Deep Reinforcement Learning Versus Evolution Strategies: A Comparative Survey

Amjad Yousef Majid, Serge Saaybi, Vincent Francois-Lavet, R. Venkatesha Prasad, and Chris Verhoeven, *Member, IEEE*

***Abstract*— Deep reinforcement learning (DRL) and evolution strategies (ESs) have surpassed human-level control in many sequential decision-making problems, yet many open challenges still exist. To get insights into the strengths and weaknesses of DRL versus ESs, an analysis of their respective capabilities and limitations is provided. After presenting their fundamental concepts and algorithms, a comparison is provided on key aspects, such as scalability, exploration, adaptation to dynamic environments, and multiagent learning. Current research challenges are also discussed, including sample efficiency, exploration versus exploitation, dealing with sparse rewards, and learning to plan. Then, the benefits of hybrid algorithms that combine DRL and ESs are highlighted.**

***Index Terms*— Deep reinforcement learning (DRL), evolution strategies (ESs), exploration, meta-learning, multiagent, parallelism.**

I. Introduction

IN THE biological world, the intellectual capabilities of humans and animals have developed through a combination of evolution and learning. On the one hand, evolution has allowed living beings to improve genetically over successive generations such that higher forms of intelligence have appeared, on the other hand, adapting rapidly to new situations is possible due to the learning capability of animals and humans.

In the aim of developing artificial (general) intelligence, these two phenomena have motivated the development of distinct approaches that could both play an important role in the quest for intelligent machines. From the learning perspective, *reinforcement learning* (RL) shows many parallels with how humans and animals can deal with new unknown sequential decision-making tasks. Meanwhile, *evolution strategies* (ESs) are engineering methods inspired by the mechanism that let intelligence emerge in the biological world—repeatedly selecting the best-performing individuals.

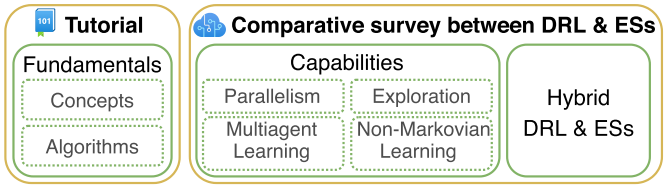

Fig. 1. Structure of the survey.

The RL framework is formalized as an agent acting on an environment with the goal of maximizing a cumulative reward that the agent obtains from the environment [1], [2]. To highlight the main elements of this framework, let us consider the following example. Imagine playing a table tennis game (environment) with a robot (agent). The robot has not explicitly been programmed to play the game but it can instead observe the position of all the key objects, such as the ball (environment state), and has also access to the score of the game (reward). The robot's goal is to maximize its score. For that, it tries different techniques of hitting the ball (through a sequence of actions), observes the outcome, and gradually enhances its playing strategy (policy). The main limitation of classical RL algorithms is the difficulty of generalizing to new slightly different situations (e.g., images with slightly different lighting conditions). To tackle such data, RL algorithms are nowadays often combined with deep neural networks (DNNs), giving rise to a whole new field of research known as deep RL (DRL) that can automatically learn from high-dimensional observations [3].

Instead of improving one candidate solution, ESs consider a population of solutions. They iteratively generate candidate solutions, evaluate them, and bias the search toward the best-scoring ones [4]. The advantage of this approach is its simplicity and minimum requirements on the optimization side. The disadvantage is its high computational demand and the difficulty of learning from high-dimensional data.

The parallel development of DRL and ESs indicates that each has its advantages (and disadvantages), depending on the problem setup. To enable scientists and researchers to choose the best algorithm for the problem at hand, we summarized the pros and cons of these approaches through the development of a comparative survey: we compared DRL and ESs from different learning aspects such as scalability, exploration,

Manuscript received 13 October 2021; revised 5 August 2022 and 2 January 2023; accepted 25 March 2023. This work was supported in part by Cognizant Technology Solutions through the Internet of Swarms Project and in part by Rijksdienst voor Ondernemend Nederland under PPS O&I. *(Corresponding author: Amjad Yousef Majid.)*

Amjad Yousef Majid and Chris Verhoeven are with the Department of Microelectronics, Delft University of Technology, 2628 Delft, The Netherlands (e-mail: a.y.majid@tudelft.nl; c.j.m.verhoeven@tudelft.nl).

Serge Saaybi and R. Venkatesha Prasad are with the Department of Software Technology, Delft University of Technology, 2628 Delft, The Netherlands (e-mail: s.c.e.saaybi@student.tudelft.nl; r.r.venkateshaprasad@tudelft.nl).

Vincent Francois-Lavet is with the Department of Computer Science, Vrije Universiteit Amsterdam, 1081 Amsterdam, The Netherlands (e-mail: vincent.francoislavet@vu.nl).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TNNLS.2023.3264540.

Digital Object Identifier 10.1109/TNNLS.2023.3264540

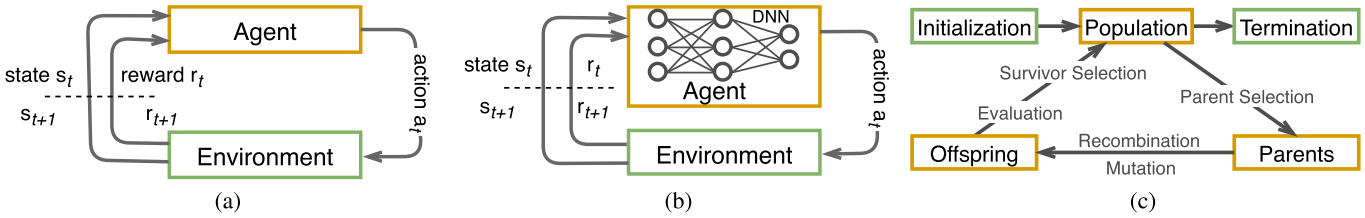

Fig. 2. Iteration loops of (Deep) RL and evolutionary strategies. (a) RL. (b) DRL. (c) ESs.

and the ability to learn in dynamic environments (Fig. 1). Additionally, we discuss how combining DRL and ESs in hybrid systems can leverage the advantages of both approaches.

To date, there have been different papers reviewing some aspects of DRL and ESs. For example, derivative-free RL and ESs are reviewed in [5], covering aspects, such as scalability and exploration, and systems that hybridize DRL and ESs are surveyed in [6]. However, different from prior work, this article surveys the literature with a bird's-eye view, focusing on the main developmental directions instead of individual algorithms.

The rest of this survey is organized as follows: Section II presents the fundamental architectural concepts and algorithms behind RL and ESs; Sections III-A–III-D compare the capabilities of DRL and ESs; and in Section IV, hybrid systems, which combine DRL and ESs, are presented. Section V outlines open challenges and potential research directions. Finally, we conclude this survey in Section VI. The main takeaways of each section are summarized in a concise subsection titled "observations."

II. Fundamentals

Here, we cover the fundamentals of DRL and ESs, including formal definitions and the main algorithmic families. In addition, non-Markovian and multiagent settings are introduced.

A. Reinforcement Learning

RL is a computational approach to understanding and automating goal-directed learning and decision-making [1]. The goal of an RL agent is to maximize the total reward it receives over a trajectory of interaction with the environment [Fig. 2(a)]. Generally, this interaction is modeled as a Markov decision process (MDP). An MDP is defined by the tuple $\langle \mathcal{S}, \mathcal{A}, T, R, \rho_0 \rangle$, where $\mathcal{S}$ denotes the state space, $\mathcal{A}$ is the action space, $T(s, a, s')$ is a transition function that defines the probability of transitioning from the current state s to the next state s' after an agent takes action a, $R(s, a, s')$ is the reward function that defines the immediate reward r that the agent observes after taking action a and the environment transition from s to s', and ρ_0 is the starting state distribution. The accumulation of the rewards r starting from time t until the end of the interaction is referred to as the *return*. Mathematically, the return is expressed as

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

where R_t is a random variable that models the rewards r and $\gamma \in (0, 1]$ is a discount factor that weights the immediate and future rewards. It also prevents G_t from approaching infinity when the agent-environment interaction has no terminating state.

The state-value function $v^\pi(s)$ gives the expected return of being in state s and following policy π

$$v^\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v^\pi(s')]. \quad (1)$$

The action-value function (or Q-function) $q^\pi(s, a)$ yields the expected return of taking action a in state s and following policy π thereafter

$$q^\pi(s, a) = \sum_{s',r} p(s', r|s, a) \left[r + \gamma \sum_{a'} \pi(a'|s') q^\pi(s', a') \right]. \quad (2)$$

Value-based RL refers to a family of RL algorithms that optimize value functions and use them to select the most rewarding actions. Examples of such algorithms are state-action-reward-state-action (SARSA) [7] and Q-learning [1]. In the classical RL settings, these algorithms represent the value functions as tables which make them unsuitable for problems with large state and/or action spaces. To overcome this limitation DRL uses DNNs to find low-dimensional representations of large state spaces [Fig. 2(b)] [3]. Therefore, algorithms like deep SARSA [8] and deep Q-network (DQN) [9] can learn directly from high-dimensional data, such as images.

Instead of optimizing a value function to select optimal actions, *policy-based RL* algorithms optimize the policy directly. A policy, in the most general form, is a function that outputs a probability distribution over the action space given a state, $\pi(a|s)$. An optimal policy, $\pi^*(a|s)$, is the one that yields the maximum return from any state. Policy gradient algorithms [10] represent the workhorse of policy-based RL. Compared with value-based algorithms policy-based ones are better suited for learning stochastic policies and for dealing with high-dimensional or continuous action spaces. However, naive policy-based algorithms have high variance in estimating the gradient and are sample inefficient.

The idea of learning a policy and value function to reduce the variance and speed up learning has led to the emergence of the actor–critic algorithms [1]. The actor refers to a parameterized DNN that acts as a policy, and the critic refers to a DNN that estimates the value of the taken action (the Q value). Examples of actor–critic algorithms include advantage actor–critic (A2C) [11], asynchronous advantage

actor–critic (A3C) [12], and deep deterministic policy gradient (DDPG) [13].

In addition to learning a value function and/or policy, an algorithm may also learn a model of the environment. Thanks to the model, an RL agent can predict the outcome of future interactions, and thereby, substantially improves its sampling efficiency. Dyna-Q [14] is a classical model-based RL algorithm.

1) SARSA: is a model-free RL algorithm that optimizes the state-action values (Q values) to select (near) optimal actions. It updates the Q values using

$$Q(s,a) \leftarrow Q(s,a) + \alpha\big[r + \gamma Q(s',a') - Q(s,a)\big] \quad (3)$$

where α is the learning rate. This update rule shows that SARSA is a temporal difference (TD) learning method. That is, it updates a Q value by bootstrapping from the current Q value estimate. To select an action, SARSA uses a policy that relies on the Q values, such as the ε-greedy algorithm. ε-greedy selection chooses an action with the maximum Q value with a probability of $1-\varepsilon$, and otherwise, an action is selected uniformly from $\mathcal{A}$. SARSA is an on-policy algorithm because its behavior policy, i.e., the policy that causes the environment to transition, is also the one estimated by the Q values.

Q-learning [1] is similar to SARSA, but it is an off-policy algorithm, which means that its behavior and target policies are different. In particular, the Q-learning's update rule selects the action from the next state with the maximum Q value which is not necessarily the one chosen by ε-greedy as in SARSA. The update rule of Q-learning is

$$Q(s,a) \leftarrow Q(s,a) + \alpha\big[r + \gamma \max_a Q(s',a') - Q(s,a)\big]. \quad (4)$$

Off-policy algorithms are more data efficient than on-policy ones because they can reuse old data repeatedly.

DQN [15] combines Q-learning with a convolutional neural network (CNN) to act in environments with high-dimensional input spaces (e.g., images of Atari games). The CNN gets a state (e.g., a mini-batch of images) as input and produces Q values of all possible actions. Then, an ε-greedy algorithm is used to select an action. Using a DNN to approximate an optimal value function causes the DRL agent to be unstable [15]. DQN does two things to improve the stability of the agent: first, it samples from an experience replay buffer (i.e., a buffer of $D = \{(s,a,r,s'),(s',a',r',s''),\dots\}$) to reduce the correlation in the training data [16], and second, it uses a target network (a second DNN) that is updated only after a certain number of iterations to further stabilize learning. To optimize the DNN parameters, DQN uses stochastic gradient descent with the following loss function:

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim U(D)} \times \left[\left(r + \gamma \max_{a'} Q(s',a';\overline{\theta_i}) - Q(s,a;\boldsymbol{\theta}_i)\right)^2\right] \quad (5)$$

where θ_i and $\overline{\theta_i}$ are the parameters of the Q-network and target network, respectively; and the experiences, (s,a,r,s'), are drawn from D uniformly.

REINFORCE [17] is a stochastic gradient-based algorithm that directly optimizes a parameterized policy, $\pi(a|s,\boldsymbol{\theta})$. Such a policy can be represented by a neural network where the policy's parameters are the network weights. Thanks to the gradient policy theorem, we can ensure changing the weights (and thereby the policy) in the direction that leads to performance improvement [1]. For episodic REINFORCE, the performance can be defined as the value of the starting state. In the continuing settings, the average rate of reward per time step is the preferred performance metric. The episodic REINFORCE uses stochastic gradient ascent (SGA) and the following rule:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \gamma^t \alpha G \nabla \ln \pi(a|s,\boldsymbol{\theta})$$

to update the policy parameters, where α is a scalar that controls the updating rate. Although this update rule converges to a local minimum, it has high variance as it relies on an empirical return estimate. Consequently, this version of REINFORCE learns slowly. One simple but effective idea to reduce the variance is to subtract the return of a trajectory from a baseline. To reduce the variance and thereby speed up learning the return is subtracted from a baseline ($b(s)$), which results in the REINFORCE-with-baseline update rule

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\gamma^t(G - b(s))\nabla \ln \pi(a|s,\boldsymbol{\theta}).$$

Some of the used baselines include the average of return values and the state-value function [11], [17]. If the state-value function estimate is learned in a temporal-difference (TD) manner, then the algorithm becomes known as actor–critic in the literature.

Dyna-Q [1], [14] uses experiences of interaction between an agent and environment to learn a model of the environment's dynamics. Then, to improve its Q value estimates, it mixes simulated experiences produced by the learned model and actual experiences collected from interaction with the environments. Its update rule is similar to that of the Q-learning algorithm. Interacting with an environment's model called planning. Planning can speed up learning significantly, but learning an accurate model can be challenging.

