solutions are evaluated asynchronously (e.g., after a fixed number of training steps). Jaderberg et al. [154] demonstrated that the agent using only pixels and game points as input could learn to play highly competitively in a popular multi-player first-person video game. The ingredients in the proposed method include PBT of agents, internal reward optimization, and hierarchical RL with scalable computational architectures. Jung et al. [155] proposed the population-guided parallel policy search (P3S) to improve the performance of RL. P3S employs a population to search a better policy by exploiting the best policy information which is similar to PBT. However, the way of using the best policy information is different. In P3S, instead of copying the parameter of the best learner, it introduces a soft manner to guide the population for better search in the policy space. P3S maintains a batch of identical learners with their own value-functions and policies sharing a common experience replay buffer, and searches a good policy cooperated by the guidance of the best policy information. In a nutshell, parallel and distributed computation is necessary for derivative-free reinforcement learning.

8 Discussion and conclusion

In this article, we summarize some recent progress in applying derivative-free optimization to reinforcement learning. Since derivative-free optimization is a generally applicable tool, it can be utilized in different levels and aspects of reinforcement learning. Here, we focus on the aspects of parameter updating, model selection, exploration, and parallel/distributed derivative-free methods. Due to the complexity of policy search in reinforcement learning, successful research studies of using derivative-free optimization are noticeable in all of these aspects.

Derivative-free reinforcement learning approaches have their own advantages. Firstly, during the optimization process, they do not perform gradient back-propagation, are easy to train, possess the ability of search globally, do not care whether the reward function is sparse or dense, do not care the length of time horizons, do not require to carefully tune the discount factor, and can be applied to optimize the non-differentiable policy functions. Secondly, they could provide better exploration tailored to the problems. Thirdly, they are highly suitable for parallelization and only require the low communication cost. Usually, the amount of information that needs to be exchanged among workers does not rely on the size of neural networks.

At the same time, we also notice that the combination of derivative-free optimization and reinforcement learning has several limitations. These are mainly from the issues of current derivative-free optimization methods. The foremost ones could be the issues of sample complexity and scalability.

Derivative-free optimization algorithms could be comparable to some state-of-the-art gradient-based ones in reinforcement learning tasks, but usually they require more samples [45]. The low sample efficiency may block the further application of derivative-free optimization algorithms to reinforcement learning. Recently, some emerging works focusing on improving sample efficiency by the way of introducing sample reuse [96, 156], surrogate models [157], importance sampling [158], and momentum [159, 160]. However, these advances are far from enough and more future works on this direction are urgently needed.

Due to the sampling mechanism, derivative-free optimization methods have limitation of scaling up to the search space with very high dimensionality. This is a crucial issue for deep reinforcement learning. Deep neural network models often have more than a million parameters, for which parameter optimization directly by derivative-free optimization can be inefficient. While some recent works dedicated to tackling the scalability issue [61, 126, 161–167], more future works on developing the scalable derivative-free optimization methods tailored to reinforcement learning are appealing.

Other issues include noise-sensitivity as well as structure-insensitivity. Since derivative-free optimization relies on solution evaluations, noisy evaluation can badly affect the policy search in reinforcement learning [168]. Reinforcement learning usually has inner structures that can help better solve the learning, such as hierarchical policy models and curiosity-driven exploration, while derivative-free methods commonly ignore the inner structure of the problem. How to make the derivative-free optimization well aware of the inner structure and utilizing the structure is also an important direction and remains under explored.

For the above issues, the hybrid of derivative-free and gradient-based algorithms could be a potential and promising way. Some existing successful attempts reviewed in this article indicate that, by fusing derivative-free and gradient-based algorithms in a clever manner, the strengths of both sides could be absorbed, and more powerful reinforcement learning algorithms would be expected.

With the successful cases and fast progress, we expect in a near future that derivative-free methods will play an even more important role in developing novel and efficient reinforcement learning approaches, and hope that this review article will serve as a catalyst for this goal.

Acknowledgements This work was supported by the Program A for Outstanding PhD Candidate of Nanjing University, National Science Foundation of China (61876077), Jiangsu Science Foundation (BK20170013), and Collaborative Innovation Center of Novel Software Technology and Industrialization. Yang Yu is the corresponding author of this article. The authors would like to thank Xiong-Hui Chen and Zhao-Hua Li for improving the article.

