Deep Reinforcement Learning Versus Evolution Strategies: A Comparative Survey

Amjad Yousef Majid
Delft University of Technology
a.y.majid@tudelft.nl

Serge Saaybi
Delft University of Technology
s.c.e.saaybi@student.tudelft.nl

Tomas van Rietbergen
Delft University of Technology
T.L.vanRietbergen@student.tudelft.nl

Vincent Francois-Lavet
VU Amsterdam
vincent.francoislavet@vu.nl

R Venkatesha Prasad
Delft University of Technology
r.r.venkateshaprasad@tudelft.nl

Chris Verhoeven
Delft University of Technology
C.J.M.Verhoeven@tudelft.nl

***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 multi-agent learning. Then, the benefits of hybrid algorithms that combine concepts from DRL and ESs are highlighted. Finally, to have an indication about how they compare in real-world applications, a survey of the literature for the set of applications they support is provided.**

***Index Terms*—Deep Reinforcement Learning, Evolution Strategies, Multi-agent**

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 race for developing artificial general intelligence, these two phenomena have motivated the development of two 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 how the mechanism that let intelligence emerge in the biological world—repeatedly selecting the best performing individuals.

In this paper, we discuss RL and ESs together analyzing their strengths and weaknesses regarding their sequential decision-making capabilities and shed light on potential directions for further development.

The RL framework is formalized as an agent acting on an environment with the goal of maximizing a cumulative reward over the trajectory of interaction with the environment [1]. Imagine playing a table tennis game (environment) with a robot (agent). The robot has not explicitly been programmed to play the game. Instead, it can observe the score of the game (rewards). The robot's goal is to maximize its score. For that purpose, it tries different techniques of hitting the ball (actions), observes the outcome, and gradually enhances its playing strategy (policy).

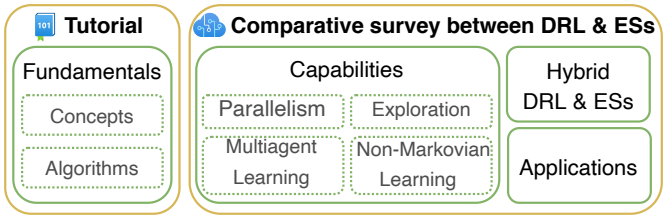
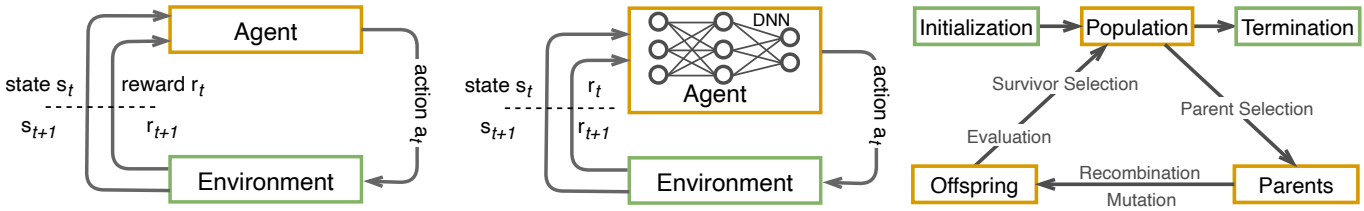


Fig. 1: The structure of the survey

Despite the proven convergence of RL algorithms to optimal policies—best solutions to the problems at hand—they face difficulties processing high-dimensional data (e.g., images). To tackle problems with high-dimensional data, RL algorithms are nowadays often combined with deep neural networks, giving raise to a whole field of research known as Deep RL (DRL) [2].

As a contrasting approach to DRL, ES algorithms utilize a random process to iteratively generate candidate solutions. Then, they evaluate these solutions and bias the search in direction of the best scoring ones [3]. In recent years, ESs have seen an increase in popularity and has been successfully applied to several applications, including optimizing objective functions for many RL tasks [4, 5].

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, the ability to learn in dynamic environments and from an application standpoint (Figure 1). We also discuss how combining DRL and ESs in hybrid systems can leverage the advantages



(a) RL (b) DRL (c) ESs

Fig. 2: Iteration loops of (Deep) Reinforcement Learning and Evolutionary Strategies.

of both approaches.

To date, there have been different papers summarizing different features of DRL and ESs. For example, derivative-free reinforcement learning (e.g., ESs) has been reviewed in [6], covering aspects such as scalability and exploration. A survey related to DRL for autonomous driving is provided in [7], and the challenges, solutions, and applications of multi-agent DRL systems are reviewed in [8]. However, contrasting with prior work, our paper surveys the literature with a bird's-eye view, focusing on the main developmental directions instead of individual algorithms.

The rest of the paper is organized as follows: Section II presents the fundamental architectural concepts behind RL and ESs; Section III summarizes fundamental algorithms of RL, DRL and ESs; Sections IV-A, IV-B, IV-C and IV-D compare the capabilities of DRL and ESs; In Section V, we present hybrid systems that combine DRL and ESs. Section VI compares them from an applications' point of view. Section VII outlines open challenges and potential research directions. Finally, we conclude the paper in Section VIII. The main takeaways of each section are summarized in a concise subsection titled "Comparison".

II. Fundamentals

This section covers the fundamental elements of DRL and ESs, including formal definitions and the main algorithmic families.

A. Reinforcement Learning

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 when interacting with an environment (Figure 2a), which is generally modeled as a Markov Decision Process (MDP). An MDP is defined by the tuple $(\mathcal{S}, \mathcal{A}, T, R)$, 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'.

The total *return* starting from time t until the end of the interaction between an agent and its environment is expressed as

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

where R_t and is a random variable that models the immediate reward, r, and $\gamma \in [0, 1)$ is a discount factor that weights the immediate and future rewards. *Value functions* are the expected return of being in a state or taking a particular action. The state-value function $v_\pi(s)$ gives the expected return from state s 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) $q^\pi(s, a)$ is the expected return of taking action a in state s and following policy π thereafter,

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

The action selection process of an agent is governed by its policy, which in the general stochastic case yields an action according to a probability distribution over the action space conditioned on a given state $\pi(s, a)$.

There are four main RL algorithmic families:

Policy-based Algorithms. A policy-based algorithm optimizes and memorizes a policy explicitly, that is, it directly searches the policy space for an (approximate) optimal policy, π^*. Examples of such algorithms are policy iteration [9], policy gradient [10] and REINFORCE [11]. Policy-based algorithms can be applied to any type of action space: continuous, discrete or a mixture (multiactions). However, these algorithms generally have high variance and are sample-inefficient.

Value-based Algorithms. A value-based algorithm learns a value function, $v^\pi(s)$ or $q^\pi(s, a)$. Then, a policy is extracted according to the learned value function. Examples of such algorithms are value iteration [12], SARSA [13], Q-learning and DQN [14]. Value-based algorithms are more sample-efficient than policy-based ones. However, under ordinary circumstances the convergence of these algorithms is not guaranteed.

Actor-critic-based Algorithms. The actor-critic approach tries to combine the strengths of policy- and value-based algorithms into a single algorithmic architecture [15]. The

actor is a policy-based algorithm that tries to learn the optimal policy, whereas the critic is a value-based algorithm that evaluates the actions taken by the actor.