B. Evolution Strategies

ESs are set of a population-based black-box optimization algorithms often applied to continuous search spaces problems to find the optimal solutions [18], [19]. ESs do not require modeling the problem as an MDP, and neither does the objective function $f(\boldsymbol{x})$ have to be differentiable and continuous. The latter explains why ESs are gradient-free optimization techniques. They do, however, require the objective function $f(\boldsymbol{x})$ to be able to assign a fitness value to (i.e., to evaluate) each input $\boldsymbol{x} \in \mathbb{R}^n$ such that $f : \mathbb{R}^n \to \mathbb{R}$, $\boldsymbol{x} \to f(\boldsymbol{x})$.

The basic idea behind ESs is to bias the sampling process of candidate solutions toward the best individuals found so far until a satisfactory solution is found. Samples can be drawn, for instance, from a (multivariate) normal distribution whose shape is described by what is called *strategic parameters*, i.e., the mean m and the standard deviation σ. These can be modified online to make the search process more efficient.

The generic ESs process is shown in Fig. 2(c) and its elements are explained in the following.

1) *Initialization:* The algorithm generates an initial population P consisting of μ individuals.
2) *Parent Selection:* A subset of the population is selected to function as parents during the recombination step.
3) *Reproduction:* It consists of two steps.
   a) *Recombination:* Two or more parents are combined to produce a mean for the new generation.
   b) *Mutation:* A small amount of noise—usually normally distributed—is added to the recombination results

$$\boldsymbol{x}_k^{g+1} \sim \mathcal{N}(\boldsymbol{m}^{(g)}, \sigma^{(g)} I) = \boldsymbol{m}^{(g)} + \sigma^{(g)} \mathcal{N}(0, I)$$

   where g is the generation index, k is the number of offsprings, and I is the identity matrix.
4) *Evaluation:* A fitness value is assigned to each candidate solution using the objective function $f(x_i)$.
5) *Survivor Selection:* The best μ individuals are selected to form the population for the next generation. Generally, the algorithm iterates from step 2 to step 5 until a satisfactory solution is found.

Employing ESs as an alternative to RL is not new [20], [21], [22], [23]. However, recent advances in computing power and parallelization have reignited the interest in ESs [24], [25].

1) Fundamental Algorithms: The *(1 + 1)-ES* (one parent, one offspring) is the simplest ES conceived by Rechenberg [26]. First, a parent candidate solution, $\mathbf{x}_p$, is drawn according to a uniform random distribution from an initial set of solutions, $\{\mathbf{x}_i, \mathbf{x}_j\}$. The selected parent, $\mathbf{x}_p$, together with its fitness values enter the evolution loop. In each generation (or iteration), an offspring candidate solution, $\mathbf{x}_o$, is created by adding a vector drawn from an uncorrelated multivariate normal distribution to $\mathbf{x}_p$ as follows:

$$\mathbf{x}_o = \mathbf{x}_p + \mathbf{y}\sigma, \quad \mathbf{y} \sim \mathcal{N}(0, \mathbf{I}).$$

If the offspring $\mathbf{x}_o$ is found to be fitter than the parent $\mathbf{x}_p$, then it becomes the new parent for the next generation; otherwise, it is discarded. This process is repeated until a termination condition is met. The amount of mutation (or perturbation) added to $\mathbf{x}_p$ is controlled by the stepsize parameter σ. The value of σ is updated every predefined number of iterations according to the well-known (1/5th) success rule [27], [28]: if $\mathbf{x}_o$ is fitter than $\mathbf{x}_p$ (1/5th) of the times, then σ should stay the same; if $\mathbf{x}_o$ is fitter *more* than (1/5th) of the times, then σ should be increased, and otherwise, it should be decreased.

The $(\mu/\rho^+\lambda)$-ES was originally proposed by Schwefel [29] as an extension to the $(1 + 1)$-ES. Instead of using one parent to generate one offspring, it uses μ parents to generate λ offsprings using both recombination and mutation. In the comma-variation of this algorithm [i.e., $(\mu/\rho,\lambda)$-ES], the selection of the parents happens solely from the offsprings; whereas in the plus-variation, the selection of the parents for the next generation happens from the union of the offsprings and old parents. ρ refers to the number of parents used to generate each offspring.

An element (or an individual) that the $(\mu/\rho^+\lambda)$-ES evolves consists of $(\mathbf{x}, \mathbf{s}, f)$, where $\mathbf{x}$ is the candidate solution, $\mathbf{s}$ are the strategy parameters that control the significance of the mutation, and f holds the fitness value of $\mathbf{x}$. Consequently, the evolution process itself tunes the strategy parameters, which is known as self-adaptation. Thus, unlike $(1 + 1)$-ES, $(\mu/\rho^+\lambda)$ do not need external control settings to adjust the strategy parameters.

Covariance Matrix Adaptation ESs (CMA-ESs) is one of the most popular gradient-free optimization algorithms [30], [31], [32], [33]. To search a solution space, it samples a population, λ, of new search points (offsprings) from a multivariate normal distribution

$$\boldsymbol{x}_i^{g+1} = \boldsymbol{m}^{(g)} + \sigma^{(g)} \mathcal{N}(\mathbf{0}, \boldsymbol{C}^{(g)}), \quad \text{for } i = 1, \ldots, \lambda$$

where g is the generation number (i.e., $g = 1, 2, 3, \ldots$), $\boldsymbol{x}_i \in \mathbb{R}^n$ is the ith offspring, $\boldsymbol{m}$ and σ denote the mean and standard deviation of $\boldsymbol{x}$, $\boldsymbol{C}$ represents the covariance matrix, and $\mathcal{N}(\mathbf{0}, \boldsymbol{C})$ is a multivariate normal distribution. To compute the mean for the next generation, $\boldsymbol{m}^{g+1}$, CMA-ES computes a weighted average of the best—according to their fitness values—μ candidate solutions, where $\mu < \lambda$ represents the parent population size. Through this selection and the assigned weights, CMA-ES biases the computed mean toward the best candidate solutions of the current population. It automatically adapts the stepsize σ (the mutation strength) using the cumulative stepsize adaption (CSA) algorithm [30] and an evolution path, $\boldsymbol{p}_\sigma$: if $\boldsymbol{p}_\sigma$ is longer than the expected length of the evolution path under random selection $\mathbb{E}||\mathcal{N}(\mathbf{0}, \boldsymbol{I})||$, then increase the stepsize; otherwise, decrease it. To direct the search toward promising directions, CMA-ES updates the covariance matrix in each iteration. The update consists of two main parts: 1) rank-1 update, which computes an evolution path for the mutation distribution means, similar to the stepsize evolution path and 2) rank-μ update, which computes a covariance matrix as a weighted sum of covariances of the best μ individuals. The obtained results from these steps are used to update the covariance matrix $\boldsymbol{C}$ itself. The algorithm iterates until a satisfactory solution is found (we refer the interested reader to [33] for a more detailed explanation).

Natural Evolution Strategies: NESs are a family of black-box optimization algorithms that use the natural gradient to update the search distribution in the direction of the highest fitness [34]. Relying on natural gradient—instead of plain gradient—makes NES resistant to premature convergence, oscillations, and unwanted side effects of a particular parameterization. NES main steps are as follows.

1) *Sampling:* NES samples its individuals from a probability (Gaussian) distribution over the search space. Its goal is to update the distribution parameters $\boldsymbol{\theta}$ to maximize the average fitness $f(\boldsymbol{x})$ of the sampled individuals $\boldsymbol{x}$.
2) *Search Gradient Estimation:* NES estimates a search gradient on the parameters by evaluating the samples previously computed. It then decides on the best direction to take to achieve a higher expected fitness.
3) *Gradient Ascent:* NES computes gradient ascent along the estimated gradient.
4) Back to 1) unless a stopping criterion is met [34].

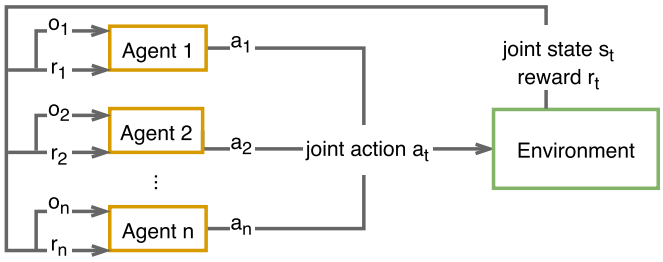

Fig. 3. Multiagent RL overview [46].

C. Non-Markovian Settings

The Markov property denotes the situation where the future states of a process depend only on the current state and not on events or states from the past. In many real-world applications, agents can only partially observe the state of their environments. This means agents need to take into account the history of observations—actions and rewards—to produce a better estimation of the underlying hidden state [35], [36], [37]. These problems are usually modeled as a partially observable MDP (POMDP). Researchers have addressed the POMDP problem setup through the proposal of many RL models [38] and ESs [39]. For example, Hausknecht and Stone [40], Igel [41] employed a neural network with a recurrent architecture to consider past observations.

Another common scenario where an agent does not have perfect information is when it faces a distribution of environments. This setting is known as Meta-RL. The objective of meta-RL is to learn a policy that can be quickly generalized across a distribution of tasks or environments. Generally, a meta-learner achieves this through a two stages optimization process: first, a meta-policy is trained on a distribution of tasks to learn their common dynamics; then, the meta-learner tailors the learned meta-policy to a particular task through the interaction with the given environment [42]. Examples of meta-RL tasks include: solving different mazes [43], dealing with component failures [44], or driving different cars [45].

D. Multiagent Learning

A multiagent system (MAS) refers to multiple cooperating or competing agents working toward maximizing their own objectives in a shared environment [47]. Since the environment and reward are affected by the joint actions of all agents, the single-agent MDP model cannot be directly applied to MAS settings (Fig. 3). The Markov games (MGs) [48] framework comes as a generalization of the MDP that captures the entanglement of the multiple agents. MG defines two distinct types of games: cooperative and competitive.

In a *cooperative game* agents seek to maximize a common reward signal by taking actions that favor their outcome while taking into account their effects on others. Most contemporary applications are based upon a cooperative setup. Examples include traffic signal control [49], [50], robot path finding [51], [52], and air traffic control [53]. On the other hand, agents in a *competitive game* receive different rewards based on the overall outcome of the joint actions. In this setup, certain actions might be beneficial to one set of agents while being disadvantageous to others.

Another important aspect of an MAS is the *centralized or decentralized* control approach. In a centralized control, a single-entity governs the decisions of all agents based on all available joint actions, rewards, and observations [54]. While this approach enables optimal decisions, it quickly becomes computationally intractable as the number of agents grows [55]. In decentralized settings, agents make decisions independently, based on local information. Decentralized systems can be subdivided into two categories: "decentralized networked agents," and "fully decentralized agents" [56]. In the former, the agents communicate with each other to optimize their actions. In the latter, agents make independent decisions without information exchange. While this means that no explicit messages can be sent, it is still possible to influence other agents by affecting their reward [51]. While the decentralized approach is more scalability and robustness, it increases the complexity of the system significantly.

E. Observations

Our main takeaways from this section are as follows.

1) An ES algorithm is a black-box optimization method that keeps a pool of multiple candidate solutions, while an RL method generally has a single agent that improves its policy by interacting with its environment.
2) RL can naturally be combined with deep learning and gradient descent techniques (with backpropagation), which is key for sample efficiency (i.e., generalization).
3) Value-based RL methods usually operate in discrete action spaces, while the actor–critic architecture extends this ability to continuous action spaces. ESs can operate on discrete or continuous action spaces by default.
4) Both ESs and on-policy RL algorithms are data inefficient: on-policy algorithms make use of data that are generated from the current policy and discard older data and ESs discard all but a subset of candidate solutions in each iteration.
5) Value-based off-policy DRL algorithms, such as DQN, can be data efficient because they can work with a replay memory that allows the reuse of off-policy data. However, they can become unstable for long horizons and high discount factors [57].
6) In multiagent settings, both centralized and decentralized MASs have their pros and cons. A semicentralized MAS may surpass both. Learning in such settings is a promising research direction.

III. Deep Reinforcement Learning Versus Evolution Strategies

This section compares different aspects of DRL and ESs, such as their ability to parallelize computations, explore environments, and learn in multiagent and dynamic settings.

A. Parallelism

Despite the success of DRL and ESs, they are still computationally intensive approaches to tackling sequential

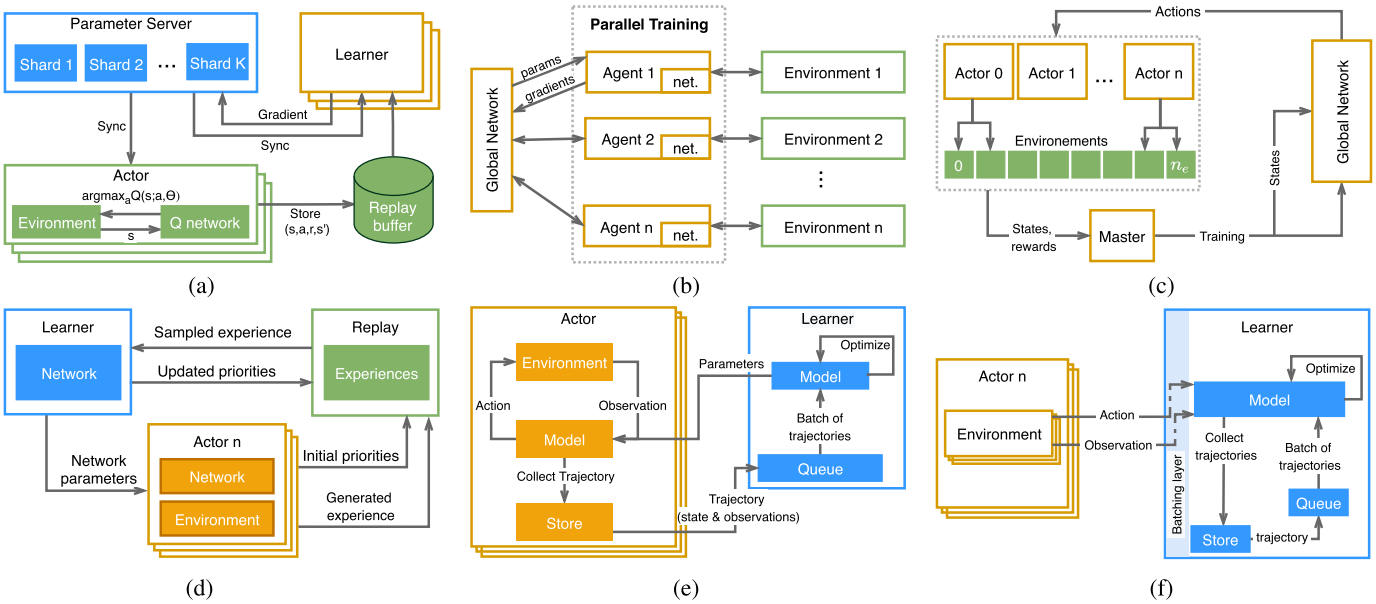

Fig. 4. Parallel DRL algorithms architectures. (a) Gorila [58]. (b) A3C [12]. (c) Batched A2C [11]. (d) Ape-X [59]. (e) IMPALA [60]. (f) SEED [61].

decision-making problems. Parallel execution is, thus, an important approach to speed up the computation [58].