References

1. Sutton R S, Barto A G. Reinforcement Learning: An Introduction. Cambridge, Massachusetts: MIT Press, 1998
2. Wiering M, Van Otterlo M. Reinforcement Learning: State-of-the-Art. Berlin, Heidelberg: Springer, 2012
3. Dietterich T G. Machine learning research: four current directions. Artificial Intelligence Magazine, 1997, 18(4): 97–136
4. Mnih V, Kavukcuoglu K, Silver D, Rusu A A, Veness J, Bellemare M G, Graves A, Riedmiller M, Fidjeland A K, Ostrovski G. Human-level control through deep reinforcement learning. Nature, 2015, 518(7540): 529–533
5. Silver D, Huang A, Maddison C J, Guez A, Sifre L, Driessche G V D, Schrittwieser J, Antonoglou I, Panneershelvam V, Lanctot M. Mastering the game of Go with deep neural networks and tree search. Nature, 2016, 529(7587): 484–489
6. Silver D, Hubert T, Schrittwieser J, Antonoglou I, Lai M, Guez A, Lanctot M, Sifre L, Kumaran D, Graepel T, Lillicrap T, Simonyan K, Hassabis D. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. Science, 2018, 362(6419): 1140–1144
7. Abbeel P, Coates A, Quigley M, Ng A Y. An application of reinforcement learning to aerobatic helicopter flight. In: Proceedings of the 19th International Conference on Neural Information Processing Systems. 2006, 1–8
8. Zoph B, Le Q V. Neural architecture search with reinforcement learning. In: Proceedings of the 5th International Conference on Learning Representations. 2017
9. Huang C, Lucey S, Ramanan D. Learning policies for adaptive tracking with deep feature cascades. In: Proceedings of the IEEE International Conference on Computer Vision. 2017, 105–114
10. Yu L, Zhang W, Wang J, Yu Y. SeqGAN: sequence generative adversarial nets with policy gradient. In: Proceedings of the 31st AAAI Conference on Artificial Intelligence. 2017, 2852–2858
11. Wang Y C, Usher J M. Application of reinforcement learning for agent-based production scheduling. Engineering Applications of Artificial Intelligence, 2005, 18(1): 73–82
12. Choi J J, Laibson D, Madrian B C, Metrick A. Reinforcement learning and savings behavior. The Journal of Finance, 2009, 64(6): 2515–2534
13. Shi J C, Yu Y, Da Q, Chen S Y, Zeng A. Virtual-taobao: virtualizing real-world online retail environment for reinforcement learning. In: Proceedings of the 33rd AAAI Conference on Artificial Intelligence. 2019, 4902–4909
14. Boyan J A, Littman M L. Packet routing in dynamically changing networks: a reinforcement learning approach. In: Proceedings of the 6th International Conference on Neural Information Processing Systems. 1993, 671–678
15. Frank M J, Seeberger L C, O'reilly R C. By carrot or by stick: cognitive reinforcement learning in parkinsonism. Science, 2004, 306(5703): 1940–1943
16. Samejima K, Ueda Y, Doya K, Kimura M. Representation of action-specific reward values in the striatum. Science, 2005, 310(5752): 1337–1340
17. Shalev-Shwartz S, Shamir O, Shammah S. Failures of gradient-based deep learning. In: Proceedings of the 34th International Conference on Machine Learning. 2017, 3067–3075
18. Conn A R, Scheinberg K, Vicente L N. Introduction to Derivative-Free Optimization. Philadelphia, PA: SIAM, 2009
19. Kolda T G, Lewis R M, Torczon V. Optimization by direct search: new perspectives on some classical and modern methods. SIAM Review, 2003, 45(3): 385–482
20. Rios L M, Sahinidis N V. Derivative-free optimization: a review of algorithms and comparison of software implementations. Journal of Global Optimization, 2013, 56(3): 1247–1293
21. Sigaud O, Wilson S W. Learning classifier systems: a survey. Soft Computing, 2007, 11(11): 1065–1078
22. Moriarty D E, Schultz A C, Grefenstette J J. Evolutionary algorithms for reinforcement learning. Journal of Artificial Intelligence Research, 1999, 11: 241–276
23. Whiteson S. Evolutionary computation for reinforcement learning. In: Wiering M, van Otterlo M, eds. Reinforcement Learning: State-of-the-Art. Springer, Berlin, Heidelberg, 2012, 325–355
24. Bellman R. A Markovian decision process. Journal of Mathematics and Mechanics, 1957, 6(5): 679–684
25. Bartlett P L, Baxter J. Infinite-horizon policy gradient estimation. Journal of Artificial Intelligence Research, 2001, 15: 319–350
26. Holland J H. Adaptation in Natural and Artificial Systems. Ann Arbor, MI: The University of Michigan Press, 1975
27. Hansen N, Müller S D, Koumoutsakos P. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). Evolutionary Computation, 2003, 11(1): 1–18
28. Shahriari B, Swersky K, Wang Z, Adams R P, Freitas d N. Taking the human out of the loop: a review of Bayesian optimization. Proceedings of the IEEE, 2016, 104(1): 148–175
29. De Boer P T, Kroese D P, Mannor S, Rubinstein R Y. A tutorial on the cross-entropy method. Annals of Operations Research, 2005, 134(1): 19–67
30. Munos R. From bandits to Monte-Carlo tree search: the optimistic principle applied to optimization and planning. Foundations and Trends in Machine Learning, 2014, 7(1): 1–129
31. Yu Y, Qian H, Hu Y Q. Derivative-free optimization via classification. In: Proceedings of the 30th AAAI Conference on Artificial Intelligence. 2016, 2286–2292
32. He J, Yao X. Drift analysis and average time complexity of evolutionary algorithms. Artificial Intelligence, 2001, 127(1): 57–85
33. Yu Y, Zhou Z H. A new approach to estimating the expected first hitting time of evolutionary algorithms. Artificial Intelligence, 2008, 172(15): 1809–1832
34. Bull A D. Convergence rates of efficient global optimization algorithms. Journal of Machine Learning Research, 2011, 12: 2879–2904
35. Jamieson K G, Nowak R D, Recht B. Query complexity of derivative-free optimization. In: Proceedings of the 25th International Conference on Neural Information Processing Systems. 2012, 2681–2689
36. Yu Y, Qian H. The sampling-and-learning framework: a statistical view of evolutionary algorithms. In: Proceedings of the 2014 IEEE Congress on Evolutionary Computation. 2014, 149–158
37. Duchi J C, Jordan M I, Wainwright M J, Wibisono A. Optimal rates for zero-order convex optimization: the power of two function evaluations. IEEE Transactions on Information Theory, 2015, 61(5): 2788–2806
38. Yu Y, Qian C, Zhou Z H. Switch analysis for running time analysis of evolutionary algorithms. IEEE Transactions on Evolutionary Computation, 2015, 19(6): 777–792
39. Kawaguchi K, Kaelbling L P, Lozano-Perez T. Bayesian optimization with exponential convergence. In: Proceedings of the 28th International Conference on Neural Information Processing Systems. 2015, 2809–2817
40. Kawaguchi K, Maruyama Y, Zheng X. Global continuous optimization with error bound and fast convergence. Journal of Artificial Intelligence Research, 2016, 56: 153–195
41. Mitchell M. An Introduction to Genetic Algorithms. Cambridge, MA: MIT Press, 1998
42. Taylor M E, Whiteson S, Stone P. Comparing evolutionary and temporal difference methods in a reinforcement learning domain. In: Proceedings of the 2006 Conference on Genetic and Evolutionary Computation. 2006, 1321–1328
43. Abdolmaleki A, Lioutikov R, Peters J, Lau N, Reis L P, Neumann G. Model-based relative entropy stochastic search. In: Proceedings of the