Model-based Algorithms. All of the algorithmic families mentioned previously concern *model-free* algorithms. In contrast, model-based algorithms learn or make use of a model of the transition dynamics of an environment. Once an agent has access to such a model, it can use it to "imagine" the consequences of taking a particular set of actions without acting on the environment. Such capability enables an RL agent to evaluate the expected actions of an opponent in games [16, 17] and to make better use of gathered data, which is very useful in tasks such as controlling a robot [18]. However, for many problems, it is difficult to produce close to reality models.

Deep Reinforcement Learning (DRL) refers to the combination of Deep Learning (DL) and RL (Figure 2b) [2]. DRL uses DNNs to approximate one of the learnable functions of RL. Correspondingly, there are three main families of DRL algorithms: value-based, policy-based, and model-based [14, 16, 19]. For example, the DNN of a policy-based DRL agent takes the state of the environment as input and produces an action as output (Figure 2b). The action selection process is governed by the parameters $\boldsymbol{\theta}$ of the DNN. The parameters selection is optimized using a backpropagation algorithm during the training phase.

B. Evolution Strategies

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 [20, 21]. ESs do not require modeling the problem as an MDP, neither the objective function $f(\boldsymbol{x})$ has 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 towards 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 (i.e., the mean m and the standard deviation σ) is described by what are called *strategic parameters*. These can be modified online to make the search process more efficient. The generic ESs process is shown in Figure 2c and its elements are explained below:

1) *Initialization*: the algorithm generates an initial population P consisting of μ individuals.
2) *Parent selection*: a sub-set of the population is selected to function as parents during the recombination step.
3) *Reproduction* 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 is added to the recombination results. A common way of implementing mutation is to sample from a multivariate normal distribution centered around the mean obtained from the previous recombination step:
   $$\boldsymbol{x}_k^{g+1} \sim \mathcal{N}(\boldsymbol{m}^{(g)}, \sigma^{(g)} I) = \boldsymbol{m}^{(g)} + \sigma^{(g)} \mathcal{N}(0, I),$$
   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.

The idea of employing ESs as an alternative to RL is not new [22, 23, 24, 25], but recently it has seen a renewed interest (e.g. [4, 26]).

C. Comparison

Our main takeaways of the above fundamental concepts are:

- The objective of an RL algorithm is to maximize the sum of discounted rewards, whereas an ESs algorithm does not require such formulation. However, the objective for RL settings can be converted to ESs settings with a terminal state that provides a reward equivalent to the fitness function.
- The problem setup differs between RL and ESs. An ESs 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.
- An ESs algorithm aims at finding candidate solutions that optimize a fitness function, whereas the goal of DRL is to keep advancing one or two function approximators which in turn need to optimize the equivalent of the fitness function, usually defined by the discounted return.
- The ESs approach is most similar to the policy-based DRL approach: both aim at finding parameters in a search space such that the resulting parameterized function optimizes certain objectives (expected return for DRL or fitness score for ESs). The main distinction is that ESs, unlike DRL, do not calculate gradients nor use backpropagation.
- 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.

III. Fundamental Algorithms

Fundamental algorithms of (D)RL and ESs are introduced in this section.

A. Reinforcement Learning Algorithms

SARSA is a model-free algorithm that leverages temporal-differences for prediction [1]. It updates the Q-value, $Q(s_t, a_t)$, while following a policy. The interaction between the agent and environment results in the following sequence $\ldots, s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1}, \ldots$: the agent takes an action a_t

TABLE I: Fundamental (Deep) Reinforcement Learning and Evolution Strategies algorithms

Algorithm	Classification	Action Space	Memory Consumed	Limitations	Backprop.	Ref.
SARSA	on-policy value-based RL	discrete	exponential in state and action spaces	tackling continuous space, does not generalize between similar states	X	[1]
Q-learning	off-policy value-based RL	discrete	exponential in state and action spaces	tackling continuous space, does not generalize between similar states	X	[27]
REINFORCE	policy-based RL	discrete/continuous	typically, it requires storing DNN parameters	data inefficiency, higher variance compared to DQN	✓	[11]
DQN	off-policy value-based RL	discrete	it requires storing DNN parameters and a replay buffer	the learning of the Q-function can suffer from instabilities	✓	[28] [2]
CMA-ES	black-box ES optimization	discrete/continuous	high memory requirement	high space and time complexity when dealing with large scale optimization problems	X	[29] [30]
NES & OpenAI-ES	black-box ES optimization	discrete/continuous	less memory usage than CMA-ES	data inefficiency due to gradient approximation	X	[31] [32]

while being in a state s_t, and consequently, the environment transitions to a state s_{t+1} and the agent observes a reward r_{t+1}. For action selection, SARSA uses ε-greedy algorithm, which selects the action with maximum $Q(s_t, a_t)$ with probability of $1-\varepsilon$, and otherwise, it draws an action uniformly from $\mathcal{A}$. SARSA is an on-policy algorithm, that is, it evaluates and improves the same policy that selects the taken actions. SARSA's update equation is

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\big[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\big], \quad (3)$$

where α is the learning rate.

Q-Learning [1] is similar to SARSA with a key difference: It is an off-policy algorithm, which means that it learns an optimal Q-value function from data obtained via any policy (without introducing a bias). In particular, The Q-learning update rule compares the Q-value of the current state-action pair, $Q(s_t, a_t)$, with a pair from the next state that has the maximum Q-value, $Q(s_{t+1}, a_{t+1})$, which is not necessarily the one chosen by ε-greedy as in SARSA. The update rule of Q-learning is

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\big[r_{(t+1)} + \gamma \max_a Q(s_{(t+1)}, a_{(t+1)}) - Q(s_t, a_t)\big]. \quad (4)$$

Off-policy algorithms are more data-efficient than on-policy ones, because they can use the collected data repeatedly.

REINFORCE [11] is a fundamental stochastic gradient descent algorithm for policy gradient algorithms. It leverages a DNN to approximate the policy π and update its parameters $\boldsymbol{\theta}$. The network receives an input from the environment and outputs a probability distribution over the action space, $\mathcal{A}$. The steps involved in the implementation of REINFORCE are:

1) Initialize a Random Policy (i.e., the parameters of a DNN)
2) Use the policy $\pi_{\boldsymbol{\theta}}$ to collect a trajectory $\tau = (s_0, a_0, r_1, s_1, a_1, r_2, ..., a_H, r_{H+1}, s_{H+1})$
3) Estimate the return for this trajectory
4) Use the estimate of the return to calculate the policy gradient:

$$\nabla_{\boldsymbol{\theta}} \mathcal{J}(\boldsymbol{\theta}) = \mathbb{E}_{\pi_{\boldsymbol{\theta}}}[\nabla \log \pi(a|s;\boldsymbol{\theta}) Q_\pi(s,a)] \quad (5)$$

5) Adjust the weights $\boldsymbol{\theta}$ of the Policy: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \nabla_\theta J(\boldsymbol{\theta})$
6) Repeat from step 2 until termination.

Deep Q-network (DQN) [28] combines Q-learning with a convolutional neural network (CNN) [33] to act in environments with high-dimensional input spaces (e.g., images of Atari games). It gets a state (e.g., a mini-batch of images) as input and produces Q-values of all possible actions. The CNN is used to approximate the optimal action-value function (or Q-function). Such usage, however, causes the DRL agent to be unstable [34]. To counter that, DQN samples an experience replay [35] dataset $D_t = \{(s_1, a_1, r_2, s_2), \ldots, (s_t, a_t, r_{t+1}, s_{t+1})\}$ and uses a target network that is updated only after a certain number of iterations. To update the network parameters at iteration i, DQN uses the following loss function

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')} \sim U(D) \left[\left(r + \gamma \max_{a'} Q(s', a'; \overline{\boldsymbol{\theta}_i}) - Q(s', a'; \boldsymbol{\theta}_i) \right)^2 \right] \quad (6)$$

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.