1) Parallelism in Deep Reinforcement Learning: In parallel-DRL, many agents (or actors) run in parallel to accelerate the learning process. Each actor gathers its own learning experiences. These experiences are, then, shared to optimize a global network (Fig. 4) [62], [63]. The rest of this section presents important parallel-DRL algorithms.

Gorila [58] is a large-scale distributed DRL architecture. It can be thought of as distributed DQN with four major components: actors, learners, a parameter server, and a replay memory [Fig. 4(a)]. The actors interact with multiple environment instantiations in parallel and store their experiences (i.e., a set of $\{s, a, r, s'\}$) in the replay memory. Each actor has its own policy (e.g., ε-greedy), and therefore, they generate different trajectories of experiences. Consequently, Gorila explores environments better than DQN, and therefore, it has a better overall performance. Learners sample the experience replay memory, compute gradients, and send them to the parameter server. The parameter server updates its parameters according to the received gradients and then synchronizes the learning process by updating the actors' and learners' Q-networks.

a) A3C and GA3C: While using a replay buffer helps in stabilizing the learning process, it requires additional memory, adds computational overhead, and can only be used with off-policy algorithms. Motivated by these limitations, Mnih et al. [12] introduced the A3C as an alternative to Gorila. A3C consists of a global network and multiple workers (or agents) interacting with independent instances of an environment [Fig. 4(b)]. The agents are implemented as CPU threads within a single machine, which reduces the communication cost imposed by distributed systems, such as Gorila. Each agent generates a trajectory of experiences and calculates its gradient. These gradients are sent to the global network to update its parameters. Then, the workers obtain a copy of the global network parameters to update theirs. This asynchronous way of learning diversifies and decorrelates data updates, which stabilizes the learning process. GA3C [64] makes use of GPUs and shows better scalability and performance than A3C.

b) Batched A2C and DPPO: A downside of A3C is that asynchronous updates may lead to suboptimal collective updates to the global network. To overcome this, batched advantage actor–critic (batched A2C) employs a master node to synchronize the update process of the global network [11]. Furthermore, batched A2C architecture maintains the advantages of both Gorila and A3C. Similar to Gorila, batched A2C runs on GPUs and the number of actors is highly scalable while still running on a single-machine akin to A3C and GA3C [64]. At each step, batched A2C samples from the policy and generates a batch of actions for n_w workers on n_e environment instances [Fig. 4(c)]. The resulting experiences are then stored and used by the master to update the policy (global network). The batched approach allows for easy parallelization by synchronously updating a unique copy of the parameters, with the drawback of higher communication costs. Distributed proximal policy optimization (DPPO) [65] features architecture similar to that of A2C but uses the PPO [66] algorithm for learning.

c) Ape-X and R2D2: Ape-X [59] extends the prioritized experience buffer to the parallel-DRL settings and shows that this approach is highly scalable. The Ape-X architecture consists of many actors, a single learner, and a prioritized replay buffer [Fig. 4(d)]. Each actor interacts with its instance of the environment, gathers data, and computes its initial priorities. The generated experiences are stored in a shared prioritized buffer. The learner samples the buffer to update its network and the priorities of the experiences in the buffer. In addition, the learner also periodically updates the network parameters of the actors. Ape-X's distributed architecture can be coupled with different learning algorithms, such as DQN [15] and DDPG [13]. R2D2 [67] has a similar architecture but

TABLE I
PARALLELIZED DRL AND ESS SYSTEMS

Algorithms	Architecture	Experiments	Limitations	Ref.
Gorila	replay buffer, actors, learners, and the parameter server each runs on a separate machine; GPU	outperforms DQN in 41/49 Atari games with reduced wall-time	high bandwidth for communicating gradients and parameters	[58] [60]
A3C	many actors each running on a CPU thread and update a global network	outperforms Gorila on the Atari games while training for half the time	possibility of inconsistent parameter updates; large bandwidth between learner and actors; does not make use of hardware accelerators	[12] [60]
Batched A2C	multi-actors, a master, which synchronizes actors' updates, and a global network; GPU	requires less training time as compared to Gorila, A3C, and GA3C	high variance in complex environments limits its performance; episodes of varying length cause a slowdown during initialization	[11]
Ape-X	multi-actors, a shared learner, and prioritized replay memory; CPU/GPU	outperforms Gorila and A3C on the Atari domain with less wall-clock training time	inefficient CPUs usage; large bandwidth for communicating between actors and learner	[59]
IMPALA	multi-actors; single or multiple learners; replay buffer; GPUs	outperforms Batched A2C and A3C. Less sensitive to hyperparameters selection than A3C	uses CPUs for inference which is inefficient; requires large bandwidth for sending parameters and trajectories	[60]
SEED	multi-actors and a single learning; GPU/TPU	surpasses the performance of IMPALA	centralized inference may lead to increase latency	[61]
OpenAI-ES	set of parallel workers; CPUs	shows comparative performance as compared to DRL algorithms	requires 240 CPU hours to train a simulated humanoid to walk	[24]

outperforms Ape-X by using recurrent neural network (RNN)-based RL agents.

d) Importance Weighted Actor-Learner Architecture (IMPALA): Due to the use of an on-policy method in an off-policy setting GA3C [64] suffers from poor convergence. IMPALA [60] corrected this with the use of V-trance: an off-policy actor–critic algorithm that aims at mitigating the effect of the lag between when actions are taken by the actors and when the learner estimates the gradient in distributed settings. IMPALA's architecture consists of multiple actors interacting with their environment instances and single or multiple synchronized learners [Fig. 4(e)]. IMPALA's actors send experiences directly to the learners, instead of computing and sending the gradients like in A3C. The learners then utilize these experiences to optimize their policies and value functions. After that, the learners update the actors' network parameters. The separation between acting and learning and V-trace enable IMPALA to have stable learning while achieving high throughput.

e) Scalable, Efficient Deep-RL (SEED): SEED [61] improves on the IMPALA system by moving inference to the learner [Fig. 4(f)]. Consequently, the trajectories' collection becomes part of the learner and the actors only send observations and actions to the learner. SEED makes use of TUPs and GPUs and shows significant improvement over other approaches.

2) Parallelism in Evolution Strategies: Compared with DRL, ESs require significantly less bandwidth to parallelize a given task. Salimans et al. [24] proposed OpenAI-ES: an algorithm derived from NES (see Section II) that directly optimizes the parameters $\boldsymbol{\theta}$ of a policy. By sharing the seeds of the random processes prior to the optimization process, OpenAI-ES requires exchanging only scalars (minimal bandwidth) between workers to parallelize the search process. The main steps of OpenAI-ES are.

1) Sample a Gaussian noise vector, $\varepsilon_i \sim \mathcal{N}(0, I)$.
2) Evaluate workers' fitness functions, $f_i \leftarrow f(\boldsymbol{\theta}_t, \sigma\varepsilon_i)$.
3) Exchange the fitness values, f_i, between the workers.
4) Reconstruct ε_i using known random seeds.
5) Adjust parameters according to $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t + \alpha(1/n\sigma) \sum_{j=1}^{n} f_j \varepsilon_j$, where $\boldsymbol{\theta}$ is a weighted vector of a DNN.
6) Repeat from step 2 until termination.

The results, relative simplicity, and generality of OpenAI-ES have attracted the attention of many researchers. From a parallelization perspective, the follow-up work indirectly improved OpenAI-ES's parallelization: by searching more intelligently a smaller number of agents (or less time) is needed to match OpenAI-E's performance. For example, Chrabaszcz et al. [25] improved the OpenAI-ES algorithm by complementing it with two techniques: structured exploration and compact policies. The authors showed with these modifications up to 13× fewer parameters are needed.

Conti et al. [68] proposed a novelty search ES (NS-ES) algorithm, which hybridizes OpenAI-ES with NS—an evolutionary algorithm that searches for diverse policies without considering the reward signal [69]. The authors also introduced a variant of NS-ES by replacing NS with a quality diversity (QD) algorithm. Their results show that the NS- and QD-based algorithms improve the ES algorithm's performance on RL tasks with sparse rewards as they help avoid local optima. Liu et al. [70] proposed trust region ESs (TRES). It optimizes a surrogate objective function, which enables repeated data sample usage. Finally, Fuks et al. [71] proposed ES with progressive episode lengths (PELs). The main idea of PEL is to allow an agent to do small and easy tasks to gain knowledge quickly and then use this knowledge to tackle more complex tasks. PEL leverages the same parallelization idea as OpenAI-ES [24] and Table I snapshots the main characteristics of parallel DRL and ESs algorithms, and Fig. 5 shows their chronological order.

B. Exploration

One of the fundamental challenges that a learning agent faces when interacting with a partially known environment is the exploration–exploitation dilemma. That is, when should an agent try out suboptimal actions to improve its estimation of

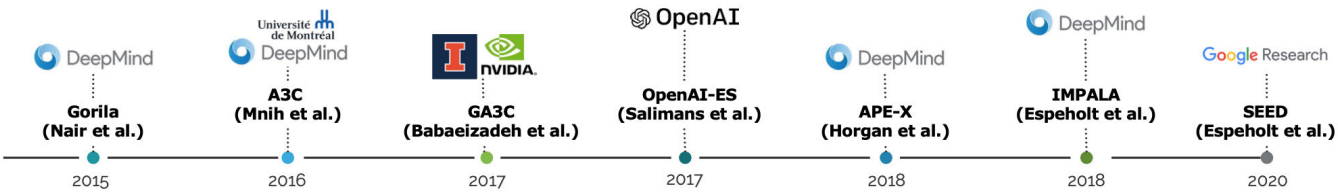

Fig. 5. Parallel DRL and ESs algorithms are shown on a timeline.

the optimal policy, and when should it use its current optimal policy estimation to make useful progress? This dilemma has attracted ample attention. Beyond this tradeoff, advanced exploration techniques are essential for tackling environments with sparse rewards [72], [73]. In a sparse reward setting, a DRL agent gets only occasional feedback for its actions. Consequently, it is hard for the agent to learn action–reward associations to approach optimal policy [72], [74], [75], [76].

1) Exploration in (Deep) Reinforcement Learning: Simple exploration techniques balance exploration and exploitation by selecting estimated optimal actions most of the time and random ones on occasion. For example, the ϵ-greedy exploration algorithm [1] acts greedily with probability $1-\epsilon$ and selects a random action with probability ϵ.

More complex exploration strategies estimate the value of an exploratory action by making use of the environment-agent interaction history. Upper confidence bound (UCB) [77] does that by adding a term to the reward signal that reflects the algorithm uncertainty about moving to a particular state

$$r^{+}(s,a) = r(s,a) + B(N(s))$$

where $N(s)$ represents the frequency of visiting state s, and $B(N(s))$ is a reward bonus decreases with $N(s)$. In other words, UCB promotes the selection of actions with high rewards, $r(s,a)$, or the ones with high uncertainty (less frequently visited). The Thompson sampling (TS) method [78] maintains a distribution over the parameters of a model. In the beginning, it samples parameters at random. But as the agent explores an environment, TS adapts the distribution to favor more promising parameter sets. As such, UCB and TS naturally reduce the probability of selecting exploratory actions and become more confident about optimal policies over time. Thus, they are inherently more efficient than ϵ-greedy.

a) From RL to DRL: DRL agents act on environments with continuous or high-dimensional state-action spaces (e.g., Montezuma's Revenge and StarCraft II). Such spaces render count-based algorithms (e.g., UCB) and the ones that require maintaining a distribution over state-action spaces (e.g., TS) useless in their original formulation. To explore such challenging environments with sparse reward signals, many algorithms have been proposed. Generally, these algorithms couple approximation techniques with exploration algorithms proposed for simple RL settings [79], [80], [81].

b) Pseudocount methods: To extend count-based exploration methods (e.g., UCB) to DRL settings, Bellemare et al. [82] approximate the counting process using a context tree switching (CTS) density model. The model's goal is to provide a score that increases when a state is revisited. The score is then used to generate a reward bonus that is inversely proportional to the score value. This bonus is then added to the reward signal provided by the environment as an incentive for the agent to visit less-visited states. Ostrovski et al. [83] improved this approach by replacing the simple CTS density model with a neural density model called PixelCNN. Another approach to utilize counting to explore environments with high-dimensional spaces is by mapping the observed states to a hashing table [84] and counting the hashing codes instead of states. Finally, Machado et al. [85] used a form of implicit counting via the norm of the successor state representation [86] to generate a reward bonus that encourages the agent to visit less-visited states.

c) Information gain: In exploration based on information gain, the algorithm provides a reward bonus proportional to the information obtained after taking an action. This bonus is then added to the reward provided by the environment to push the agent to explore less known states [87]. Houthooft et al. [88] proposed to learn a transition dynamic model with a Bayesian neural network. The information gain is measured as the KL divergence between the current and updated parameter distribution after a new observation. Based on this information the reward signal is augmented with a bonus. Pathak et al. [89] used a forward dynamic model to predict the next state. The reward bonus is then set to be proportional to the error between the predicted and observed state. To make this method effective, the authors utilized an inverse model, removing irrelevant—for the comparison—state features. Burda et al. [90] defines the exploration bonus based on the error of a neural network in predicting features of the observations given by a fixed randomly initialized neural network. All the aforementioned information gain methods obtain information in a reactive manner: the information gain happens after the interaction with the environment. Shyam et al. [91] proposed an active exploration algorithm that involves learning an ensemble of forward dynamics models. The differences between the models' predictions are used as a metric to measure states' novelties. Then, they explore novel states or states that the models disagree with the most. Pathak et al. [80] showed how to extend the active exploration idea to environments with stochastic dynamics, such as robot manipulators. Having an ensemble of models can lead to "deep exploration" as we will see next.