28th International Conference on Neural Information Processing Systems. 2015, 3537–3545
44. Hu Y Q, Qian H, Yu Y. Sequential classification-based optimization for direct policy search. In: Proceedings of the 31st AAAI Conference on Artificial Intelligence. 2017, 2029–2035
45. Salimans T, Ho J, Chen X, Sidor S, Sutskever I. Evolution strategies as a scalable alternative to reinforcement learning. 2017, arXiv:1703.03864
46. Snoek J, Larochelle H, Adams R P. Practical Bayesian optimization of machine learning algorithms. In: Proceedings of the 25th International Conference on Neural Information Processing Systems. 2012, 2960–2968
47. Thornton C, Hutter F, Hoos H H, Leyton-Brown K. Auto-WEKA: combined selection and hyperparameter optimization of classification algorithms. In: Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2013, 847–855
48. Real E, Moore S, Selle A, Saxena S, Suematsu Y L, Tan J, Le Q V, Kurakin A. Large-scale evolution of image classifiers. In: Proceedings of the 34th International Conference on Machine Learning. 2017, 2902–2911
49. Real E, Aggarwal A, Huang Y, Le Q V. Regularized evolution for image classifier architecture search. In: Proceedings of the AAAI Conference on Artificial Intelligence. 2019, 4780–4789
50. Zhang Y, Sohn K, Villegas R, Pan G, Lee H. Improving object detection with deep convolutional networks via Bayesian optimization and structured prediction. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2015, 249–258
51. Qian C, Yu Y, Zhou Z H. Subset selection by pareto optimization. In: Proceedings of the 28th International Conference on Neural Information Processing Systems. 2015, 1765–1773
52. Qian C, Shi J C, Yu Y, Tang K. On subset selection with general cost constraints. In: Proceedings of the 26th International Joint Conference on Artificial Intelligence. 2017, 2613–2619
53. Brown M, An B, Kiekintveld C, Ordóñez F, Tambe M. An extended study on multi-objective security games. Autonomous Agents and Multi-Agent Systems, 2014, 28(1): 31–71
54. Domingos P M. A few useful things to know about machine learning. Communications of the ACM, 2012, 55(10): 78–87
55. Yu Y. Towards sample efficient reinforcement learning. In: Proceedings of the 27th International Joint Conference on Artificial Intelligence. 2018, 5739–5743
56. Plappert M, Houthooft R, Dhariwal P, Sidor S, Chen R Y, Chen X, Asfour T, Abbeel P, Andrychowicz M. Parameter space noise for exploration. In: Proceedings of the 6th International Conference on Learning Representations. 2018
57. Pathak D, Agrawal P, Efros A A, Darrell T. Curiosity-driven exploration by self-supervised prediction. In: Proceedings of the 34th International Conference on Machine Learning. 2017, 2778–2787
58. Duan Y, Chen X, Houthooft R, Schulman J, Abbeel P. Benchmarking deep reinforcement learning for continuous control. In: Proceedings of the 33rd International Conference on Machine Learning. 2016, 1329–1338
59. Schulman J, Levine S, Abbeel P, Jordan M I, Moritz P. Trust region policy optimization. In: Proceedings of the 32nd International Conference on Machine Learning. 2015, 1889–1897
60. Bach F R, Perchet V. Highly-smooth zero-th order online optimization. In: Proceedings of the 29th Conference on Learning Theory. 2016, 257–283
61. Qian H, Yu Y. Scaling simultaneous optimistic optimization for high-dimensional non-convex functions with low effective dimensions. In: Proceedings of the 30th AAAI Conference on Artificial Intelligence. 2016, 2000–2006
62. Yao X. Evolving artificial neural networks. Proceedings of the IEEE, 1999, 87(9): 1423–1447
63. Stanley K O, Clune J, Lehman J, Miikkulainen R. Designing neural networks through neuroevolution. Nature Machine Intelligence, 2019, 1(1): 24–35
64. Such F P, Madhavan V, Conti E, Lehman J, Stanley K O, Clune J. Deep neuroevolution: genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. 2017, arXiv preprint arXiv:1712.06567
65. Morse G, Stanley K O. Simple evolutionary optimization can rival stochastic gradient descent in neural networks. In: Proceedings of the 2016 Conference on Genetic and Evolutionary Computation. 2016, 477–484
66. Zhang X, Clune J, Stanley K O. On the relationship between the OpenAI evolution strategy and stochastic gradient descent. 2017, arXiv preprint arXiv:1712.06564
67. Koutník J, Cuccu G, Schmidhuber J, Gomez F J. Evolving large-scale neural networks for vision-based reinforcement learning. In: Proceedings of the 2013 Conference on Genetic and Evolutionary Computation. 2013, 1061–1068
68. Hausknecht M J, Lehman J, Miikkulainen R, Stone P. A neuroevolution approach to general Atari game playing. IEEE Transactions on Computational Intelligence and AI in Games, 2014, 6(4): 355–366
69. Risi S, Togelius J. Neuroevolution in games: state of the art and open challenges. IEEE Transactions on Computational Intelligence and AI in Games, 2017, 9(1): 25–41
70. Chrabaszcz P, Loshchilov I, Hutter F. Back to basics: benchmarking canonical evolution strategies for playing atari. In: Proceedings of the 27th International Joint Conference on Artificial Intelligence. 2018, 1419–1426
71. Mania H, Guy A, Recht B. Simple random search of static linear policies is competitive for reinforcement learning. In: Proceedings of the 32nd International Conference on Neural Information Processing Systems. 2018, 1805–1814
72. Malik D, Pananjady A, Bhatia K, Khamaru K, Bartlett P, Wainwright M J. Derivative-free methods for policy optimization: guarantees for linear quadratic systems. In: Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics. 2019, 2916–2925
73. Hansen N, Arnold D V, Auger A. Evolution strategies. In: Kacprzyk J, Pedrycz W, eds. Springer Handbook of Computational Intelligence. Springer, Berlin, Heidelberg, 2015, 871–898
74. Hansen N, Ostermeier A. Adapting arbitrary normal mutation distributions in evolution strategies: the covariance matrix adaptation. In: Proceedings of 1996 IEEE International Conference on Evolutionary Computation. 1996, 312–317
75. Hansen N, Ostermeier A. Completely derandomized self-adaptation in evolution strategies. Evolutionary Computation, 2001, 9(2): 159–195
76. Heidrich-Meisner V, Igel C. Evolution strategies for direct policy search. In: Proceedings of the 10th International Conference on Parallel Problem Solving from Nature. 2008, 428–437
77. Heidrich-Meisner V, Igel C. Neuroevolution strategies for episodic reinforcement learning. Journal of Algorithms, 2009, 64(4): 152–168
78. Peters J, Schaal S. Natural actor-critic. Neurocomputing, 2008, 71(7-9): 1180–1190
79. Heidrich-Meisner V, Igel C. Hoeffding and Bernstein races for selecting policies in evolutionary direct policy search. In: Proceedings of the 26th International Conference on Machine Learning. 2009, 401–408
80. Stulp F, Sigaud O. Path integral policy improvement with covariance matrix adaptation. In: Proceedings of the 29th International Conference on Machine Learning. 2012
81. Szita I, Lörincz A. Learning Tetris using the noisy cross-entropy method. Neural Computation, 2006, 18(12): 2936–2941
82. Wierstra D, Schaul T, Peters J, Schmidhuber J. Natural evolution strategies. In: Proceedings of the 2008 IEEE Congress on Evolutionary Computation. 2008, 3381–3387