B. Evolutionary Strategies Algorithms

The (1+1)**-ES** (one parent, one offspring) is the simplest ES conceived by Rechenberg [36]. 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:

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

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 $\frac{1}{5th}$ success rule [37, 38]: if $\mathbf{x}_o$ is fitter than $\mathbf{x}_p$ $\frac{1}{5th}$ of the times then σ should stay the same; if $\mathbf{x}_o$ is fitter *more* than $\frac{1}{5th}$ of the times then σ should be increased, and otherwise it should be decreased.

The $(\mu/\rho^+,\lambda)$**-ES** was originally proposed by Schwefel [39] 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 for the next generation 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. The ρ in the name of the algorithm 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 Evolution Strategies (CMA-ES) is one of the most popular gradient-free optimisation algorithms [40, 41, 42, 43]. To search a solution space, it samples a population, λ, of new search points (offsprings) from a multivariate normal distribution:

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

where g is the generation number (i.e., $g = 1, 2, 3, \ldots$), $\boldsymbol{x}_i \in \mathbb{R}^n$ is the i-th 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 towards the best candidate solutions of the current population. It automatically adapts the stepsize σ (the mutation strength) using the Cumulative Stepsize Adaption (CSA) algorithm [40] 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 towards promising directions, CMA-ES updates the covariance matrix in each iteration. The update consists of two main parts: (i) rank-1 update, which computes an evolution path for the mutation distribution means, similarly to the step-size evolution path; and (ii) 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 [43] for a more detailed explanation).

Natural Evolution Strategies (NES) is similar in many ways to the previously defined ES algorithms, the core idea behind it relates to the use of gradients to adequately update a search distribution [44]. The basic idea behind NES consists of:

- *Sampling:* NES samples its individuals from a probability distribution (usually a Gaussian distribution) over the search space. The end goal of NES is to update the distribution parameters $\boldsymbol{\theta}$ to maximize the average fitness $F(\boldsymbol{x})$ of the sampled individuals $\boldsymbol{x}$.
- *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.
- *Gradient ascent:* NES computes gradient ascent along the estimated gradient
- Iterates over the previous steps until a stopping criterion is met [44].

Salimans et al. [4] proposed a variant of NES for optimizing the policy parameters θ. As gradients are unavailable, they are estimated via gaussian smoothing of the objective function $F(X)$ which represents the expected return.

C. Comparison

Our main observations of the fundamental algorithms are:

- Both ES and on-policy RL algorithms are data inefficient: on-policy algorithms make use of data that is generated from the current policy and discard older data; ES discard all but a sub-set of candidate solutions in each iteration.
- The computation requirements per iteration of ESs are often lower than that of DRL as it does not require backpropagating error values.
- Value-based DRL algorithms such as DQN can be data-efficient because they can work with a replay memory that allows a reuse of off-policy data. However, they can become unstable for long horizons and high discount factors [45].
- Policy-based RL and ESs are similar in that they both search for good policies directly.
- Table I highlights important characteristics of the mentioned algorithms.

IV. Deep Reinforcement Learning Versus Evolution Strategies

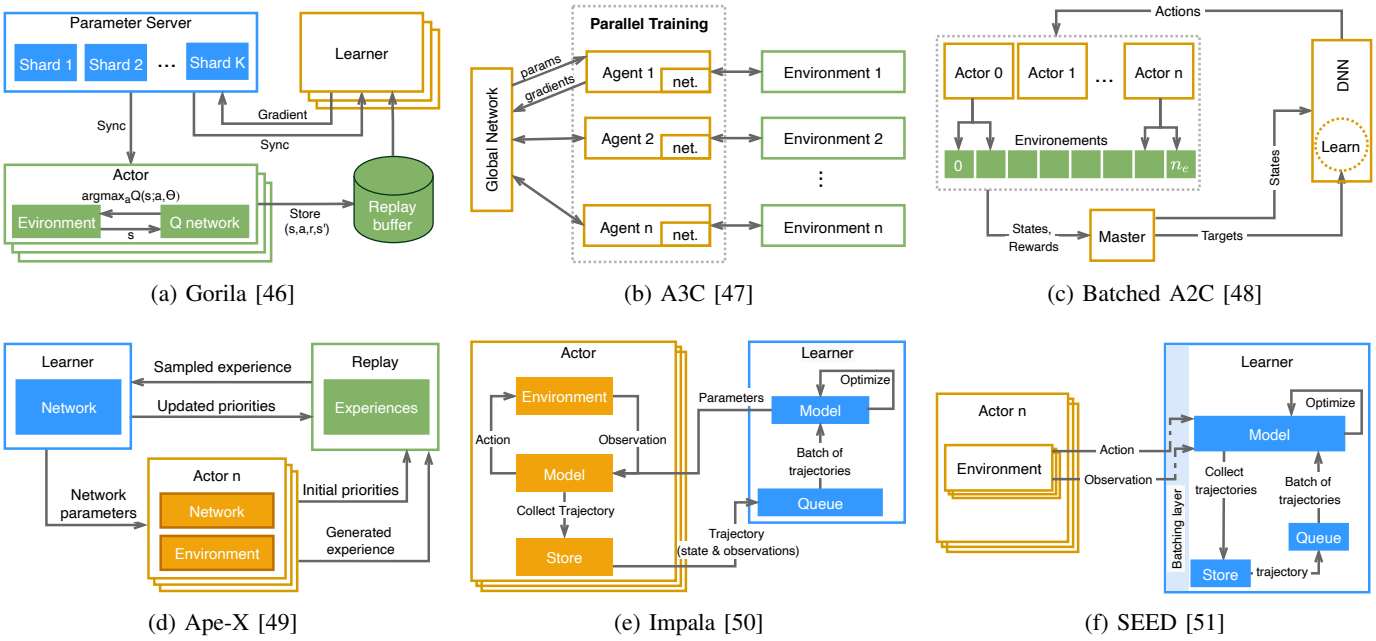
This section compares different aspects of DRL and ESs, such as their ability to parallelize computations, explore an environment, and learn in multi-agent and dynamic settings.

A. Parallelism

Despite the success of DRL and ESs, they are still computationally intensive approaches to tackle sequential decision-making problems. Parallel execution is thus an important approach to speed up the computation [46]. Below, we look into the rich literature of parallel DRL and ES algorithms.

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



(a) Gorila [46] (b) A3C [47] (c) Batched A2C [48]

(d) Ape-X [49] (e) Impala [50] (f) SEED [51]

Fig. 3: Parallel Deep Reinforcement Learning algorithms architectures

optimize a global network (Figure 3) [52, 53]. The rest of this section presents important parallel-DRL algorithms.