d) Approximate posterior sampling: Inspired by TS, Osband et al. [92] introduced bootstrapped DQN. Bootstrapped DQN trains a DNN with N bootstrapped heads to approximate a distribution over Q-functions (or DQN). At the start of each episode, bootstrapped DQN draws a sample at

random from the ensemble Q-functions and acts greedily with respect to this sample. This strategy enables an RL agent to do temporally extended exploration (or deep exploration), which is particularly important when the agent receives a sparse environmental reward. Chen et al. [93] integrates UCB with bootstrapped DQN by calculating the mean and variance of a subset of the ensemble Q-functions.

e) Memory-based: Savinov et al. [94] proposed a new curiosity method that uses episodic memory to form the novelty bonus. The bonus is computed by comparing the current observation with the observations in memory and a reward is given for observations that require more environment steps taken to reach. Ecoffet et al. [95] introduced go-explore: an RL agent that aims to solve hard exploration problems, such as Montezuma's revenge. Go-explore runs in two phases. In phase one, the agent explores randomly, remembers interesting states, and continues, after a reset, random exploration from one of the interesting states. Once a solution has been found, phase two begins where the go-explore agent robustifies the best-found solution by randomizing the environment and running imitation learning using the best solution. Badia et al. [96] proposed "never give up" (NGU): an agent that also targets hard exploration problems. NGU augments the environmental reward with a combination of two intrinsic novelty rewards: 1) an episodic reward, which enables the agent to quickly adapt within an episode and 2) a life-long novelty reward, which down-modulates states that become familiar across many episodes. Furthermore, NGU uses a universal value function approximator to learn several exploration policies with different exploration–exploitation tradeoffs at the same time. Agent57 [97] manages the tradeoff between exploration and exploitation using a "meta-controller" that adaptively selects a correct policy (ranging from very exploratory to purely exploitative) in the training phase. Agent57 outperforms the standard human benchmark on all 57 Atari games.

f) Exploration is the goal: Eysenbach et al. [98] noted that devising a reward function to drive exploration in sparse reward settings is challenging and requires domain-specific knowledge. Therefore, they proposed an exploration strategy that has two stages: unsupervised and supervised exploration. In stage one, the agent's objective is to learn policies (or skills) that are as diverse as possible without considering any rewards. In stage two, the agent uses its learned skills and a reward-driven exploration strategy to achieve a particular goal. The authors showed that such an exploration strategy makes the agents more capable of tackling novel environments and long-horizon tasks. In addition to diversity, Sharma et al. [99] argued that finding predictable skills eases their composition in hierarchical RL settings which leads to more capable RL agents. Interestingly, while this approach makes diversity a learning objective, ESs (which we will review next) uses diverse policies to approach an optimal one [74].

2) Exploration in Evolution Strategies: ESs optimize the fitness score while exploring the best solutions found so far. The exploration is realized through the recombination and mutation steps. Despite their effectiveness in exploration, ESs may still get trapped in local optima [100]. To overcome this limitation, many ES algorithms with enhanced exploration have been proposed.

One way to extract approximate gradients from a nonsmooth objective function, $f(\boldsymbol{\theta})$, is by adding noise to its parameter vector, $\boldsymbol{\theta}$. This yields a *new differentiable* function, $f_{\text{ES}}(\boldsymbol{\theta})$. OpenAI-ES [24] exploits this idea by sampling noise from a Gaussian distribution and adding it to the $\boldsymbol{\theta}$. The algorithm then optimizes using SGA. Additionally, OpenAI-ES relays on a few auxiliary techniques to enhance its performance: virtual batch normalization [101] for enhanced exploration, antithetic sampling [102] for reduced variance, and fitness shaping [34] for avoiding local optima.

Choromanski et al. [103] proposed two strategies to enhance the exploration of derivative-free optimization (DFO) methods, such as OpenAI-ES [24]: 1) structured exploration, which replaces random Gaussian directions with random orthogonal and quasi-Monte Carlo finite difference directions for more efficient parameter exploration and 2) compact policies, whereby imposing a parameter sharing structure on the policy architecture to significantly reduces the dimensionality of the problem without losing accuracy.

Maheswaranathan et al. [104] proposed guided ES: a random search that is augmented using surrogate gradients which are correlated with the true gradient. The key idea is to track a low-dimensional subspace that is defined by the recent history of surrogate gradients. Sampling this subspace leads to a drastic reduction in the variance of the search direction. However, this approach has two shortcomings: 1) the bias of the surrogate gradients needs to be known and 2) when the bias is too small, guided ES cannot find a better descent direction than the surrogate gradient. Meier et al. [105] draw inspiration from how momentum is used for optimizing DNNs to improve upon guided ES [104]. The authors showed how to optimally combine the surrogate gradient directions with random search directions and how to iteratively approach the true gradient for linear functions. They assessed their algorithm against a standard ESs algorithm on different tasks showing its superiority.

Choromanski et al. [106] noted that fixing the dimensionality of subspaces (as in guided ES [104]) leads to suboptimal performance. Therefore, they proposed adaptive sample-efficient blockbox optimization (ASEBO) an algorithm that adaptively controls the dimensionality of subspaces based on gradient estimators from previous iterations. ASEBO was compared with several ESs and DRL algorithms and showed a promising averaged performance.

Liu et al. [107] proposed self-guided ESs (SGES). This work is inspired by both ASEBO [106] and guided ES [104]. Furthermore, it is based on two main ideas: leveraging historically estimated gradients and building a guiding subspace from which search directions are sampled probabilistically. The results show that SGES outperforms OpenAI-ES [24], guided ES [104], and CMA-ES.

The aforementioned methods suffer from the curse of dimensionality due to the high variance of Monte Carlo gradient estimators. Motivated by this, Zhang et al. [108] proposed directional Gaussian smoothing ES (DGS-ES). It encourages

TABLE II
DRL AND ESs EXPLORATION ALGORITHMS

Algorithm	Description	Performance	Env.	Ref.
Bootstrapped DQN	uses DNNs and ensemble Q-functions to explore an environment	outperforms DQN by orders of magnitude in terms of cumulative rewards	sim.	[92]
UCB+InfoGain	integrates UCB with Q-function ensemble	outperforms bootstrapped DQN	sim.	[112]
State pseudo-count	uses density models to approximate state visitation count which is used to compute the reward bonus	superior to DQN, especially in hard-to-explore environments	sim.	[82]
VIME	measures information gain as KL divergence between current and updated distribution after an observation	improves the performance of TRPO [113], REINFORCE [17] when added to them	sim.	[88]
ICM	measures information gain as the difference between the predicted and observed state	outperforms TRPO-VIME in VizDoom (a sparse 3D environment)	sim.	[89]
Episodic curiosity	uses episodic memory to form the novelty bonus	outperforms ICM in visually rich 3D environments, e.g., VizDoom	sim.	[94]
Go-Explore	(i) Go-explore explores until a solution is found; (ii) Go-explore Robustifies the found solution	Performance improvements on hard exploration problems over other methods such as DQN+PixelCNN, DQN+CTS, BASS-hash	sim.	[95]
Never Give Up	combines both episodic and life-long novelties	obtains a median human normalized score of 1344%; the first algorithm that achieves non-zero rewards in the game of Pitfall	sim.	[96]
Agent57	uses a meta-controller for selecting the right policy	surpasses the standard human benchmark on all 57 Atari games	sim.	[97]
DIAYN	has two stages of explorations: unsupervised and supervised	uses skills learned in the first stage to tackle novel environments and long horizon tasks	sim.	[98]
OpenAI-ES	adds Gaussian noise to the parameter vector, computes a gradient, and takes a step in its direction	improves exploratory behaviors as compared to TRPO on tasks such as learning gaits of the MuJoCo humanoid walker	sim.	[24]
Structured Exploration	complements OpenAI-ES [24] with structured exploration and compact policies for efficient exploration	solves robotics tasks from OpenAI Gym using NN with 300 parameters (13x fewer than OpenAI-ES) and with near linear time complexity	sim.	[103]
Guided ES	leverages surrogate gradients to define a low-dimensional subspace for efficient sampling	improves over vanilla ESs and first-order methods that directly follow the surrogate gradient	sim.	[104]
ASEBO	adapts the dimensionality of the subspaces on-the-fly for efficient exploration	optimizes high-dimensional balck-box functions and performs consistently well across several tasks compared to state-of-the-art algorithms	sim.	[106]
DGS-ES	uses directional Gaussian smoothing to explore along non-local orthogonal directions.	improves on state-of-the-art algorithms (e.g., OpenAI-ES and ASEBO) on some problems	sim.	[108]
Iterative gradient estimation refinement	iteratively uses the last update direction as a surrogate gradient for the gradient estimator. Over time this will result in improved gradient estimates.	converges relatively fast to the true gradient for linear functions. It improves gradient estimation of ESs at no extra computational cost on MNIST and RL tasks	sim.	[105]
SGES	adapts a low-dimensional subspace on the fly for more efficient sampling and exploring	has lower gradient estimation variance as compared to OpenAI-ES. Superior performance over many ESs algorithms.	sim.	[107]
NS-ES, NSR-ES, and NSRA-ES	Hybridize novelty search (NS) and quality diversity (QD) with ESs to tackle sparse RL problems.	avoid local optima encountered by ESs while achieving higher performance on Atari and simulated robot tasks	sim.	[68]
AEES	Hybrid Cauchy and Gaussian distribution for enhanced mutation and exploration	samples a hybrid distribution to avoid local optima and enhance exploration	sim.	[111]

nonlocal exploration and improves high-dimensional exploration. In contrast to regular Gaussian smoothing, directional Gaussian smoothing conducts 1-D nonlocal explorations along d orthogonal directions. The Gauss–Hermite quadrature is then used for improving the convergence speed of the algorithm. Results show that DGS-ES is superior to OpenAI-ES [24] and ASEBO [106].

To encourage exploration in environments with sparse or deceptive rewards, Conti et al. [68] proposed hybridizing ESs with directed exploration methods (i.e., NS [109] and QD [110]). The combination resulted in three algorithms: NS-ES, NSR-ES, and NSRA-ES. NS-ES builds on the OpenAI-ES exploration strategy. OpenAI-ES approximates a gradient and takes a step in that direction. In NS-ES, the gradient estimate is that of the expected novelty. It gives directions on how to change the current policy's parameters $\boldsymbol{\theta}$ to increase the average novelty of the parameter distribution. NSR-ES is a variant of NS-ES. It combines both the reward and novelty signals to produce policies that are both novel and high performing. NSRA-ES is an extension of NSR-ES that dynamically adapts the weights of the novelty and the reward gradients for more optimal performance.

Ajani and Mallipeddi [111] discussed the advantages and disadvantages of using Cauchy or Gaussian distribution for mutation. The authors stated that the Cauchy distribution is preferred when the search is far away from the optimal solution, and Gaussian is better near the global optimum. However, identifying when to switch from Cauchy to Gaussian distribution is hard. Therefore, a hybrid mutation approach was adopted to enhance the exploration of the proposed algorithm. Results show great performance. But, the tests were done in a single environment. We hope to see follow-up work that examines hybrid mutation approaches in more complex environments. Table II summarizes important DRL and ESs exploration algorithms. Fig. 6 shows important exploration agents in chronological order.

C. Meta Reinforcement Learning

Meta-RL can be subdivided into two categories [114].

Recurrent Models (RNN-Based Learners): Leveraging the agent-environment interaction history provides more information, which leads to improved learning [45], [115]. This idea can be implemented using RNNs (or other recurrent models) [43], [116], [117], [118]. The RNNs can be trained on a set of tasks to learn a hidden state (meta-policy), then this hidden state can be further adapted given new observations from an unseen task.

The general architecture of a meta-RL algorithm is illustrated in Fig. 7, where an agent is modeled as two loops, both

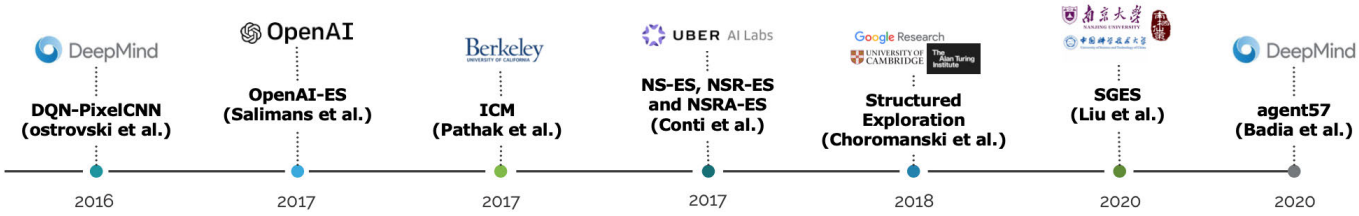

Fig. 6. DRL and ESs exploration algorithms are shown on a timeline.

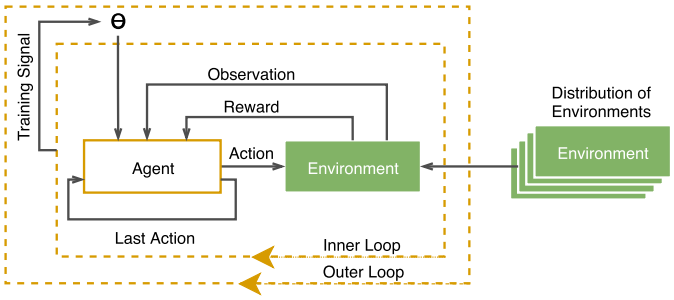

Fig. 7. Schematic of meta-RL [123].

implementing RL algorithms. The outer loop samples a new environment every iteration and tunes the parameters of the inner loop. Consequently, the inner loop adjusts more rapidly to new tasks.

Duan et al. [117] and Wang et al. [118] proposed analogous recurrent meta-RL agents: R^2 and DRL-meta. These agents use RNNs to track characteristics of interaction trajectories. The main difference between both approaches relates to the set of environments. Environments in [118] are issued from a parameterized distribution [119]. In contrast, those in [117] are relatively unrelated [119].

Such RNN-based methods have proven to be efficient on many RL tasks. However, their performance decreases as the complexity of the task increases, especially with long temporal dependencies. Additionally, RNN-based meta-learners cannot pinpoint specific prior experiences [43], [120].

To overcome these limitations, Mishra et al. [43] proposed simple neural attentive learner (SNAIL). It combines temporal convolutions and attention mechanisms. The former aggregates information from past experiences and the latter pinpoints specific pieces of information. SNAIL consists of three main parts: 1) DenseBlock, a causal 1-D-convolution with a specific dilation rate; 2) TCBlock, a series of DenseBlocks with exponentially increasing dilation rates; and 3) AttentionBlock, where key-value lookups take place. This general-purpose model has shown its efficacy on tasks ranging from supervised to RL. Despite that, challenges, such as a long time, needed for getting the right architectures of TCBlocks and DenseBlocks. [120] persist.