83. Wierstra D, Schaul T, Glasmachers T, Sun Y, Peters J, Schmidhuber J. Natural evolution strategies. Journal of Machine Learning Research, 2014, 15(1): 949–980
84. Salimans T, Goodfellow I J, Zaremba W, Cheung V, Radford A, Chen X. Improved techniques for training GANs. In: Proceedings of the 29th International Conference on Neural Information Processing Systems. 2016, 2226–2234
85. Geweke J. Antithetic acceleration of Monte Carlo integration in Bayesian inference. Journal of Econometrics, 1988, 38(1–2): 73–89
86. Brockhoff D, Auger A, Hansen N, Arnold D V, Hohm T. Mirrored sampling and sequential selection for evolution strategies. In: Proceedings of the 11th International Conference on Parallel Problem Solving from Nature. 2010, 11–21
87. Todorov E, Erez T, Tassa Y. MuJoCo: a physics engine for model-based control. In: Proceedings of 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems. 2012, 5026–5033
88. Bellemare M G, Naddaf Y, Veness J, Bowling M. The arcade learning environment: an evaluation platform for general agents. Journal of Artificial Intelligence Research, 2013, 47: 253–279
89. Brockman G, Cheung V, Pettersson L, Schneider J, Schulman J, Tang J, Zaremba W. OpenAI Gym. 2016, arXiv preprint arXiv:1606.01540
90. Mnih V, Badia A P, Mirza M, Graves A, Lillicrap T, Harley T, Silver D, Kavukcuoglu K. Asynchronous methods for deep reinforcement learning. In: Proceedings of the 33rd International Conference on Machine Learning. 2016, 1928–1937
91. Lehman J, Chen J, Clune J, Stanley K O. ES is more than just a traditional finite-difference approximator. In: Proceedings of the 2018 Conference on Genetic and Evolutionary Computation. 2018, 450–457
92. Choromanski K, Rowland M, Sindhwani V, Turner R E, Weller A. Structured evolution with compact architectures for scalable policy optimization. In: Proceedings of the 35th International Conference on Machine Learning. 2018, 969–977
93. Chen Z, Zhou Y, He X, Jiang S. A restart-based rank-1 evolution strategy for reinforcement learning. In: Proceedings of the 28th International Joint Conference on Artificial Intelligence. 2019, 2130–2136
94. Choromanski K, Pacchiano A, Parker-Holder J, Tang Y, Sindhwani V. From complexity to simplicity: adaptive ES-active subspaces for blackbox optimization. In: Proceedings of the 31st International Conference on Neural Information Processing Systems. 2019
95. Constantine P G. Active Subspaces — Emerging Ideas for Dimension Reduction in Parameter Studies. volume 2 of SIAM spotlights. Philadelphia, PA: SIAM, 2015
96. Liu G, Zhao L, Yang F, Bian J, Qin T, Yu N, Liu T Y. Trust region evolution strategies. In: Proceedings of the 33rd AAAI Conference on Artificial Intelligence. 2019, 4352–4359
97. Tang Y, Choromanski K, Kucukelbir A. Variance reduction for evolution strategies via structured control variates. In: Proceedings of International Conference on Artifical Intelligence and Statistics. 2020, 646-656
98. Fuks L, Awad N, Hutter F, Lindauer M. An evolution strategy with progressive episode lengths for playing games. In: Proceedings of the 28th International Joint Conference on Artificial Intelligence. 2019, 1234–1240
99. Houthooft R, Chen Y, Isola P, Stadie B C, Wolski F, Ho J, Abbeel P. Evolved policy gradients. In: Proceedings of the 31th International Conference on Neural Information Processing Systems. 2018, 5405–5414
100. Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O. Proximal policy optimization algorithms. 2017, arXiv preprint arXiv:1707.06347
101. Duan Y, Schulman J, Chen X, Bartlett P L, Sutskever I, Abbeel P. RL^2: Fast reinforcement learning via slow reinforcement learning. 2016, arXiv preprint arXiv:1611.02779
102. Finn C, Abbeel P, Levine S. Model-agnostic meta-learning for fast adaptation of deep networks. In: Proceedings of the 34th International Conference on Machine Learning. 2017, 1126–1135
103. Ha D, Schmidhuber J. Recurrent world models facilitate policy evolution. In: Proceedings of the 31st International Conference on Neural Information Processing Systems. 2018, 2455–2467
104. Yu W, Liu C K, Turk G. Policy transfer with strategy optimization. In: Proceedings of the 7th International Conference on Learning Representations. 2019
105. Lehman J, Stanley K O. Abandoning objectives: evolution through the search for novelty alone. Evolutionary Computation, 2011, 19(2): 189–223
106. Gangwani T, Peng J. Policy optimization by genetic distillation. In: Proceedings of the 6th International Conference on Learning Representations. 2018
107. Ross S, Gordon G J, Bagnell D. A reduction of imitation learning and structured prediction to no-regret online learning. In: Proceedings of the 14th International Conference on Artificial Intelligence and Statistics. 2011, 627–635
108. Bodnar C, Day B, Lió P. Proximal distilled evolutionary reinforcement learning. In: Proceedings of the 34th AAAI Conference on Artificial Intelligence. 2020, 3283–3290
109. Khadka S, Tumer K. Evolution-guided policy gradient in reinforcement learning. In: Proceedings of the 31st International Conference on Neural Information Processing Systems. 2018, 1196–1208
110. Lehman J, Chen J, Clune J, Stanley K O. Safe mutations for deep and recurrent neural networks through output gradients. In: Proceedings of the 2018 Conference on Genetic and Evolutionary Computation. 2018, 117–124
111. Fujimoto S, Hoof H, Meger D. Addressing function approximation error in actor-critic methods. In: Proceedings of the 35th International Conference on Machine Learning. 2018, 1582–1591
112. Rasmussen C E, Williams C K I. Gaussian Processes for Machine Learning. Cambridge, Massachusetts: MIT Press, 2006
113. Kushner H J. A new method of locating the maximum of an arbitrary multipeak curve in the presence of noise. Journal of Basic Engineering, 1964, 86: 97–106
114. Močkus J, Tiesis V, Žilinskas A. Toward global optimization. In: Dixon L C W, Szego G P, eds. The Application of Bayesian Methods for Seeking the Extremum. Elsevier, Amsterdam, Netherlands, 1978, 117–128
115. Srinivas N, Krause A, Kakade S M, Seeger M W. Gaussian process optimization in the bandit setting: no regret and experimental design. In: Proceedings of the 27th International Conference on Machine Learning. 2010, 1015–1022
116. Freitas d N, Smola A J, Zoghi M. Exponential regret bounds for Gaussian process bandits with deterministic observations. In: Proceedings of the 29th International Conference on Machine Learning. 2012
117. Brochu E, Cora V M, Freitas d N. A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. 2010, arXiv preprint arXiv:1012.2599
118. Wilson A, Fern A, Tadepalli P. Using trajectory data to improve Bayesian optimization for reinforcement learning. Journal of Machine Learning Research, 2014, 15(1): 253–282
119. Calandra R, Seyfarth A, Peters J, Deisenroth M P. An experimental comparison of Bayesian optimization for bipedal locomotion. In: Proceedings of the 2014 IEEE International Conference on Robotics and Automation. 2014, 1951–1958
120. Calandra R, Seyfarth A, Peters J, Deisenroth M P. Bayesian optimization for learning gaits under uncertainty — an experimental comparison on a dynamic bipedal walker. Annals of Mathematics and Artificial Intelligence, 2016, 76(1-2): 5–23
121. Marco A, Berkenkamp F, Hennig P, Schoellig A P, Krause A, Schaal S, Trimpe S. Virtual vs. real: trading off simulations and physical experiments in reinforcement learning with Bayesian optimization. In: Proceedings of the 2017 IEEE International Conference on Robotics and