Gorila [46] is the first massively distributed architecture for DRL. It consists of four major components: actors, learners, a parameter server, and a replay buffer (Figure 3a). Each actor has its Q-network. It interacts with an instance of the same environment and stores the generated experiences (i.e., a set of $\{s, a, r, s'\}$) in the replay buffer. Learners sample the experience replay buffer and use DQN to compute gradients. Sampling from a buffer reduces the correlation between data updates and the effect of non-stationarity in the data. These gradients are then sent asynchronously to the parameter server to update its Q-network. After that, the parameter server updates the actors' and learners' Q-networks to synchronize the learning process.

A3C & GA3C. While using a replay buffer helps in stabilizing the learning process, it requires additional memory and computational power and can only be used with off-policy algorithms. Motivated by these limitations, Mnih et al. [47] introduced the Asynchronous Advantage Actor-Critic (A3C) as an alternative to Gorila. A3C consists of a global network and multiple agents each with its own network (Figure 3b). The agents are implemented as CPU threads within a single machine, which reduces the communication cost imposed by distributed systems such as Gorila. The agents interact in parallel with their independent copy of the environment. Each agent calculates the value and the policy gradients which are used to update the global network parameters. This method of learning diversifies and decorrelates data updates which stabilize the learning process. GA3C [54] makes use of GPUs and shows better scalability and performance than A3C.

Batched A2C & DPPO. A downside of A3C is that asynchronous updates may lead to sub-optimal collective updates to the global network. To overcome this, Batched Advantage Actor-Critic (Batched A2C) employs a master node (or a coordinator) to synchronize the update process of the global network [48]. Batched A2C tries to capitalize on 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 [54]. Figure 3c presents the Batched A2C architecture. At each time step, Batched A2C samples from the policy and generates a batch of actions for n_w workers on n_e environment instances. 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) [55] features architecture similar to that of A2C, and uses the PPO [56] algorithm for learning.

Ape-X & R2D2. Ape-X [49] 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 (Figure 3d). 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 [28] and DDPG [19]. R2D2 [57] has a similar architecture but outperforms Ape-X using recurrent neural network (RNN)-based RL agents.

IMPALA. Due to the use of an on-policy method in an

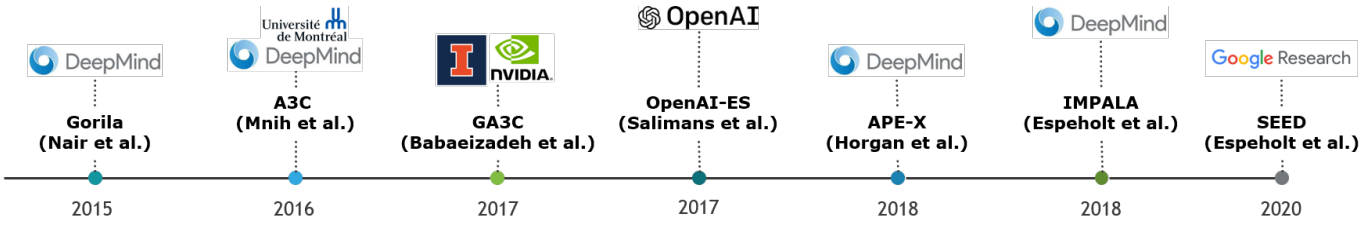


Fig. 4: Parallel Deep Reinforcement Learning and Evolution Strategies algorithms shown on a timeline

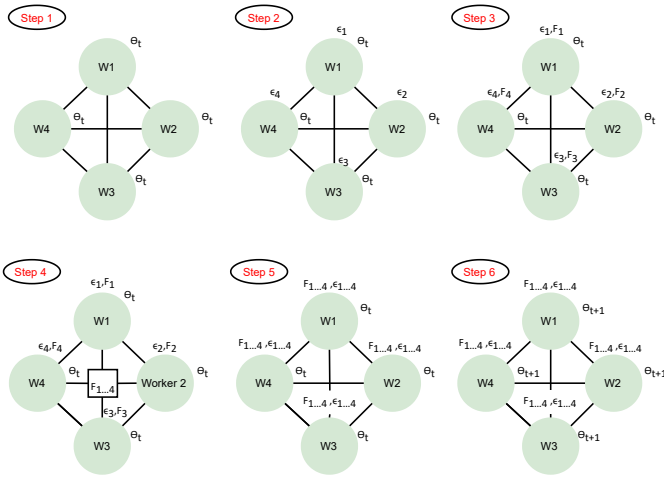


Fig. 5: Parallelization steps in OpenAI-ES [4]

off-policy setting GA3C [54] suffers from poor convergence. IMPALA [50] 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 (Figure 3e). However, different from A3C's actors, IMPALA's actors send the gathered experiences (instead of the gradients) to the learner. The learner then utilizes these experiences to optimizes its policy and value functions. After that, it updates the actors' models parameters. The separation between acting and learning and V-trace enable IMPALA to have stable learning while achieving high throughput. When training a very deep model the speed of a single GPU is often the bottleneck. To overcome this challenge, IMPALA (in addition to a single learner) supports multiple synchronized learners.

SEED. SEED [51] improves on the IMPALA system by moving inference to the learner (Figure 3f). 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 to DRL, ES algorithms require significantly less bandwidth to parallelize a given task. Salimans et al. [4] proposed OpenAI-ES: an algorithm derived from NES (see Section III) 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 illustrated in Figure 5 and summarized as follows:

1) Sample a Gaussian noise vector, $\varepsilon_i \sim N(0, I)$.
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 $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t + \alpha\frac{1}{n\sigma}\sum_{j=1}^{n} F_j\varepsilon_j$, where $\boldsymbol{\theta}$ is a weighted vector of a DNN.
6) Repeat from step 2 until termination.

Several researchers proposed algorithms inspired by OpenAI-ES [4]. For example, Conti et al. [32] proposed Novelty Search Evolution Strategy (NS-ES) algorithm which hybridizes OpenAI-ES with Novelty Search (NS)—a directed exploration algorithm. The authors also introduced a variant of NS-ES by replacing NS with Quality Diversity (QD) algorithm. Their results show that the NS- and QD-based algorithms improve ES algorithms performance on RL tasks with sparse rewards, as they help avoid local optima. Liu et al. [58] proposed Trust Region Evolution Strategies (TRES). TRES is more sampled data efficient than classical ESs. It optimizes a surrogate objective function that enable reusing sampled data multiple times. TRES utilizes random seeds sharing introduced by [4] to achieve extremely low bandwidth. Finally, Fuks et al. [29] proposed Evolution Strategy with Progressive Episode Lengths (PEL). The main idea of PEL is to allow an agent to do small and easy tasks to gain knowledge quickly and then to use this knowledge to tackle more complex tasks. PEL leverages the same parallelization idea as OpenAI-ES [4] and shows a great improvement over canonical ES algorithms.

3) Comparison

Our observations about parallelizing DRL and ES are:

- Despite the additional complexity, parallelism accelerates the execution of DRL and ES algorithms.
- Parallel DRL usually communicates network parameters or gradient vectors between nodes, while parallel ES share only scalar values between workers.
- Table II snapshots the main characteristics of the presented algorithms, and Figure 4 shows how parallel DRL and ES algorithms evolved over time.