Gradient-Based Models: Model agnostic meta-learning (MAML) [121] realizes meta-learning principles by learning an initial set of parameters, $\boldsymbol{\theta_0}$, of a model such that taking a few gradient steps is sufficient to tailor this model to a specific task. More precisely, MAML learns $\boldsymbol{\theta_0}$ such that for any randomly sampled task, $\mathcal{T}$, with a loss function, $\mathcal{L}$, the agent will have a modest loss after n updates

$$\boldsymbol{\theta_0} = \arg\min_{\boldsymbol{\theta}} \mathbb{E}_{\mathcal{T}}\left[\mathcal{L}_{\mathcal{T}}\left(U_{\mathcal{T}}^{n}(\boldsymbol{\theta})\right)\right]$$

where $U_{\mathcal{T}}^{n}(\boldsymbol{\theta})$ refers to an update rule, such as gradient descent.

Nichol et al. [122] proposed Reptile a first-order meta-learning framework, that is considered to be an approximation of MAML. Similar to first-order MAML (FOMAML), Reptile does not calculate second derivatives, which makes it less computationally demanding. It starts by repeatedly sampling a task, then performing N iterations of stochastic gradient descent on each task to compute a new set of parameters. Then, it moves the model weights toward the new parameters. Next, we look at how meta-learning makes ESs more efficient.

1) Meta Evolution Strategies: Gajewski et al. [124] introduced "evolvability ES," an ES-based meta-learning algorithm for RL tasks. It combines concepts from evolvability search [125], ESs [24], and MAML [121] to encourage searching for individuals whose immediate offsprings show signs of behavioral diversity (that is, it searches for parameter vectors whose perturbations lead to differing behaviors) [125]. Consequently, evolvability ES facilitates adaptation and generalization while leveraging the scalability of ESs [124], [126]. Evolvability ES shows a competitive performance to gradient-based meta-learning algorithms. Quality evolvability ES [126] noted that the original evolvability ES [127] can only be used to solve problems where the task performance and evolvability align. To eliminate this restriction, quality evolvability ES optimizes for both—task performance and evolvability—simultaneously.

Song et al. [128] argue that policy gradient-based MAML algorithms [121] face significant difficulties when estimating the second derivative using backpropagation on stochastic policies. Therefore, they introduced ES-MAML, a meta-learner that leverages ES [24] for solving MAML problems without estimating second derivatives. The authors empirically showed that ES-MAML is competitive with other meta-RL algorithms. Song et al. [129] combined hill-climbing adaptation with ES-MAML to develop a noise-tolerant meta-RL learner.

Wang et al. [130] introduced instance weighted incremental ESs (IW-IESs). It incorporates an instance weighting mechanism with ESs to generate an adaptable and salable meta-learner. IW-IES assigns weights to offspring proportional to the amount of new knowledge they acquire. The weights are assigned based on one of the two metrics: instance novelty and instance quality. Compared with ES-MAML, IW-IES proved competitive for robot navigation tasks.

TABLE III
META-LEARNING

Algorithms	Description	Performance	Env.	Ref.
DRL-meta	trains an RNN on a distribution of RL tasks. The RNN serves as a dynamic task embedding storage. DRL-meta uses the LSTM	outperforms other benchmarks on the bandits' problems; properly adapts to invariances in MDP tasks	sim.	[118]
RL^2	trains an RNN on a distribution of RL tasks. The RNN serves as a dynamic task embedding storage.	comparable to theoretically optimal algorithms in small-scale settings. It has the potential to scale to high-dimensional tasks	sim.	[117]
SNAIL	combines temporal convolution and attention mechanisms	outperforms LSTM and MAML	sim.	[43]
MAML	searches for optimal initial parameters for a task distribution such that individual tasks can be solve with a few gradient steps.	outperforms classical methods such as random and pretrained methods	sim.	[121]
Reptile	similar to first-order MAML	on-par with the performance of MAML	sim.	[122]
Evolvability ES	combines concepts from evolvability search, ES, and MAML to enable a quickly adaptable meta-learner	competitive with MAML on 2D and 3D locomotion tasks	sim.	[124]
ES-MAML	uses ESs to overcome the limitations of MAML	competitive with policy gradient methods	sim.	[128]
IW-IES	uses NES for updating the RL policy network parameters in a dynamic environment	outperforms ES-MAML on set of robot navigation tasks	sim.	[130]

Meta-RL is particularly suited for tackling the sim-to-real problem: simulation provides previous experiences that are used to learn a general policy, and the data obtained from operating in the real world fine tunes that policy [131], [132]. Examples of using meta-RL to train physical robots include: Nagabandi et al. [44] built on top of MAML a model-based meta-RL agent to train a legged millirobot; Arndt et al. [133] proposed a similar framework to MAML to train a robot on a task of hitting a hockey puck; and Song et al. [129] introduced a variant of ES-MAML to train and quickly adapt the policy commanding a legged robot. Table III summarizes the main characteristics of the presented algorithms.

D. Learning in Multiagent Settings

An MAS is a distributed system of multiple cooperating or competing agents, working toward maximizing their objectives within a shared environment [47]. Next, multiagent DRL (MADRL) and multiagent ESs are discussed.

1) Multiagent Deep Reinforcement Learning: Moving from a single-agent RL to a multiagent RL brings about new complex challenges with respect to learning and evaluating outcomes. This can be attributed to several factors, including the exponential growth of the search space and the nonstationarity of the environment [134]. *Nonstationarity*: In MADRL, agents learn concurrently and their actions reshape their shared surroundings repeatedly. This causes their environment to be nonstationary. Consequently, the convergence of well-known algorithms, such as Q-learning, can no longer be guaranteed as the Markov property assumption is violated [15], [135], [136]. Many papers attempt to address the nonstationarity problem. Castaneda [137] proposed two algorithms: Deep loosely coupled Q-network (DLCQN) and deep repeated update Q-network (DRUQN). DLCQN modifies an independence degree for each agent based on the agent's negative rewards and observations. The agent then utilizes this independence degree to decide when to act independently or cooperatively. DRUQN tries to avoid policy bias by making the value of an action inversely proportional to the probability of selecting that action. The use of an experience replay buffer with DQN enables efficient learning. However, due to the nonstationarity of the environment in MADRL data stored in an experience replay buffer can become outdated. To counter this unwanted behavior, lenient-DQN conceived by Palmer et al. [138] utilizes decaying temperature values for adjusting the policy updates sampled from the replay buffer.

a) Scalability: One way to deal with the nonstationarity problem is to train the agents in a centralized fashion and let them act according to a joint policy. However, this approach is not scalable because the state-action space grows exponentially with a number of agents [139], [140], [141]. To balance the challenges imposed by nonstationarity and scalability, a centralized training and decentralized execution approach has been proposed [142], [143], [144], [145].

b) Credit assignment: One of the main challenges of learning in a cooperative setting is termed a "multiagent credit assignment problem," which refers to how to divide a reward obtained on a team level amongst individual learners [146]. Due to the complex interaction dynamics of the agents, it is not trivial to determine whose actions were beneficial to the group reward. Researchers have proposed different approaches to deal with the challenges associated with MADRL problems.

c) Independent-learning: Under this approach, each agent considers other agents as part of the environment; consequently, each agent is trained independently [136], [147], [148]. This approach does not suffer from the scalability problem [148], [149], but it makes the environment nonstationary from each agent's perspective [150]. Furthermore, it conflicts with the usage of experience replay that improves the DQN algorithm [15]. To stabilize the experience replay buffer in MADRL settings, Foerster et al. [149] used importance sampling and replay buffer samples aging.

d) Fully observable critic: A way to deal with the nonstationarity of an MADRL environment is by leveraging an actor–critic approach. Lowe et al. [151] proposed a multiagent deep deterministic policy gradient (MADDPG) algorithm, where the actor policy accesses only the local observations, whereas the critic has access to the actions, observations, and target policies of all agents during training. As the critic has

global observability, the environment becomes stationary even though the policies of other agents change. Many extensions to MADDPG have been proposed [152], [153], [154], [155].

e) Value function decomposition: To coordinate the agents' actions, learning a centralized action-value function, Q_{tot}, is desirable. However, when the number of agents is large, learning such a function is challenging. Independent learning (where each agent learns its action-value function, Q_i) does not face such a challenge, but it also neglects interactions between agents, which results in suboptimal collective performance. Value function decomposition methods try to capitalize on the advantages of these two approaches. It represents Q_{tot} as a mixing of Q_i that is conditioned only on local information. Value-decomposition network (VDN) algorithm assumes that Q_{tot} can be additively decomposed into $N\,Q_i$ for N agents. QMIX [143] algorithm improves on VDN by relaxing some of the additivity constraints and enforcing positive weights on the mixer network.

f) Learning to communicate: Cooperative environments may allow agents to communicate and achieve their shared objective more optimally [156], [157]. Foerster et al. [158] proposed two algorithms, reinforced interagent learning (RIAL) and differentiable interagent learning (DIAL), which utilize DNNs to learn to communicate. RIAL is based on deep recurrent Q-network with independent Q-learning. It shares the parameters of a single-neural network between the agents. In contrast, DIAL passes gradients directly via the communication channel during learning. While a discrete communication channel is used in realizing RIAL and DIAL, CommNet [159] utilizes a continuous vector channel. Over this channel, agents obtain the summed transmissions of other agents. Results show that agents can learn to communicate and improve their performance over noncommunicating agents.

g) Partial observability: Foerster et al. [158] introduced a deep distributed recurrent Q-network (DDRQN) algorithm based on a long short-term memory network to deal with POMDP problems in multiagent settings. Gupta et al. [160] extended single-agent RL algorithms based on policy gradient, temporal-difference error, and actor–critic methods to multiagent settings. Their work shows the importance of using DRL with curriculum learning to address the problem of learning cooperative policies in partially observable environments.

We refer the reader to the following surveys for a more in-depth discussion on MADRL: Hernandez-Leal et al. [135] focus on the nonstationarity problem in MADRL; Oroojlooy-Jadid and Hajinezhad [150] scope their survey to decentralized MADRL with cooperative goals; Da Silva and Costa [161] survey transfer learning for MADRL; and Du and Ding [162] review challenges and applications of MADRL.

2) Multiagent Evolution Strategies: ES algorithms do not require the problem to be formulated as an MDP; therefore, they do not suffer from the nonstationarity of the environment. Consequently, it is relatively easy to extend a single-agent ES algorithm to the multiagent domain and develop an application. Hiraga et al. [163] developed robotics controllers based on ESs for managing congestion in robotic swarm path formation using LEDs. A neural network represents the controller of the robot. (μ, λ)-ES is utilized to optimize the weights of the controller. A copy of the controller is implemented on N different robots, before being evaluated and assessed depending on the swarm's performance. Another similar approach was proposed in [164] for building a swarm capable of cooperatively transporting food to a nest and collectively distinguishing between foods and poisons. Hiraga et al. [164] developed a controller for a robotic swarm using CMA-ES, aiming to automatically generate the behavior of the robots.

Tang et al. [165] proposed an adversarial training multiagent learning system, in which a quadruped robot (protagonist) is trained to become more agile by hunting an ensemble of robots that are escaping (adversaries) following different strategies. An ensemble of adversaries is used, as each will propose a different escape strategy, thus improving agility. Training is done using ESs and more specifically by augmenting CMA-ES to the multiagent framework. There are two steps for training: An outer loop that iteratively trains the protagonist and adversaries, and an inner loop for optimizing the policy of each. Policies are represented by feed-forward neural networks and are optimized with CMA-ES.

Chen and Gao [166] proposed a predator–prey system that leverages ESs. It consists of multiple predators trained to catch prey in a certain time frame. The predator controllers are homogeneous and are represented by DNNs, whose parameters are optimized with ESs (OpenAI-ES and CMA-ES) and Bayesian optimization. The DNN has three inputs, one hidden layer, and two outputs for controlling the angular velocities of the two wheels. As for the prey's controller, it follows a simple fixed evasion strategy: having computed a danger zone map, the prey navigates toward the least dangerous locations. After performing various experiments, the predators showcased a successful collective behavior: moving following a formation and avoiding collisions.

Multiagent Credit Assignment Problem: In a multiagent setting, agents often receive a shared reward for all the agents, making it harder to learn proper cooperative behaviors. Li et al. [167], thus, proposed to use parallelized ESs along with a value decomposition network (VDN) (useful for identifying each agent's contribution to the training process) for solving cooperative multiagent tasks. Fig. 8 is an overview of the overall parallelized ES with value decomposition (PES-VD) algorithm, which consists of two phases. First, the policies of each agent are represented by a DNN with parameters $\boldsymbol{\theta}$, optimized using parallelized ES. Each agent, thus, identifies its actions independently following its policy. In the second place, seeing how the reward is common to the whole team, a value VDN is used to compute the fitness for each of the different policies. PES-VD is implemented in parallel on multiple cores: M workers evaluate the policies and compute the gradients of the VDN and a master node collects the data and updates the policies and the VDN accordingly.

Various researchers proposed multiagent solutions for swarm scenarios leveraging ESs. Each robot in the swarm runs the same network, thus maintaining collective behavior. Rais Martínez and Aznar Gregori [168] assess the performance of ESs (CMA-ES, PEPG, SES, GA, and OpenAI-ES) for

TABLE IV
HYBRID ALGORITHMS HIGHLIGHTS

Algorithm	Description	Experiments	Env.	Ref.
CEM-RL	combines a cross-entropy method and Twin Delayed Deep Deterministic policy gradient [174] to find robust policies	outperforms CEM, TD3, multi-actor TD3, and Evolutionary Reinforcement Learning [180]	sim.	[173]
Evolved Policy Gradients (EPG)	uses gradient descent and CMA-ES for policy and loss function optimization, respectively	learns faster than policy gradient methods and provides qualitatively different behavior than other meta-learners	sim.	[182]
MO-CMA-ES	integrates a multi-policy soft actor-critic algorithm with a multi-objective covariance matrix adaptation evolution strategy to approach uniform Pareto frontier	exceeds other algorithms such as hypervolume-based [187], radial [188], Pareto following [188], and Deep Neuroevolution [171] algorithm on computing the Pareto frontier	sim.	[183]
Fine-tuned DRL	combines CMA-ES and DQN or DDPG to train and fine-tune a DNN control policy	surpasses gradient-based methods while requiring less iterations than gradient-free ones	sim.	[184]

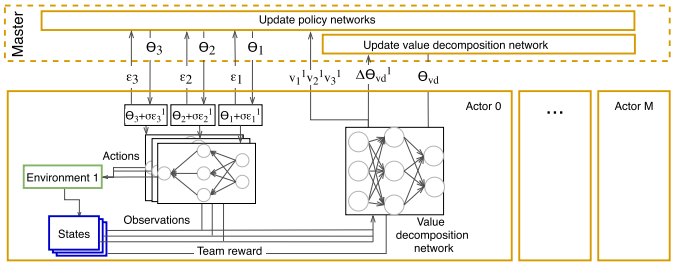

Fig. 8. PES-VD network overview.