Automation. 2017, 1557–1563
122. Letham B, Bakshy E. Bayesian optimization for policy search via online-offline experimentation. 2019, arXiv preprint arXiv:1904.01049
123. Swersky K, Snoek J, Adams R P. Multi-task Bayesian optimization. In: Proceedings of the 26th International Conference on Neural Information Processing Systems. 2013, 2004–2012
124. Vien N A, Zimmermann H, Toussaint M. Bayesian functional optimization. In: Proceedings of the 32nd AAAI Conference on Artificial Intelligence. 2018, 4171–4178
125. Vien N A, Dang V H, Chung T. A covariance matrix adaptation evolution strategy for direct policy search in reproducing kernel Hilbert space. In: Proceedings of The 9th Asian Conference on Machine Learning. 2017, 606–621
126. Eriksson D, Pearce M, Gardner J R, Turner R, Poloczek M. Scalable global optimization via local Bayesian optimization. In: Proceedings of the 32nd International Conference on Neural Information Processing Systems. 2019, 5497–5508
127. Lozano J A, Larranaga P, Inza I, Bengoetxea E. Towards a New Evolutionary Computation: advances on Estimation of Distribution Algorithms. Berlin, Germany: Springer-Verlag, 2006
128. Hashimoto T, Yadlowsky S, Duchi J C. Derivative free optimization via repeated classification. In: Proceedings of the 2018 International Conference on Artificial Intelligence and Statistics. 2018, 2027–2036
129. Zhou A, Zhang J, Sun J, Zhang G. Fuzzy-classification assisted solution preselection in evolutionary optimization. In: Proceedings of the 33rd AAAI Conference on Artificial Intelligence. 2019, 2403–2410
130. Dasgupta D, McGregor D. Designing application-specific neural networks using the structured genetic algorithm. In: Proceedings of the International Conference on Combinations of Genetic Algorithms and Neural Networks. 1992, 87–96
131. Stanley K O, Miikkulainen R. Efficient reinforcement learning through evolving neural network topologies. In: Proceedings of the 2002 Conference on Genetic and Evolutionary Computation. 2002, 569–577
132. Stanley K O, Miikkulainen R. Evolving neural networks through augmenting topologies. Evolutionary Computation, 2002, 10(2): 99–127
133. Singh S P, Sutton R S. Reinforcement learning with replacing eligibility traces. Machine Learning, 1996, 22(1–3): 123–158
134. Whiteson S, Stone P. Sample-efficient evolutionary function approximation for reinforcement learning. In: Proceedings of the 21st AAAI Conference on Artificial Intelligence. 2006, 518–523
135. Whiteson S, Stone P. Evolutionary function approximation for reinforcement learning. Journal of Machine Learning Research, 2006, 7: 877–917
136. Kohl N, Miikkulainen R. Evolving neural networks for strategic decision-making problems. Neural Networks, 2009, 22(3): 326–337
137. Gauci J, Stanley K O. A case study on the critical role of geometric regularity in machine learning. In: Proceedings of the 23rd AAAI Conference on Artificial Intelligence. 2008, 628–633
138. Hausknecht M J, Khandelwal P, Miikkulainen R, Stone P. HyperNEAT-GGP: a hyperNEAT-based Atari general game player. In: Proceedings of the 2012 Conference on Genetic and Evolutionary Computation. 2012, 217–224
139. Ebrahimi S, Rohrbach A, Darrell T. Gradient-free policy architecture search and adaptation. In: Proceedings of the 1st Conference on Robot Learning. 2017, 505–514
140. Zoph B, Le Q V. Neural architecture search with reinforcement learning. In: Proceedings of the 5th International Conference on Learning Representations. 2017
141. Gaier A, Ha D. Weight agnostic neural networks. In: Proceedings of the 31st International Conference on Neural Information Processing Systems. 2019, 5365–5379