TABLE II: Parallelized Deep Reinforcement Learning and Evolution Strategies 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	[46] [50]
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	[47] [50]
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	[48]
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	[49]
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	[50]
SEED	multi-actors and a single learning; GPU/TPU	surpasses the performance of IMPALA	centralized inference may lead to increase latency	[51]
OpenAI ES	set of parallel workers; CPUs	outperforms other solution on most Atari games with less training: better than A3C in 23 games and worse in 28	evaluates many episodes requiring a lot of CPU time: 4000 CPU hours for a single ES run	[4] [5]

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 the optimal policy, and when should it use its current optimal policy estimation to make useful progress? This dilemma has attracted ample attention. Below, we summarize the main exploration methods in DRL and ESs.

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. This is the case for the well-known ϵ-greedy exploration algorithm [1] that 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) [59] does that by making the reward signal equals the estimated value of a Q-function plus a value that reflects the algorithm's lack of confidence about this estimate,

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

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 method (TS) [60] 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 the optimal policy over time. Therefore, they are inherently more efficient than ϵ-greedy.

From RL to DRL. DRL agents act on environments with continuous or high-dimensional state-action spaces (e.g., Montezuma's Revenge, 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 [61, 62, 63, 64]. Below we outline important DRL exploration algorithms.

pseudo-count methods. To extend count-based exploration methods (e.g., UCB) to DRL settings, Bellemare et al. [65] 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 to incentive the agent to visit less-visited states. Ostrovski et al. [66] 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 [67] and counting the hashing codes instead of states. Then a reward bonus similar to that of UCB is designed utilizing the hash code counts.

Approximate posterior sampling. Inspired by TS, Osband et al. [68] introduced Bootstrapped DQN. Bootstrapped DQN trains a DNN with N bootstrapped heads to approximate a distribution over Q-functions (bootstrapping is the process of approximating a distribution by sampling with replacement from a population multiple times and then aggregating these samples). 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. [69] integrates UCB with Bootstrapped DQN by calculating the mean and variance of a subset of the ensemble Q-functions. O'Donoghue et al. [70] combined TS with uncertainty Bellman equations to propagate the uncertainty in the Q-values over multiple timesteps.

Information gain. In exploration based on information gain, the algorithm provides a reward bonus proportional to the information obtained after taking an action. This reward bonus is then added to the reward provided by the environment to push the agent to explore novel (or less known) states [78]. Houthooft et al. [72] 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. [73] 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 next state. To make this method effective, the authors utilized an inverse model, removing irrelevant -for the comparison- state features. Burda et al. [79] 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.

Memory-based. Savinov et al. [74] 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 some effort to be reached (effort is materialized by the number of environment steps taken to reach an observation). Ecoffet et al. [75] introduced Go-explore: an RL agent that aims to solve hard exploration problems such as Montezuma's Revenge and Pitfall. Go-explore runs in two phases. In phase one, the agent explore randomly, remembers interesting states and continues (after reset) random exploration from one of the interesting states (the authors assume the agent can deterministically go back to an interesting state). After finding a solution to the problem, phase two begins where the Go-explore agent robustifies its the best found solution by randomizing the environment and running imitation learning using the best solution. Badia et al. [76] 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: (i) An episodic reward, which enables the agent to quickly adapt within an episode, and, (ii) the life-long novelty reward, which down-modulates states that become familiar across many episodes. Further, NGU uses a Universal Value Function Approximator (UVFA) to learn several exploration policies with different exploration-exploitation trade-offs at the same time. Agent57 [77] aims to manage the tradeoff between exploration and exploitation using a "meta-controller" that adaptively selects a correct policy (ranging from very exploratory to purely exploitative) for the training phase. Agent57 outperforms the standard human benchmark on all 57 Atari games.

2) Exploration in Evolution Strategies

ES algorithms optimize the fitness score while exploring around 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 [58, 80]. To overcome this limitation, many ESs algorithms with enhanced exploration techniques have been proposed.

One way to extract approximate gradients from a non-smooth objective function, $F(\boldsymbol{\theta})$, is by adding noise to its parameter vector, $\boldsymbol{\theta}$. This yields a *new differentiable* function, $F_{ES}(\boldsymbol{\theta})$. OpenAI-ES [4] exploits this idea by sampling noise from a Gaussian distribution and adding it to the parameter vector $\boldsymbol{\theta}$. The algorithm then optimizes using stochastic gradient ascent. Additionally, OpenAI-ES relays on a few auxiliary techniques to enhance its performance: virtual batch normalization [31] for enhanced exploration, antithetic sampling [81] for reduced variance, and fitness shaping [44] for improving local optima avoidance.

Choromanski et al. [82] proposed two strategies to enhance the exploration of Derivative Free Optimization (DFO) methods such as OpenAI-ES [4]: (i) structured exploration, where the authors showed that random orthogonal and Quasi Monte Carlo finite difference directions are much more effective than random Gaussian directions for parameter exploration; and (ii) compact policies, whereby imposing a parameter sharing structure on the policy architecture, they were able to significantly reduce the dimensionality of the problem without losing accuracy and thus speeding up the learning process.

Maheswaranathan et al. [83] 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: (i) the bias of the surrogate gradients needs to be known; and (ii) when the bias is too small, Guided ES cannot find a better descent direction than the surrogate gradient. Meier et al. [84] draw inspiration from how momentum is used for optimizing DNNs to improve upon Guided ES [83]. 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. [85] noted that fixing the dimensionality of subspaces (as in Guided ES [83]) leads to suboptimal performance. Therefore, they proposed ASEBO: an algorithm that adaptively controls the dimensionality of subspaces based on gradient estimators from previous iterations. ASEBO was compared to several ESs and DRL algorithms and showed promising averaged performance.

TABLE III: Deep Reinforcement Learning exploration algorithms

Algorithm	Description	Experiments	Ref.
Bootstrapped DQN	uses DNNs and ensemble Q-functions to explore an environment	outperforms DQN by orders of magnitude in terms of cumulative rewards	[68]
UCB+InfoGain	integrates UCB with Q-function ensemble	outperforms bootstrapped DQN	[70]
State pseudo-count	uses density models and pseudo-count to approximate state visitation count which is used to compute the reward bonus	superior to DQN, especially in hard-to-explore environments	[65]
VIME	measures information gain as KL divergence between current and updated distribution after an observation	improves the performance of TRPO [71], REINFORCE [11] when added to them	[72]
ICM	uses a forward dynamic model to predict states and measures information gain as the difference between the predicted and observed state	outperforms TRPO-VIME in VizDoom (a sparse 3D environment)	[73]
Episodic curiosity	uses episodic memory to form the novelty bonus	outperforms ICM in visually rich 3D environments from VizDoom and DMLab	[74]
Go-Explore	The previously visited states are stored in memory. In phase one Go-explore explores until a solution is found. In phase two Go-explore Robustifies the found solution	Performance improvements on hard exploration problems over other methods such as DQN+PixelCNN, DQN+CTS, BASS-hash	[75]
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	[76]
Agent57	uses a meta-controller for adaptively selecting the right policy: ranging from purely exploratory to purely exploitative	first DRL agent that surpasses the standard human benchmark on all 57 Atari games	[77]