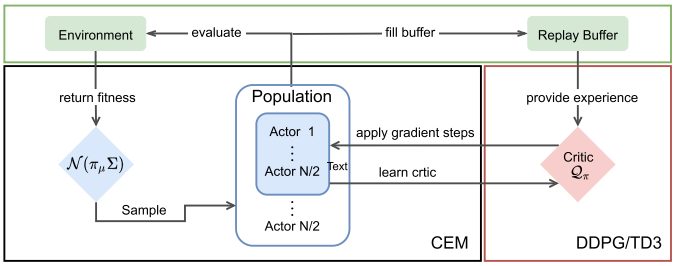

Fig. 9. CEM-RL [173]: a hybrid algorithm that combines cross-entropy with deep deterministic gradient policy [174].

multiagent learning in the swarm aggregation task. Similarly, Fan et al. [169] used ESs on different multiagent UAV swarm combat scenarios. Aznar et al. [170] developed a swarm foraging behavior using DRL and CMA-ES.

E. Observations

Our main takeaways from this section are as follows.

1) ESs communicate only scalars to parallelize the optimization process, whereas DRL algorithms communicate parameters or gradient vectors. Consequently, ESs are easier to parallelize and scale than DRL.
2) ESs sample candidate solutions (or policies) and commit to them until the next generation. Therefore, ESs do temporally extended exploration (or deep exploration) [171]. On the other hand, DRL algorithms need to explicitly enforce deep exploration [92].
3) Distributed settings diversify data updates, which enable DRL algorithms to leave out the experience replay buffer [12]. ESs do not utilize a replay buffer.
4) Benchmarking exploration strategies happen almost exclusively in simulated/gaming environments. Consequently, the efficacy of these algorithms in real-world applications is mostly unknown.
5) Gradient-based meta-RL faces many challenges. For example, estimating the first- and second-order derivatives, high variance, and high computation needs.
6) ES-based meta-RL attempts to address the limitations of gradient-based meta-RL; however, ES-based meta-RL itself faces significant challenges, such as sample efficiency.
7) Meta-RL is particularly suited for tackling the sim-to-real problem: a generic policy is trained in simulation and fine-tuned via interaction with the real world.
8) New algorithms, such as PES-VD [167], propose a direct solution to some of the main challenges of MADRL. PES-VD uses a VDN for solving multiagent credit assignment problems.
9) Using ESs for multiagent learning is a growing field with a large potential as to the many advantages ESs brings to concepts, such as "collective robotic learning" and "cloud robotics" [172], with its improved parallelism [24].

IV. HYBRID DEEP REINFORCEMENT LEARNING AND EVOLUTION STRATEGIES ALGORITHMS

Although DRL and ESs have the same objective—optimizing an objective function in a potentially unknown environment—they have different strengths and weaknesses [175], [176]. For example, DRL can be sample efficient, while ESs have robust convergence properties and exploration strategies. The hybrid approach combines DRL and ESs to get the best of both worlds. Although the idea is not new [177], hybridizing DRL and ESs has gained momentum, driven by their recent success [24], [178], [179]. Next, we describe a few population-guided parallel learning schemes that enhance the performance of RL algorithms (Table IV).

Pourchot and Sigaud [173] addressed the problem of policy search by proposing CEM-RL: a hybrid algorithm that combines a cross-entropy method (CEM) with either the twin delayed deep deterministic (TD3) policy gradient [174] or the deep deterministic policy gradient DDPG [13] algorithms (Fig. 9). The CEM-RL architecture consists of a population of actors that are generated using CEM, and a single DDPG or TD3 agent. The actors generate diversified training data for the DDPG/TD3 agent, and the gradients obtained from DDPG/TD3 are periodically inserted into the population of

the CEM to optimize the searching process. The authors showed that CEM-RL is superior to CEM, TD3 [174], and evolution RL (ERL) [180]: a hybrid algorithm that combines a DDPG agent with an evolutionary algorithm. Shopov and Markova [181] combined ESs and MADRL (DQNs) for sequential games and showcased the model's efficiency as compared with classical multiagent reinforcement training with ϵ-greedy. The agent was tested in two environments: one with almost no obstacles; and the second with many obstacles to increasing the probability of getting the agent to trap into a local minimum. The ESs-based agent showed better performance in an environment with many obstacles.

Houthooft et al. [182] devised a hybrid RL agent, evolved policy gradients (EPGs), which optimize a policy and a loss function. EPG consists of two optimization loops: the inner loop uses stochastic gradient descent to optimize the agent's policy, while the outer one utilizes ES to tune the parameters of a loss function that the inner loop minimizes. This architecture enables EPG to learn faster than a standard RL agent.

Chen et al. [183] proposed a hybrid agent to approximate the Pareto frontier uniformly in a multiobjective decision-making problem. The authors argued that despite the fast convergence of DRL, it cannot guarantee a uniformly approximated Pareto frontier. On the other hand, ES achieves a well-distributed Pareto frontier, but they face difficulties optimizing a DNN. Thus, Chen et al. [183] proposed a two-stage multiobjective RL (MORL) framework. In the first stage, a multipolicy soft actor–critic algorithm learns multiple policies collaboratively. And, in the second stage, a multiobjective covariance matrix adaptation ES (MO-CMA-ES) fine-tunes policy-independent parameters to approach a uniform Pareto frontier.

de Bruin et al. [184] used a hybrid approach to train and fine-tune a DNN control policy. Their approach consists of two main steps: 1) learning a state representation and initial policy from high-dimensional input data using gradient-based methods (i.e., DQN or DDPG) and 2) fine-tuning the final action selection parameters of the DNN using CMA-ES. This architecture enables the policy to surpass its in performance and its gradient-based counterpart while using fewer trials compared with a pure gradient-free policy.

Several other researchers have also proposed solutions hybridizing ES and DRL for various applications. For example, Song et al. [185] proposed ES-ENAS, a neural architecture search (NAS) algorithm for identifying RL policies using ES and Efficient NAS (ENAS); Ferreira et al. [186] used ES to learn agent-agnostic synthetic environments (SEs) for RL.

A. Observations

Here, we summarize our observations of this section.

1) DRL suffers from temporal credit assignment, and sensitivity in the hyperparameters' selection and might suffer from more brittle exploration due to its unique agent settings, while ES has low data efficiency and struggle with large optimization tasks.
2) Combining both approaches can help address some of these identified challenges.
3) Some hybrid methods proposed throughout the literature seem to outperform the use of each method on its own.

V. Challenges and Future Research Directions

Although DRL and ESs have proven their worth in many AI fields, there are still many challenges to be addressed.

A. Sample Efficiency

DRL agents require a large number of samples (i.e., interactions with environments) to learn performant policies. However, collecting a sufficient number of samples is not always feasible due to computational and/or quantity of interaction limitations. Recently, using model-based RL to improve sample efficiency is gaining more attraction [189], [190].

ESs work with full-length episodes [191], [192], and they do not memorize [191]. Therefore, they are less sample efficient than DRL. There have been a few attempts to improve the sample efficiency of ESs [191], [193], but it is still a challenge.

B. Exploration Versus Exploitation

The exploration versus exploitation dilemma is one of the most prominent problems in RL. Recent breakthroughs have enabled the exploration of novel environments. For example, Osband et al. [92] observed the importance of temporal correlation and proposed the bootstrapped DQN, and Bellemare et al. [82] used density models to scale UCB to problems with high-dimensional data. Despite that, exploring complex environments is still a very active field of research.

ESs realize exploration through the recombination and mutation steps. Despite their effectiveness, ESs may still get trapped in local optima [70], [100]. Proposals have been made to enhance ESs exploration capabilities [103], [104]; however, more work in this direction is needed.

C. Sparse Reward

A reward signal guides the learning process of an RL agent. When this signal is sparse learning becomes hard. Approaches, such as reward shaping [1], curiosity-driven methods [89], and curriculum learning [194], improve learning in sparse settings. However, the challenge of learning from sparse rewards is far from being solved. Directing the exploration of ES algorithms to counter the sparsity and/or deceptiveness of a task is one of the most important challenges to scale ESs to more complex environments and make them more efficient [19].

D. Learning a Plan

An emerging line of research is trying to extend the DRL framework to tackle long-horizon planning problems. The premise is that by learning a set of primitive skills (e.g., picking an object and sealing a box), an agent will be able to learn how to use these skills to devise a complete plan and achieve an intended goal (e.g., placing 20 different objects in a box and prepare it for delivery) [98].

VI. Conclusion

DRL and ES have the same objective but use different mechanisms to solve sequential decision-making problems. This survey tracks their recent development and highlights their relative strengths and weaknesses. The survey considered major learning aspects, such as parallelism, exploration, meta-learning, and multiagent learning. We hope that this comparative survey promotes ideas for more general and robust DRL/ESs algorithms. Our conclusion is that the parallel development of DRL and ESs continues, but hybridizing them has the potential to lead to significant breakthroughs.

References

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.

[2] D. Bertsekas, *Reinforcement Learning and Optimal Control*. Nashua, NH, USA: Athena Scientific, 2019.

[3] V. François-Lavet et al., "An introduction to deep reinforcement learning," *Found. Trends Mach. Learn.*, vol. 11, nos. 3–4, pp. 219–354, 2018.

[4] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing*. Berlin, Germany: Springer, 2003.

[5] H. Qian and Y. Yu, "Derivative-free reinforcement learning: A review," *Frontiers Comput. Sci.*, vol. 15, no. 6, Dec. 2021, Art. no. 156336.

[6] M. M. Drugan, "Reinforcement learning versus evolutionary computation: A survey on hybrid algorithms," *Swarm Evol. Comput.*, vol. 44, pp. 228–246, Feb. 2019.

[7] G. Rummery and M. Niranjan, "On-line Q-learning using connectionist systems," Springer, Berlin, Germany, Tech. Rep., CUED/F-INFENG/TR 166, 1994.

[8] D. Zhao, H. Wang, K. Shao, and Y. Zhu, "Deep reinforcement learning with experience replay based on SARSA," in *Proc. IEEE Symp. Comput. Intell. (SSCI)*, Dec. 2016, pp. 1–6.

[9] V. Mnih et al., "Playing Atari with deep reinforcement learning," 2013, *arXiv:1312.5602*.

[10] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Proc. NeurIPS*, 1999, pp. 1–7.

[11] A. V. Clemente, H. N. Castejón, and A. Chandra, "Efficient parallel methods for deep reinforcement learning," 2017, *arXiv:1705.04862*.

[12] V. Mnih et al., "Asynchronous methods for deep reinforcement learning," in *Proc. ICML*, 2016, pp. 1928–1937.

[13] T. P. Lillicrap et al., "Continuous control with deep reinforcement learning," 2015, *arXiv:1509.02971*.

[14] R. S. Sutton, "Dyna, an integrated architecture for learning, planning, and reacting," *ACM SIGART Bull.*, vol. 2, no. 4, pp. 160–163, Jul. 1991.

[15] V. Mnih et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[16] L.-J. Lin, "Reinforcement learning for robots using neural networks," Carnegie-Mellon Univ. Pittsburgh PA School Comput. Sci., Pittsburgh, PA, USA, Tech. Rep. ADA261434, 1993.

[17] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 229–256, May 1992.

[18] N. Hansen, D. V. Arnold, and A. Auger, "Evolution strategies," in *Springer Handbook of Computational Intelligence*. Berlin, Germany: Springer, 2015.

[19] Z. Li, X. Lin, Q. Zhang, and H. Liu, "Evolution strategies for continuous optimization: A survey of the state-of-the-art," *Swarm Evol. Comput.*, vol. 56, Aug. 2020, Art. no. 100694.

[20] V. Heidrich-Meisner and C. Igel, "Similarities and differences between policy gradient methods and evolution strategies," in *Proc. ESANN*, 2008, pp. 149–154.

[21] V. Heidrich-Meisner and C. Igel, "Evolution strategies for direct policy search," in *Proc. PPSN*, 2008, pp. 428–437.

[22] V. Heidrich-Meisner and C. Igel, "Neuroevolution strategies for episodic reinforcement learning," *J. Algorithms*, vol. 64, no. 4, pp. 152–168, Oct. 2009.

[23] V. Heidrich-Meisner and C. Igel, "Hoeffding and Bernstein races for selecting policies in evolutionary direct policy search," in *Proc. 26th Annu. Int. Conf. Mach. Learn.*, Jun. 2009, pp. 401–408.

[24] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever, "Evolution strategies as a scalable alternative to reinforcement learning," 2017, *arXiv:1703.03864*.

[25] P. Chrabaszcz, I. Loshchilov, and F. Hutter, "Back to basics: Benchmarking canonical evolution strategies for playing Atari," 2018, *arXiv:1802.08842*.

[26] I. Rechenberg, *Evolutionsstrategie Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Stuttgart-Bad Cannstatt: Friedrich Frommann Verlag, 1973.

[27] A. Slowik and H. Kwasnicka, "Evolutionary algorithms and their applications to engineering problems," *Neural Comput. Appl.*, vol. 32, no. 16, pp. 12363–12379, Aug. 2020.

[28] M. Dianati, I. Song, and M. Treiber, "An introduction to genetic algorithms and evolution strategies," Citeseer, Princeton, NJ, USA, Tech. Rep., 2002.

[29] H.-P. Schwefel, "Evolutionsstrategien für die numerische optimierung," in *Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie*. Basel, Switzerland: Birkhäuser, 1977.

[30] N. Hansen and A. Ostermeier, "Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation," in *Proc. IEEE Int. Conf. Evol. Comput.*, May 1996, pp. 312–317.

[31] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evol. Comput.*, vol. 9, no. 2, pp. 159–195, Jun. 2001.

[32] N. Hansen, S. D. Müller, and P. Koumoutsakos, "Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES)," *Evol. Comput.*, vol. 11, no. 1, pp. 1–18, Mar. 2003.