142. Conti E, Madhavan V, Such F P, Lehman J, Stanley K O, Clune J. Improving exploration in evolution strategies for deep reinforcement learning via a population of novelty-seeking agents. In: Proceedings of the 31st International Conference on Neural Information Processing Systems. 2018, 5032–5043
143. Chen X H, Yu Y. Reinforcement learning with derivative-free exploration. In: Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems. 2019, 1880–1882
144. Lillicrap T P, Hunt J J, Pritzel A, Heess N, Erez T, Tassa Y, Silver D, Wierstra D. Continuous control with deep reinforcement learning. In: Proceedings of the 4th International Conference on Learning Representations. 2016
145. Vemula A, Sun W, Bagnell J A. Contrasting exploration in parameter and action space: a zeroth-order optimization perspective. In: Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics. 2019, 2926–2935
146. Colas C, Sigaud O, Oudeyer P Y. GEP-PG: decoupling exploration and exploitation in deep reinforcement learning algorithms. In: Proceedings of the 35th International Conference on Machine Learning. 2018, 1038–1047
147. Liu Y R, Hu Y Q, Qian H, Yu Y, Qian C. ZOOpt: toolbox for derivative-free optimization. 2017, arXiv preprint arXiv:1801.00329
148. Jaderberg M, Dalibard V, Osindero S, Czarnecki W M, Donahue J, Razavi A, Vinyals O, Green T, Dunning I, Simonyan K, Fernando C, Kavukcuoglu K. Population based training of neural networks. 2017, arXiv preprint arXiv:1711.09846
149. Beattie C, Leibo J Z, Teplyashin D, Ward T, Wainwright M, Küttler H, Lefrancq A, Green S, Valdés V, Sadik A, Schrittwieser J, Anderson K, York S, Cant M, Cain A, Bolton A, Gaffney S, King H, Hassabis D, Legg S, Petersen S. DeepMind Lab. 2016, arXiv preprint arXiv:1612.03801
150. Vinyals O, Ewalds T, Bartunov S, Georgiev P, Vezhnevets A S, Yeo M, Makhzani A, Küttler H, Agapiou J P, Schrittwieser J, Quan J, Gaffney S, Petersen S, Simonyan K, Schaul T, Hasselt v H, Silver D, Lillicrap T P, Calderone K, Keet P, Brunasso A, Lawrence D, Ekermo A, Repp J, Tsing R. StarCraft II: a new challenge for reinforcement learning. 2017, arXiv preprint arXiv:1708.04782
151. Moritz P, Nishihara R, Wang S, Tumanov A, Liaw R, Liang E, Paul W, Jordan M I, Stoica I. Ray: a distributed framework for emerging AI applications. In: Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation. 2018, 561–577
152. Elfwing S, Uchibe E, Doya K. Online meta-learning by parallel algorithm competition. In: Proceedings of the 2018 Conference on Genetic and Evolutionary Computation. 2018, 426–433
153. Baker J E. Reducing bias and inefficiency in the selection algorithm. In: Proceedings of the 2nd International Conference on Genetic Algorithms. 1987, 14–21
154. Jaderberg M, Czarnecki W M, Dunning I, Marris L, Lever G, Castaneda A G, Beattie C, Rabinowitz N C, Morcos A S, Ruderman A, Sonnerat N, Green T, Deason L, Leibo J Z, Silver D, Hassabis D, Kavukcuoglu K, Graepel T. Human-level performance in 3d multiplayer games with population-based reinforcement learning. Science, 2019, 364(6443): 859–865
155. Jung W, Park G, Sung Y. Population-guided parallel policy search for reinforcement learning. In: Proceedings of the 8th International Conference on Learning Representations. 2020
156. Pourchot A, Perrin N, Sigaud O. Importance mixing: Improving sample reuse in evolutionary policy search methods. 2018, arXiv preprint arXiv:1808.05832
157. Stork J, Zaefferer M, Bartz-Beielstein T, Eiben A E. Surrogate models for enhancing the efficiency of neuroevolution in reinforcement learning. In: Proceedings of the 2019 Conference on Genetic and Evolutionary Computation. 2019, 934–942
158. Bibi A, Bergou E H, Sener O, Ghanem B, Richtárik P. A stochastic derivative-free optimization method with importance sampling: theory and learning to control. In: Proceedings of the 34th AAAI Conference on Artificial Intelligence. 2020, 3275–3282