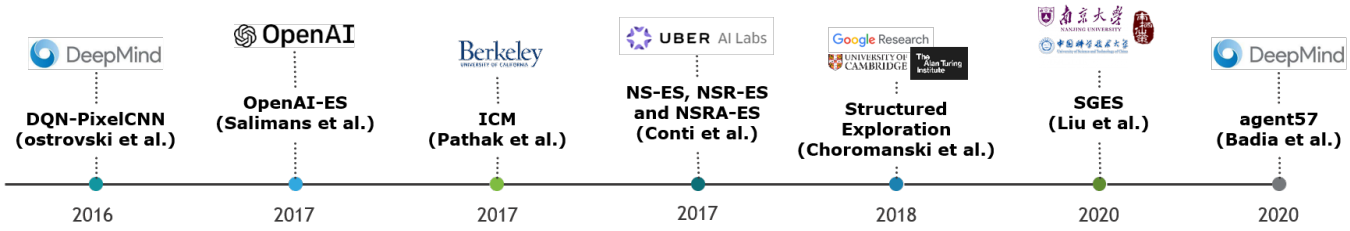


Fig. 6: Deep Reinforcement Learning and Evolution Strategies exploration algorithms shown on a timeline

Liu et al. [86] proposed Self-Guided Evolution Strategies (SGES). This work is inspired by both ASEBO [85] and Guided ES [83]. Further, it is based on two main ideas: leveraging historical estimated gradients and building a guiding subspace from which search directions are sampled probabilistically. The results show that SGES outperforms Open-AI [4], Guided ES [83], CMA-ES and vanilla 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. [87] proposed Directional Gaussian Smoothing Evolution Strategy (DGS-ES). It encourages non-local exploration and improves high-dimensional exploration. In contrast to regular Gaussian smoothing, directional Gaussian smoothing conducts $1D$ non-local explorations along d orthogonal directions. The Gauss-Hermite quadrature is then used for improving the convergence speed of the algorithm. Its superior performance is showcased by comparing it to many algorithms including OpenAI-ES [4] and ASEBO [85].

To encourage exploration in environments with sparse or deceptive reward signals, Conti et al. [32] proposed hybridizing ESs with directed exploration methods (i.e., Novelty Search (NS) [88] and Quality Diversity (QD) [89]). 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.

3) Comparison

Our observations of this section are summarized below.

- The exploration-exploitation dilemma is still an active field of research and environments with sparse and deceptive reward signals require more sophisticated and capable exploration algorithms.
- Benchmarking exploration strategies happens almost exclusively in simulated/gaming environments. Consequently, the efficacy of these algorithms in real-world applications is mostly unknown.
- Thanks to the recombination and mutation, ESs algorithms might suffer less from local optima than DRL ones.
- ESs still face some problems related to sample efficiency when exploring, as high dimensional optimization tasks can lead to high variance gradients estimates.
- Table III and Table IV summarize some important charac-

TABLE IV: Evolution Strategies exploration algorithms

Algorithm	Description	Experiments	Ref.
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	[4]
Structured Exploration	complements OpenAI-ES [4] 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	[82]
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	[83]
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	[85]
DGS-ES	uses directional Gaussian smoothing to explore along non-local orthogonal directions. It leverages Guss-Hermite quadrature for fast convergence.	improves on state-of-the-art algorithms (e.g., OpenAI-ES and ASEBO) on some problems	[87]
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	[84]
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 ESs algorithms such as OpenAI-ES, Guided ES, ASEBO, CMA-ES on blackbox functions and RL tasks	[86]
NS-ES, NSR-ES, and NSRA-ES	Hybridize Novelty search (NS) and quality diversity (QD) algorithms with ESs to improve the performance of ESs on sparse RL problems.	avoid local optima encountered by ESs while achieving higher performance on Atari and simulated robot tasks	[32]

teristics of DRL and ESs exploration algorithms.

C. Non-Markov 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. The degree to which agents can observe (changes in) the environment has an impact on their decision behavior. In certain favorable scenarios the state of the agent in its environment might be fully observable (e.g., using sensors) to an extent such that the Markov assumption holds. In other cases, the state of the environment is only partially observable and/or the agent faces a distribution of environments (Meta-RL).

1) Partially Observable

In many real-world applications, agents can only partially observe the state of their environments and might only have access to their local observations. This means agents need to take into accent the history of observations—actions and rewards—to produce a better estimation of the underlying hidden state [90, 91, 92]. These problems are usually modeled as a partially observable Markov decision process (POMDP). Researchers have addressed the POMDP problem setup through the proposal of many RL models and evolutionary strategies. In DRL, one possibility is to employ a neural network with a recurrent architecture that enables agents to consider past observations [93, 94].

2) Meta Reinforcement Learning

Meta-RL is concerned with learning a policy that can be quickly generalized across a distribution of tasks or environments (modeled as MDPs). Generally, a meta-learner achieves that through two stages optimization process: first, a meta-policy is trained on a distribution of similar tasks with the hope of learning the common dynamics across these tasks; then, the second stage fine-tunes the meta-policy while acting on a particular task sampled from a similar but unseen task distribution [95]. Examples of meta-RL tasks include: navigating towards distinct goals [96], going through different mazes [97], dealing with component failures [98], or driving different cars [99].

Meta-RL can be subdivided into two categories [96]: RNN-based [100, 101] and gradient-based learners [102, 103].

Recurrent Models (RNN-based learners). Leveraging the agent-environment interaction history provides more information, which leads to improved learning [99, 104]. This idea can be implemented using Recurrent Neural Networks (RNNs) (or other recurrent models) [97, 100, 101, 105]. 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.

General architecture of a meta-RL algorithm is illustrated in Figure 7 [106], where an agent is modeled as two loops, both implementing RL algorithms. The outer loop samples a new environment in every iteration and tunes the parameters of the inner loop. Consequently, the inner loop can adjust more rapidly to new tasks by interacting with the associated environments and optimizing for maximal rewards.

Duan et al. [101] and Wang et al. [100] proposed analogous recurrent Meta-RL agents: R^2 and DRL-meta, respectively. They implemented a long-short term memory (LSTM) and a gate recurrent unit (GRU) architecture in which the hidden states serve as a memory for tracking characteristics of interaction trajectories. The main difference between both approaches relates to the set of environments. Environments in [100] are issued from a parameterized distribution [107]. In contrast, those in [101] are relatively unrelated [107].

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, short-term memory is challenging for RNN due to the vanishing gradient problem. Furthermore, RNN-based meta-learners cannot pinpoint specific prior experiences [97, 108].

To overcome these limitations, Mishra et al. [97] proposed Simple Neural Attentive Learner (SNAIL). It combines

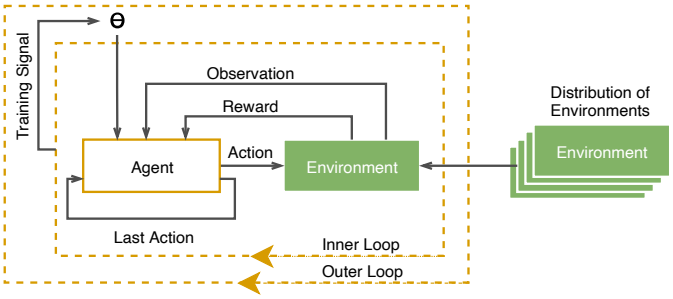


Fig. 7: Schematic of Meta-reinforcement Learning; illustrating the inner and outer loops of training [106]

temporal convolutions and attention mechanisms. The former aggregates information from past experiences and the latter pinpoints specific pieces of information. SNAIL's architecture consists of three main parts: (i) DenseBlock, a causal 1D-convolution with specific dilation rate; (ii) TCBlock, a series of DenseBlocks with exponentially increasing dilation rates; and (iii) AttentionBlock, where key-vlaue lookups take place. This general-purpose model has shown its efficacy on tasks ranging from supervised to reinforcement learning. Despite that, challenges such as the long time needed for getting the right architectures of TCBlocks and DenseBlocks. [108] persist.