[33] N. Hansen, "The CMA evolution strategy: A tutorial," 2016, *arXiv:1604.00772*.

[34] D. Wierstra, T. Schaul, T. Glasmachers, Y. Sun, J. Peters, and J. Schmidhuber, "Natural evolution strategies," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 949–980, 2014.

[35] A. K. McCallum, *Reinforcement Learning With Selective Perception and Hidden State*. Rochester, NY. USA: Univ. Rochester, 1996.

[36] R. Ortner, O.-A. Maillard, and D. Ryabko, "Selecting near-optimal approximate state representations in reinforcement learning," in *Proc. ALT*, 2014, pp. 140–154.

[37] V. François-Lavet, G. Rabusseau, J. Pineau, D. Ernst, and R. Fonteneau, "On overfitting and asymptotic bias in batch reinforcement learning with partial observability," *J. Artif. Intell. Res.*, vol. 65, pp. 1–30, May 2019.

[38] P. Zhu, X. Li, P. Poupart, and G. Miao, "On improving deep reinforcement learning for POMDPs," 2017, *arXiv:1704.07978*.

[39] B. Eker and H. L. Akın, "Using evolution strategies to solve DEC-POMDP problems," *Soft Comput.*, vol. 14, no. 1, pp. 35–47, Jan. 2010.

[40] M. Hausknecht and P. Stone, "Deep recurrent Q-learning for partially observable MDPs," 2015, *arXiv:1507.06527*.

[41] C. Igel, "Neuroevolution for reinforcement learning using evolution strategies," in *Proc. IEEE CEC*, Dec. 2003, pp. 2588–2595.

[42] T. Schaul and J. Schmidhuber, "Metalearning," *Scholarpedia*, vol. 5, no. 6, p. 4650, 2010.

[43] N. Mishra, M. Rohaninejad, X. Chen, and P. Abbeel, "A simple neural attentive meta-learner," 2017, *arXiv:1707.03141*.

[44] A. Nagabandi et al., "Learning to adapt in dynamic, real-world environments through meta-reinforcement learning," 2018, *arXiv:1803.11347*.

[45] F. Garcia and P. S. Thomas, "A meta-MDP approach to exploration for lifelong reinforcement learning," in *Proc. NeurIPS*, 2019, pp. 1–10.

[46] A. Nowé, P. Vrancx, and Y.-M. D. Hauwere, "Game theory and multi-agent reinforcement learning," in *Reinforcement Learning*. Berlin, Germany: Springer, 2012.

[47] J. Hu, H. Niu, J. Carrasco, B. Lennox, and F. Arvin, "Voronoi-based multi-robot autonomous exploration in unknown environments via deep reinforcement learning," *IEEE Trans. Veh. Technol.*, vol. 69, no. 12, pp. 14413–14423, Dec. 2020.

[48] L. S. Shapley, "Stochastic games," *Proc. Nat. Acad. Sci. USA*, vol. 39, no. 10, pp. 1095–1100, 1953.

[49] Y. Gong, M. Abdel-Aty, Q. Cai, and M. S. Rahman, "Decentralized network level adaptive signal control by multi-agent deep reinforcement learning," *Transp. Res. Interdiscipl. Perspect.*, vol. 1, Jun. 2019, Art. no. 100020.

[50] H. Wei et al., "CoLight: Learning network-level cooperation for traffic signal control," in *Proc. 28th ACM Int. Conf. Inf. Knowl. Manage.*, Nov. 2019, pp. 1913–1922.

[51] G. Sartoretti et al., "PRIMAL: Pathfinding via reinforcement and imitation multi-agent learning," *IEEE Robot. Autom. Lett.*, vol. 4, no. 3, pp. 2378–2385, Jul. 2019.

[52] G. Sartoretti, Y. Wu, W. Paivine, T. Kumar, S. Koenig, and H. Choset, "Distributed reinforcement learning for multi-robot decentralized collective construction," in *Distributed Autonomous Robotic Systems*. Cham, Switzerland: Springer, 2019.

[53] M. Brittain and P. Wei, "Autonomous air traffic controller: A deep multi-agent reinforcement learning approach," 2019, *arXiv:1905.01303*.

[54] P. Xuan and V. Lesser, "Multi-agent policies: From centralized ones to decentralized ones," in *Proc. 1st Int. Joint Conf. Auto. Agents Multiagent Syst.*, 2002, pp. 1098–1105.

[55] K. Zhang, Z. Yang, H. Liu, T. Zhang, and T. Basar, "Fully decentralized multi-agent reinforcement learning with networked agents," in *Proc. ICML*, 2018, pp. 5872–5881.

[56] K. Zhang, Z. Yang, and T. Basar, "Multi-agent reinforcement learning: A selective overview of theories and algorithms," in *Handbook of Reinforcement Learning and Control*. 2019.

[57] V. François-Lavet, R. Fonteneau, and D. Ernst, "How to discount deep reinforcement learning: Towards new dynamic strategies," 2015, *arXiv:1512.02011*.

[58] A. Nair et al., "Massively parallel methods for deep reinforcement learning," 2015, *arXiv:1507.04296*.

[59] D. Horgan et al., "Distributed prioritized experience replay," 2018, *arXiv:1803.00933*.

[60] L. Espeholt et al., "IMPALA: Scalable distributed deep-RL with importance weighted actor-learner architectures," in *Proc. ICML*, 2018, pp. 1407–1416.

[61] L. Espeholt, R. Marinier, P. Stanczyk, K. Wang, and M. Michalski, "SEED RL: Scalable and efficient deep-RL with accelerated central inference," 2019, *arXiv:1910.06591*.

[62] M. Grounds and D. Kudenko, "Parallel reinforcement learning with linear function approximation," in *Proc. 6th Int. Joint Conf. Auton. Agents Multiagent Syst.*, 2005, pp. 1–3.

[63] J. Dean et al., "Large scale distributed deep networks," in *Proc. NIPS*, 2012, pp. 1–3.

[64] M. Babaeizadeh, I. Frosio, S. Tyree, J. Clemons, and J. Kautz, "Reinforcement learning through asynchronous advantage actor-critic on a GPU," 2016, *arXiv:1611.06256*.

[65] N. Heess et al., "Emergence of locomotion behaviours in rich environments," 2017, *arXiv:1707.02286*.

[66] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, *arXiv:1707.06347*.

[67] S. Kapturowski, G. Ostrovski, J. Quan, R. Munos, and W. Dabney, "Recurrent experience replay in distributed reinforcement learning," in *Proc. ICLR*, 2018, pp. 1–19.

[68] E. Conti, V. Madhavan, F. Petroski Such, J. Lehman, K. Stanley, and J. Clune, "Improving exploration in evolution strategies for deep reinforcement learning via a population of novelty-seeking agents," in *Proc. NeurIPS*, 2018, pp. 1–12.

[69] G. Paolo, "Learning in sparse rewards settings through quality-diversity algorithms," 2022, *arXiv:2203.01027*.

[70] G. Liu et al., "Trust region evolution strategies," in *Proc. AAAI*, 2019, pp. 4352–4359.

[71] L. Fuks, N. Awad, F. Hutter, and M. Lindauer, "An evolution strategy with progressive episode lengths for playing games," in *Proc. 28th Int. Joint Conf. Artif. Intell.*, Aug. 2019, pp. 1234–1240.

[72] D. Rengarajan, G. Vaidya, A. Sarvesh, D. Kalathil, and S. Shakkottai, "Reinforcement learning with sparse rewards using guidance from offline demonstration," 2022, *arXiv:2202.04628*.

[73] T. Yang et al., "Exploration in deep reinforcement learning: A comprehensive survey," 2021, *arXiv:2109.06668*.

[74] P. Ladosz, L. Weng, M. Kim, and H. Oh, "Exploration in deep reinforcement learning: A survey," *Inf. Fusion*, vol. 85, pp. 1–22, Sep. 2022.

[75] S. Chakraborty, A. S. Bedi, A. Koppel, P. Tokekar, and D. Manocha, "Dealing with sparse rewards in continuous control robotics via heavy-tailed policies," 2022, *arXiv:2206.05652*.

[76] M. Andrychowicz et al., "Hindsight experience replay," in *Proc. NeurIPS*, vol. 30, 2017, pp. 1–11.

[77] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Mach. Learn.*, vol. 47, pp. 235–256, May 2002.

[78] D. Russo, B. Van Roy, A. Kazerouni, I. Osband, and Z. Wen, "A tutorial on Thompson sampling," 2017, *arXiv:1707.02038*.

[79] J. Achiam and S. Sastry, "Surprise-based intrinsic motivation for deep reinforcement learning," 2017, *arXiv:1703.01732*.

[80] D. Pathak, D. Gandhi, and A. Gupta, "Self-supervised exploration via disagreement," in *Proc. ICML*, 2019, pp. 5062–5071.

[81] H. Kim, J. Kim, Y. Jeong, S. Levine, and H. Oh Song, "EMI: Exploration with mutual information," 2018, *arXiv:1810.01176*.

[82] M. G. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos, "Unifying count-based exploration and intrinsic motivation," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 1–9.

[83] G. Ostrovski, M. G. Bellemare, A. Van Den Oord, and R. Munos, "Count-based exploration with neural density models," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 2721–2730.

[84] H. Tang et al., "#Exploration: A study of count-based exploration for deep reinforcement learning," in *Proc. NIPS*, 2017, pp. 1–10.

[85] M. C. Machado, M. G. Bellemare, and M. Bowling, "Count-based exploration with the successor representation," in *Proc. AAAI*, 2020, pp. 5125–5133.

[86] P. Dayan, "Improving generalization for temporal difference learning: The successor representation," *Neural Comput.*, vol. 5, no. 4, pp. 613–624, Jul. 1993.

[87] J. Schmidhuber, "Formal theory of creativity, fun, and intrinsic motivation (1990–2010)," *IEEE TAMD*, vol. 2, no. 3, pp. 230–247, Jul. 2010.

[88] R. Houthooft, X. Chen, Y. Duan, J. Schulman, F. De Turck, and P. Abbeel, "VIME: Variational information maximizing exploration," in *Proc. NIPS*, 2016, pp. 1–9.

[89] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, "Curiosity-driven exploration by self-supervised prediction," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, Jul. 2017, pp. 2778–2787.

[90] Y. Burda, H. Edwards, A. Storkey, and O. Klimov, "Exploration by random network distillation," 2018, *arXiv:1810.12894*.

[91] P. Shyam, W. Jaśkowski, and F. Gomez, "Model-based active exploration," in *Proc. ICML*, 2019, pp. 5779–5788.

[92] I. Osband, C. Blundell, A. Pritzel, and B. Van Roy, "Deep exploration via bootstrapped DQN," 2016, *arXiv:1602.04621*.

[93] R. Y. Chen, S. Sidor, P. Abbeel, and J. Schulman, "UCB exploration via Q-ensembles," 2017, *arXiv:1706.01502*.

[94] N. Savinov et al., "Episodic curiosity through reachability," 2018, *arXiv:1810.02274*.

[95] A. Ecoffet, J. Huizinga, J. Lehman, K. O. Stanley, and J. Clune, "Go-explore: A new approach for hard-exploration problems," 2020, *arXiv:1901.10995*.

[96] A. P. Badia et al., "Never give up: Learning directed exploration strategies," 2020, *arXiv:2002.06038*.

[97] A. P. Badia et al., "Agent57: Outperforming the Atari human benchmark," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 507–517.

[98] B. Eysenbach, A. Gupta, J. Ibarz, and S. Levine, "Diversity is all you need: Learning skills without a reward function," 2018, *arXiv:1802.06070*.

[99] A. Sharma, S. Gu, S. Levine, V. Kumar, and K. Hausman, "Dynamics-aware unsupervised discovery of skills," 2019, *arXiv:1907.01657*.

[100] J. Zhang, H. Tran, and G. Zhang, "Accelerating reinforcement learning with a directional-Gaussian-smoothing evolution strategy," 2020, *arXiv:2002.09077*.

[101] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training GANs," in *Proc. NeurIPS*, 2016, pp. 1–9.

[102] J. Geweke, "Antithetic acceleration of Monte Carlo integration in Bayesian inference," *J. Econ.*, vol. 38, nos. 1–2, pp. 73–89, May 1988.

[103] K. Choromanski, M. Rowland, V. Sindhwani, R. Turner, and A. Weller, "Structured evolution with compact architectures for scalable policy optimization," in *Proc. ICML*, 2018, pp. 970–978.

[104] N. Maheswaranathan, L. Metz, G. Tucker, D. Choi, and J. Sohl-Dickstein, "Guided evolutionary strategies: Augmenting random search with surrogate gradients," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 4264–4273.

[105] F. Meier, A. Mujika, M. M. Gauy, and A. Steger, "Improving gradient estimation in evolutionary strategies with past descent directions," 2019, *arXiv:1910.05268*.

[106] K. Choromanski, A. Pacchiano, J. Parker-Holder, and Y. Tang, "From complexity to simplicity: Adaptive es-active subspaces for blackbox optimization," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 1–11.

[107] F.-Y. Liu, Z.-N. Li, and C. Qian, "Self-guided evolution strategies with historical estimated gradients," in *Proc. 29th Int. Joint Conf. Artif. Intell.*, Jul. 2020, pp. 1474–1480.

[108] J. Zhang, H. Tran, D. Lu, and G. Zhang, "A novel evolution strategy with directional Gaussian smoothing for blackbox optimization," 202o, *arXiv:2002.03001*.

[109] J. Lehman and K. O. Stanley, "Novelty search and the problem with objectives," in *Genetic Program. theory Pract. IX*, 2011.

[110] J. K. Pugh, L. B. Soros, and K. O. Stanley, "Quality diversity: A new frontier for evolutionary computation," *Frontiers Robot. AI*, vol. 3, p. 40, Jul. 2016.

[111] O. S. Ajani and R. Mallipeddi, "Adaptive evolution strategy with ensemble of mutations for reinforcement learning," *Knowledge-Based Syst.*, vol. 245, Jun. 2022, Art. no. 108624.

[112] B. O'Donoghue, I. Osband, R. Munos, and V. Mnih, "The uncertainty Bellman equation and exploration," in *Proc. ICML*, 2018, pp. 3836–3845.

[113] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *Proc. ICML*, 2015, pp. 1889–1897.

[114] A. Gupta, R. Mendonca, Y. Liu, P. Abbeel, and S. Levine, "Meta-reinforcement learning of structured exploration strategies," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 1–10.

[115] S. Hochreiter, A. S. Younger, and P. R. Conwell, "Learning to learn using gradient descent," in *Proc. ICANN*, 2001, pp. 87–94.