159. Chen X, Liu S, Xu K, Li X, Lin X, Hong M, Cox D D. ZO-AdaMM: aeroth-order adaptive momentum method for black-box optimization. In: Proceedings of the 32nd International Conference on Neural Information Processing Systems. 2019, 7202–7213
160. Gorbunov E A, Bibi A, Sener O, Bergou E H, Richtárik P. A stochastic derivative free optimization method with momentum. In: Proceedings of the 8th International Conference on Learning Representations. 2020
161. Kandasamy K, Schneider J, Poczos B. High dimensional Bayesian optimisation and bandits via additive models. In: Proceedings of the 32nd International Conference on Machine Learning. 2015, 295–304
162. Wang Z, Zoghi M, Hutter F, Matheson D, Freitas N D. Bayesian optimization in a billion dimensions via random embeddings. Journal of Artificial Intelligence Research, 2016, 55: 361–387
163. Qian H, Hu Y Q, Yu Y. Derivative-free optimization of high-dimensional non-convex functions by sequential random embeddings. In: Proceedings of the 25th International Joint Conference on Artificial Intelligence. 2016, 1946–1952
164. Yang P, Tang K, Yao X. Turning high-dimensional optimization into computationally expensive optimization. IEEE Transactions on Evolutionary Computation, 2018, 22(1): 143–156
165. Mutny M, Krause A. Efficient high dimensional Bayesian optimization with additivity and quadrature fourier features. In: Proceedings of the 31st International Conference on Neural Information Processing Systems. 2018, 9019–9030
166. Müller N, Glasmachers T. Challenges in high-dimensional reinforcement learning with evolution strategies. In: Proceedings of the 15th International Conference on Parallel Problem Solving from Nature. 2018, 411–423
167. Li Z, Zhang Q, Lin X, Zhen H L. Fast covariance matrix adaptation for large-scale black-box optimization. IEEE Transaction on Cybernetics, 2020, 50(5): 2073–2083
168. Wang H, Qian H, Yu Y. Noisy derivative-free optimization with value suppression. In: Proceedings of the 32nd AAAI Conference on Artificial Intelligence. 2018, 1447–1454