Gradient-Based Models. Model Agnostic Meta-Learning (MAML) [102] 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. [109] 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 (SGD) on each task to compute a new set of parameters. Then, it moves the model weights towards the new parameters. Next, we look at how meta-learning tries to make ESs more efficient.

3) Meta Evolution Strategies

Gajewski et al. [110] introduced "Evolvability ES", an ES-based meta-learning algorithm for RL tasks. It combines concepts from evolvability search [111], ESs [4], and MAML [102] 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) [111]. Consequently, Evolvability ES facilitates adaptation and generalization while leveraging the scalability of ESs [110, 112]. Evolvability ES shows a competitive performance to gradient-based meta-learning algorithms. Quality Evolvability ES [112] noted that the original Evolvability ES [113] can only be used to solve problems where the task performance and evolability align. To eliminate this restriction, Quality Evolvability ES optimizes for both -task performance and evolability- simultaneously.

Song et al. [114] argue that policy gradient-based Model Agnostic Meta Learning (MAML) algorithms [102] face significant difficulties when estimating second derivative using backpropagation on stochastic policies. Therefore, they introduced ES-MAML, a meta-learner that leverages ES [4] 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. [115] combined Hill-Climbing adaptation with ES-MAML to develop noise-tolerant meta-RL learner. The authors showcased the performance of their algorithm using a physical legged robot.

Wang et al. [116] incorporated an instance weighting mechanism with ESs to generate an adaptable and salable meta-learner, Instance Weighted Incremental Evolution Strategies (IW-IES).

Wang et al. [116] introduced Instance Weighted Incremental Evolution Strategies (IW-IES). It incorporates an instance weighting mechanism with ESs to generate an adaptable and salable meta-learner. IW-IES assigns weights to offsprings 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 to ES-MAML, IW-IES proved competitive for robot navigation tasks.

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 [117]. Examples of using Meta-RL to train physical robots include: Nagabandi et al. [98] built on top of MAML a model-based meta-RL agent to train a legged millirobot; Arndt et al. [118] proposed a similar framework to MAML to train a robot on a task of hitting a hockey puck; and Song et al. [115] introduced a variant of ES-MAML to train and quickly adapt the policy commanding a legged robot.

4) Comparison

Key observations of this section can be summarized as:

- In many cases, RL and ESs are faced with problems that are more complex than the traditional MDP setting, such as in the partially observable case and the meta-learning setting.
- Meta-learning enables an agent to explore more intelligently and acquire useful knowledge more quickly.
- There are two main approaches for Meta-RL: gradient-based and recurrent models. Gradient-based Meta-RL is generally a two-stage optimization process: first, it optimizes on a task distribution level, and then, fine-tunes for a specific task. Meta-RL with recurrent models make use

TABLE V: Gradient-based Meta Reinforcement Learning.

Algorithms	Description	Experiments	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 architecture	outperforms other benchmarks on the bandits' problems; properly adapts to invariances in MDP tasks	[100]
RL^2	trains an RNN on a distribution of RL tasks. The RNN serves as a dynamic task embedding storage. R^2 uses the GRU architecture	comparable to theoretically optimal algorithms in small-scale settings. It has the potential to scale to high-dimensional tasks	[101]
SNAIL	combines temporal convolution and attention mechanisms	outperforms LSTM and MAML	[97]
MAML	given a task distribution, it searches for optimal initial parameters such that a few gradient steps are sufficient to solve tasks drawn from that distribution.	outperforms classical methods such as random and pretrained methods	[102]
Reptile	similar to first-order MAML	on-par with the performance of MAML	[109]

TABLE VI: Evolution Strategies for Meta Reinforcement learning

	Description	Experiments	ref.
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	[110]
ES-MAML	uses ESs to overcome the limitations of MAML	competitive with policy gradient methods; yields better adaptation with fewer queries	[114]
IW-IES	uses NES for updating the RL policy network parameters in a dynamic environment	outperforms ES-MAML on set of robot navigation tasks	[116]

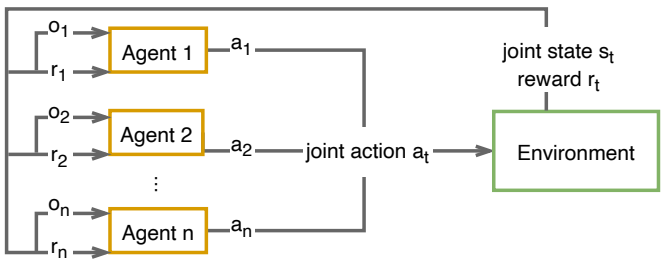


Fig. 8: Multi-agent Reinforcement Learning Overview [119]

of specific recurrent architectures to learn how to act in a distribution of environments.

- There are many challenges in Meta-RL methods, such as estimating first and second-order derivatives, high variance, and high computation needs.
- ES-based meta-RL attempts to address the limitations of gradient-based Meta-RL; however, ES-based meta-RL itself faces a different set of challenges such as the sample efficiency.
- Meta-RL is particularly suited for tackling the sim-to-real problem. For instance, a generic policy is trained in simulation and fine-tuned via the interaction with real world.

D. Learning in multiagent settings

A multi-agent system (MAS) is a distributed system of multiple cooperating or competing (physical or virtual) agents, working towards maximizing their own objectives within a shared environment [120]. Currently, MAS form one of the leading research areas of Artificial Intelligence due to their wide applicability. Virtually any application that can be partitioned and parallelized can benefit from using multiple agents.

1) multi-agent Reinforcement Learning

An MAS can be combined with DRL to form a Multi-agent Deep Reinforcement Learning (MADRL) system which addresses sequential decision-making problems for multiple agents sharing a common environment (Figure 8). MADRL agents are trained to learn certain behaviors through interaction with the environment and optionally with other agents. Since the environment and the reward states are affected by the joint actions of all agents, the single-agent MDP model cannot be directly applied to MADRL systems, as they do not adhere to the Markov property. The Markov (or Stochastic) games (MG) [121] framework comes as a generalization of the MDP that captures the entanglement of the multiple agents. There are several important properties to be considered when considering MADRL systems. In the following section we will discuss each of these properties and their resulting impact on the overall system.

Setup: Cooperative vs Competitive. In a cooperative game, also known as a team problem, the agents seek to maximize a common reward signal by taking actions that favor their outcome, while taking into account their effects on other agents. Most contemporary applications are based upon a cooperative setup. Examples of this scenario include foraging, exploration and warehouse robots.

One of the main challenges of learning in a cooperative setting is termed as *multi-agent credit assignment problem* which refers to how to divide a reward obtained on a team level amongst individual learners [122]. Due to the complex interaction dynamics of the agents, it is not trivial to determine whose actions were beneficial to the group reward.

On the other hands, agents in a competitive game receive different reward signals based on the overall outcome of the joint actions. In this setup, certain actions might be beneficial to one set of agents while being indifferent or disadvantageous for the other agents.

Control: Centralized vs Decentralized. Another important distinction to make for MADRL systems is the centralized versus decentralized control approach. In the case of centralized control, there exists a single control entity that governs the decisions of all agents based on all available joint actions, joint rewards and joint observations. While this approach enables optimal decisions, it quickly becomes computationally intractable as the number of agents in a system grows. Additionally, this creates the risk of a single point of failure since the whole system could fail if the central controller breaks.