[116] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. Lillicrap, "Meta-learning with memory-augmented neural networks," in *Proc. ICML*, 2016, pp. 1842–1850.

[117] Y. Duan, J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel, "Rl^2: Fast reinforcement learning via slow reinforcement learning," 2016, *arXiv:1611.02779*.

[118] J. X. Wang et al., "Learning to reinforcement learn," 2016, *arXiv:1611.05763*.

[119] J. G. Robles and J. Vanschoren, "Learning to reinforcement learn for neural architecture search," 2019, *arXiv:1911.03769*.

[120] M. Huisman, J. N. van Rijn, and A. Plaat, "A survey of deep meta-learning," *Artif. Intell. Rev.*, vol. 54, no. 6, pp. 4483–4541, Aug. 2021.

[121] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *Proc. ICML*, 2017, pp. 1126–1135.

[122] A. Nichol, J. Achiam, and J. Schulman, "On first-order meta-learning algorithms," 2018, *arXiv:1803.02999*.

[123] M. Botvinick, S. Ritter, J. X. Wang, Z. Kurth-Nelson, C. Blundell, and D. Hassabis, "Reinforcement learning, fast and slow," *Trends Cognit. Sci.*, vol. 23, no. 5, pp. 408–422, May 2019.

[124] A. Gajewski, J. Clune, K. O. Stanley, and J. Lehman, "Evolvability ES: Scalable and direct optimization of evolvability," in *Proc. Genetic Evol. Comput. Conf.*, Jul. 2019, pp. 107–115.

[125] H. Mengistu, J. Lehman, and J. Clune, "Evolvability search: Directly selecting for evolvability in order to study and produce it," in *Proc. Genetic Evol. Comput. Conf.*, Jul. 2016, pp. 141–148.

[126] A. Katona, D. W. Franks, and J. A. Walker, "Quality evolvability ES: Evolving individuals with a distribution of well performing and diverse offspring," 2021, *arXiv:2103.10790*.

[127] T. Hospedales, A. Antoniou, P. Micaelli, and A. Storkey, "Meta-learning in neural networks: A survey," 2020, *arXiv:2004.05439*.

[128] X. Song, W. Gao, Y. Yang, K. Choromanski, A. Pacchiano, and Y. Tang, "ES-MAML: Simple hessian-free meta learning," 2019, *arXiv:1910.01215*.

[129] X. Song et al., "Rapidly adaptable legged robots via evolutionary meta-learning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct. 2020, pp. 3769–3776.

[130] Z. Wang, C. Chen, and D. Dong, "Instance weighted incremental evolution strategies for reinforcement learning in dynamic environments," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, Mar. 29, 2022, doi: 10.1109/TNNLS.2022.3160173.

[131] J. Tan et al., "Sim-to-real: Learning agile locomotion for quadruped robots," 2018, *arXiv:1804.10332*.

[132] W. Zhu, X. Guo, D. Owaki, K. Kutsuzawa, and M. Hayashibe, "A survey of sim-to-real transfer techniques applied to reinforcement learning for bioinspired robots," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, Sep. 29, 2021, doi: 10.1109/TNNLS.2021.3112718.

[133] K. Arndt, M. Hazara, A. Ghadirzadeh, and V. Kyrki, "Meta reinforcement learning for sim-to-real domain adaptation," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2020, pp. 2725–2731.

[134] D. Lee, N. He, P. Kamalaruban, and V. Cevher, "Optimization for reinforcement learning: From a single agent to cooperative agents," *IEEE SP Mag.*, vol. 37, no. 3, pp. 123–135, May 2020.

[135] P. Hernandez-Leal, M. Kaisers, T. Baarslag, and E. Munoz de Cote, "A survey of learning in multiagent environments: Dealing with non-stationarity," 2017, *arXiv:1707.09183*.

[136] M. Tan, "Multi-agent reinforcement learning: Independent vs. cooperative agents," in *Proc. ICML*, 1993, pp. 330–337.

[137] A. O. Castaneda, "Deep reinforcement learning variants of multi-agent learning algorithms," Edinburgh, School Informat., Univ. Edinburgh, Edinburgh, Scotland, 2016.

[138] G. Palmer, K. Tuyls, D. Bloembergen, and R. Savani, "Lenient multi-agent deep reinforcement learning," 2017, *arXiv:1707.04402*.

[139] B. Kartal, J. Godoy, I. Karamouzas, and S. J. Guy, "Stochastic tree search with useful cycles for patrolling problems," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2015, pp. 1289–1294.

[140] P. Hernandez-Leal, B. Kartal, and M. E. Taylor, "A survey and critique of multiagent deep reinforcement learning," *Auto. Agents Multi-Agent Syst.*, vol. 33, no. 6, pp. 750–797, Nov. 2019.

[141] Y. Yang and J. Wang, "An overview of multi-agent reinforcement learning from game theoretical perspective," 2020, *arXiv:2011.00583*.

[142] P. Sunehag et al., "Value-decomposition networks for cooperative multi-agent learning," 2017, *arXiv:1706.05296*.

[143] T. Rashid, M. Samvelyan, C. Schroeder, G. Farquhar, J. Foerster, and S. Whiteson, "QMIX: Monotonic value function factorisation for deep multi-agent reinforcement learning," in *Proc. ICML*, 2018, pp. 1–14.

[144] J. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson, "Counterfactual multi-agent policy gradients," in *Proc. AAAI*, 2018, pp. 1–9.

[145] G. Chen, "A new framework for multi-agent reinforcement learning—Centralized training and exploration with decentralized execution via policy distillation," 2019, *arXiv:1910.09152*.

[146] L. Panait and S. Luke, "Cooperative multi-agent learning: The state of the art," *Auto. Agents Multi-Agent Syst.*, vol. 11, no. 3, pp. 387–434, Nov. 2005.

[147] M. Lauer and M. Riedmiller, "An algorithm for distributed reinforcement learning in cooperative multi-agent systems," in *Proc. ICML*, 2000, pp. 1–7.

[148] A. Tampuu et al., "Multiagent cooperation and competition with deep reinforcement learning," *PLoS ONE*, vol. 12, no. 4, Apr. 2017, Art. no. e0172395.

[149] J. Foerster et al., "Stabilising experience replay for deep multi-agent reinforcement learning," in *Proc. ICML*, 2017, pp. 1146–1155.

[150] A. OroojlooyJadid and D. Hajinezhad, "A review of cooperative multi-agent deep reinforcement learning," 2019, *arXiv:1908.03963*.

[151] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," 2017, *arXiv:1706.02275*.

[152] X. Chu and H. Ye, "Parameter sharing deep deterministic policy gradient for cooperative multi-agent reinforcement learning," 2017, *arXiv:1710.00336*.

[153] H. Ryu, H. Shin, and J. Park, "Multi-agent actor-critic with generative cooperative policy network," 2018, *arXiv:1810.09206*.

[154] H. Mao, Z. Zhang, Z. Xiao, and Z. Gong, "Modelling the dynamic joint policy of teammates with attention multi-agent DDPG," 2018, *arXiv:1811.07029*.

[155] R. E. Wang, M. Everett, and J. P. How, "R-MADDPG for partially observable environments and limited communication," 2020, *arXiv:2002.06684*.

[156] T. Kasai, H. Tenmoto, and A. Kamiya, "Learning of communication codes in multi-agent reinforcement learning problem," in *Proc. IEEE Conf. Soft Comput. Ind. Appl.*, Jun. 2008, pp. 1–9.

[157] C. L. Giles and K.-C. Jim, "Learning communication for multi-agent systems," in *Proc. Workshop Radical Agent Concepts*, 2002, pp. 377–390.

[158] J. N. Foerster, Y. M. Assael, N. de Freitas, and S. Whiteson, "Learning to communicate to solve riddles with deep distributed recurrent Q-networks," 2016, *arXiv:1602.02672*.

[159] S. Sukhbaatar, A. Szlam, and R. Fergus, "Learning multiagent communication with backpropagation," 2016, *arXiv:1605.07736*.

[160] J. K. Gupta, M. Egorov, and M. Kochenderfer, "Cooperative multi-agent control using deep reinforcement learning," in *Proc. AAMAS*, 2017, pp. 66–83.

[161] F. L. D. Silva and A. H. R. Costa, "A survey on transfer learning for multiagent reinforcement learning systems," *J. Artif. Intell. Res.*, vol. 64, pp. 645–703, Mar. 2019.

[162] W. Du and S. Ding, "A survey on multi-agent deep reinforcement learning: From the perspective of challenges and applications," *Artif. Intell. Rev.*, vol. 54, no. 5, pp. 3215–3238, Jun. 2021.

[163] M. Hiraga, Y. Wei, T. Yasuda, and K. Ohkura, "Evolving autonomous specialization in congested path formation task of robotic swarms," *Artif. Life Robot.*, vol. 23, no. 4, pp. 547–554, Dec. 2018.

[164] M. Hiraga, Y. Wei, and K. Ohkura, "Evolving collective cognition of robotic swarms in the foraging task with poison," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Jun. 2019, pp. 3205–3212.
[165] Y. Tang, J. Tan, and T. Harada, "Learning agile locomotion via adversarial training," 2020, *arXiv:2008.00603*.
[166] J. Chen and Z. Gao, "A framework for learning predator-prey agents from simulation to real world," 2020, *arXiv:2010.15792*.
[167] G. Li, Q. Duan, and Y. Shi, "A parallel evolutionary algorithm with value decomposition for multi-agent problems," in *Proc. ICSI*, 2020, pp. 616–627.
[168] J. R. Martínez and F. A. Gregori, "Comparison of evolutionary strategies for reinforcement learning in a swarm aggregation behaviour," in *Proc. MLMI*, 2020, pp. 40–45.
[169] D. D. Fan, E. A. Theodorou, and J. Reeder, "Model-based stochastic search for large scale optimization of multi-agent UAV swarms," in *Proc. IEEE Symp. Ser. Comput. Intell. (SSCI)*, Nov. 2018, pp. 2216–2222.
[170] F. Aznar, M. Pujol, and R. Rizo, "Learning a swarm foraging behavior with microscopic fuzzy controllers using deep reinforcement learning," *Appl. Sci.*, vol. 11, no. 6, p. 2856, Mar. 2021.
[171] F. Petroski Such, V. Madhavan, E. Conti, J. Lehman, K. O. Stanley, and J. Clune, "Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning," 2017, *arXiv:1712.06567*.
[172] S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2017, pp. 3389–3396.
[173] A. Pourchot and O. Sigaud, "CEM-RL: Combining evolutionary and gradient-based methods for policy search," 2018, *arXiv:1810.01222*.
[174] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *Proc. ICML*, 2018, pp. 1587–1596.
[175] K. Hansel, J. Moos, and C. Derstroff, "Benchmarking the natural gradient in policy gradient methods and evolution strategies," in *Reinforcement Learning Algorithms: Analysis and Applications*. Cham, Switzerland: Springer, 2021, pp. 1587–1596.
[176] P. Ecoffet, N. Fontbonne, J.-B. André, and N. Bredeche, "Policy search with rare significant events: Choosing the right partner to cooperate with," *PLoS ONE*, vol. 17, no. 4, Apr. 2022, Art. no. e0266841.
[177] A. Stafylopatis and K. Blekas, "Autonomous vehicle navigation using evolutionary reinforcement learning," *Eur. J. Oper. Res.*, vol. 108, no. 2, pp. 306–318, Jul. 1998.
[178] M. Jaderberg et al., "Human-level performance in 3D multiplayer games with population-based reinforcement learning," *Science*, vol. 364, no. 6443, pp. 859–865, May 2019.
[179] Y. Shao et al., "Multi-objective neural evolutionary algorithm for combinatorial optimization problems," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 34, no. 4, pp. 2133–2143, Apr. 2023.
[180] S. Khadka and K. Tumer, "Evolutionary reinforcement learning," 2018, *arXiv:1805.07917*.
[181] V. Shopov and V. Markova, "A study of the impact of evolutionary strategies on performance of reinforcement learning autonomous agents," in *Proc. ICAS*, 2018, pp. 56–60.
[182] R. Houthooft et al., "Evolved policy gradients," in *Proc. NeurIPS*, 2018, pp. 1–10.
[183] D. Chen, Y. Wang, and W. Gao, "Combining a gradient-based method and an evolution strategy for multi-objective reinforcement learning," *Applied Intelligence*, vol. 50, pp. 3301–3317, Oct. 2020.
[184] T. De Bruin, J. Kober, K. Tuyls, and R. Babuška, "Fine-tuning deep RL with gradient-free optimization," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 8049–8056, 2020.
[185] X. Song et al., "ES-ENAS: Combining evolution strategies with neural architecture search at no extra cost for reinforcement learning," 2021, *arXiv:2101.07415*.
[186] F. Ferreira, T. Nierhoff, and F. Hutter, "Learning synthetic environments for reinforcement learning with evolution strategies," 2021, *arXiv:2101.09721*.
[187] K. V. Moffaert, M. M. Drugan, and A. Nowé, "Hypervolume-based multi-objective reinforcement learning," in *Proc. EMO*, 2013, pp. 352–366.
[188] S. Parisi, M. Pirotta, N. Smacchia, L. Bascetta, and M. Restelli, "Policy gradient approaches for multi-objective sequential decision making," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2014, pp. 2323–2330.
[189] A. Nagabandi, K. Konolige, S. Levine, and V. Kumar, "Deep dynamics models for learning dexterous manipulation," in *Proc. Conf. Robot Learn.*, 2020, pp. 1101–1112.
[190] J. Tebbe, L. Krauch, Y. Gao, and A. Zell, "Sample-efficient reinforcement learning in robotic table tennis," 2020, *arXiv:2011.03275*.
[191] A. Pourchot, N. Perrin, and O. Sigaud, "Importance mixing: Improving sample reuse in evolutionary policy search methods," 2018, *arXiv:1808.05832*.
[192] O. Sigaud and F. Stulp, "Policy search in continuous action domains: An overview," *Neural Netw.*, vol. 113, pp. 28–40, May 2019.
[193] Y. Sun, D. Wierstra, T. Schaul, and J. Schmidhuber, "Efficient natural evolution strategies," in *Proc. 11th Annu. Conf. Genetic Evol. Comput.*, Jul. 2009, pp. 539–546.
[194] R. Portelas, C. Colas, L. Weng, K. Hofmann, and P.-Y. Oudeyer, "Automatic curriculum learning for deep RL: A short survey," 2020, *arXiv:2003.04664*.