Hong Qian received the BSc degree in the School of Mathematical Sciences from Soochow University, China in 2013. He is currently a PhD candidate in the Department of Computer Science and Technology, Nanjing University, China. His research interests are derivative-free optimization and its applications to machine learning. He has served as a reviewer of several world-class journals and a program committee member of several leading international conferences in the field of artificial intelligence.

Yang Yu received the BSc and PhD degrees in Computer Science from Nanjing University, China in 2004 and 2011, respectively. He joined the Department of Computer Science and Technology at Nanjing University as an assistant researcher in 2011, and is currently a professor in the School of Artificial Intelligence at Nanjing University, China. His research interests are in artificial intelligence, including reinforcement learning, machine learning, and derivative-free optimization. He has published over 40 papers in leading international journals and conferences, including *Artificial Intelligence*, IJCAI, AAAI, NeurIPS, KDD, etc. He has been granted several conference best paper awards including IDEAL'16, GECCO'11 (theory track), PAKDD'08, etc. He was in the Champion Team of 2018 OpenAI RetroContest, and the Grand Champion Team of PAKDD 2006 Data Mining Competition. He was recognized as one of the "AI's 10 to Watch" by IEEE Intelligent Systems in 2018, and received the PAKDD Early Career Award in 2018. He was invited to give an Early Career Spotlight Talk in IJCAI'18. He has served as an Area Chair of AAAI'19 and IJCAI'18; a Senior PC member of IJCAI'15/17; a Publicity Co-chair of IJCAI'16/17 and IEEE ICDM'16; a Workshop Co-chair of ACML'16 and PRICAI'18.