The decentralized approach does not make use of a central controller and relies on agents to make decisions independently, based on the information available to them locally. Decentralized systems can be subdivided into two categories: "A decentralized setting with networked agents", and "A fully decentralized setting" [123]. The former setup involves agents which can communicate with other agents and use the shared information to optimize their actions. In the latter scenario, agents make independent decisions without information exchange. While this means that no explicit messages can be sent, it is still possible to influence the behavior of other agents by affecting their reward as seen in [124]. While the decentralized approach can provide more scalability and robustness, it also significantly increases the complexity of the system as there is no central entity that has knowledge of and can control the state of each robot. An interesting future research direction might be semi-centralized MADRL systems in which one or more central entities possess partial information of a set of agents. Alternatively, it is possible to alternate techniques between different phases of the design. Chen [125] proposed a system with centralized training and exploration and decentralized execution which can increase inter-agent collaboration and sample efficiency.

Challenges in Multi-agent Reinforcement Learning. Moving from a single-agent to a multi-agent environment 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 non-stationarity of the environment [126]. Next, MADRL challenges are discussed.

Non-stationarity: In a MADRL system, agents are learning concurrently and their actions reshape their shared surroundings repeatedly, resulting in a non-stationay environment. Consequently, the convergence of well-known algorithms such as Q-learning can no longer be guaranteed as the Markov property assumption of the environment is violated [28, 127, 128]. Many papers in the literature that attempt to address the non-stationarity problem. Castaneda [129] 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 non-stationarity of the environment in MADRL settings data stored in an experience replay buffer can become outdated. To counter this unwanted behavior, Lenient-DQN conceived by Palmer et al. [130] utilizes decaying temperature values for adjusting the policy updates sampled from the experience replay memory.

Scalability: One way to deal with the non-stationarity 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 as the number of agents increases, the state-action spaces grow exponentially, a phenomenon known as "combinatorial complexity" [131, 132, 133]. To balance the challenges imposed by non-stationarity and scalability, a centralized training and decentralized execution approach has been proposed [134, 135, 136, 137, 138].

Modeling Multi-agent Reinforcement Problems. This section summarizes the common approaches of modeling and solving MADRL problems.

Independent-learning: Under this approach each agent considers other agents as part of the environment; consequently each agent is trained independently [128, 139**?**]. This approach does not suffer from the scalability problem [140**?**], but it makes the environment non-stationary from each agent's perspective [141]. Furthermore, it conflicts with the usage of experience replay that improves the DQN algorithm [28]. To stabilize the experience replay buffer in MADRL settings, Foerster et al. [140] used importance sampling and replay buffer samples aging.

Fully observable critic: A way to deal with the non-stationarity of a MADRL environment is by leveraging an actor-critic approach. Lowe et al. [136] proposed a multi-agent 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. A number of extensions to MADDPG has been proposed [142, 143, 144, 145].

Value function decomposition: Learning the optimal action-value function in fully cooperative MADRL settings is challenging. 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 sub-optimal 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 NQ_i for N agents. QMIX [135] algorithm improves on VDN by relaxing some of the additivity constrains and enforcing positive weights on the mixer network.

Learning to communicate: Cooperative environments may

allow agents to communicate. In such settings, the agents can learn a communication protocol to achieve their shared objective more optimally [146, 147]. Foerster et al. [148] proposed two algorithms, Reinforced Inter-Agent Learning (RIAL) and Differentiable Inter-Agent Learning (DIAL), that use deep networks 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 [149] 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 non-communicating agents.

Partial observability: Foerster et al. [148] introduced a deep distributed recurrent Q-network (DDRQN) algorithm based on a long short-term memory network to deal with POMDP problems in the multi-agent setting. Gupta et al. [150] extended three types of single-agent RL algorithms based on policy gradient, temporal-difference error, and actor-critic methods to the multi-agent systems domain. Their work shows the importance of using DRL with curriculum learning to address the problem of learning cooperative policies in partially observable complex environments.

We refer the interested reader to the following survey papers for a more in-depth discussion on the topic of multi-agent reinforcement learning: Hernandez-Leal et al. [127] provide a comprehensive survey on the non-stationarity problem in MADRL; OroojlooyJadid and Hajinezhad [141] scope their survey to include the papers that study decentralized MADRL models with a cooperative goal; Da Silva and Costa [151] focus on transfer learning for MADRL systems; a survey on MADRL from the perspective of challenges and applications is introduced by Du and Ding [152]; a selective overview of theories and algorithms is presented in [**?**]; and a survey and critique of MADRL is given in [132].

2) Multi-agent Evolution Strategies

ES algorithms do not require the problem to be formulated as an MDP; therefore, they do not suffer from the non-stationarity of the environment. Consequently, it is relatively easy to extend a single-agent ES algorithm to the multi-agent domain and develop an application. Hiraga et al. [153] developed robotics controllers based on ESs for managing congestion in robotic swarms path formation using LEDs. The performed experiment covered a swarm of robots, each having seven distance sensors, a ground sensor, an omnidirectional camera, and RGB LEDs. An artificial neural network (three-layered neural network) represents the controller of the robot, having as inputs: the distance sensors, ground sensors, and the cameras, and as outputs: the motors and LEDs controls. (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 [154] for building a swarm capable of cooperatively transporting food to a nest and collectively distinguishing between foods and poisons. Hiraga et al. [154] developed a controller for a robotic swarm using CMA-ES, aiming to automatically generate the behavior of the robots.

Tang et al. [155] proposed an adversarial training multi-agent 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 (agility refers to coordinated control of legs, balance control, etc.). Training is done using ESs and more specifically by augmenting CMA-ES to the multi-agent framework. There are two steps for training: An outer loop which 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 [156] proposed a predator-prey system that leverages ESs (OpenAI-ES, CMA-ES). It consists of having multiple predators trained to catch prey in a certain time frame. The predator controllers are homogeneous and are represented by neural networks which parameters are optimized with ESs (OpenAI-ES, CMA-ES) and Bayesian Optimization. The NN has three inputs (the inverse of the distance from the predator to the other nearest predator, the angle between the orientation of the predator and the direction of the prey relative to the predator, the distance between the predator itself and the prey), 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 towards the least dangerous locations. After performing various experiments, the predators showcased a successful collective behavior: moving following a formation and avoiding collisions.

Multi-agent credit assignment problem. In a multi-agent setting, agents often receive a shared reward for all the agents, making it harder to learn proper cooperative behaviors. Li et al. [157] thus proposed to use Parallelized ESs along with a Value Decomposition Network (useful for identifying each agent's contribution to the training process) for solving cooperative multi-agent tasks. Figure 9 is an overview of the overall PES-VD algorithm, which consists of two phases. First, the policies of each agent are represented by a NN with parameters $\boldsymbol{\theta}$, optimized using Parallelized ES. Each agent thus identifies its actions independently following its policy and by interacting with its environment. In a second place, seeing how the reward is common to the whole team, a Value Decomposition Network 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 Value Decomposition Network and a master node collects the data and updates the policies and the Value Decomposition Network accordingly.

Various researchers proposed multi-agent solutions for swarm scenarios leveraging ESs. Each robot in the swarm runs the same network, thus maintaining collective behavior.

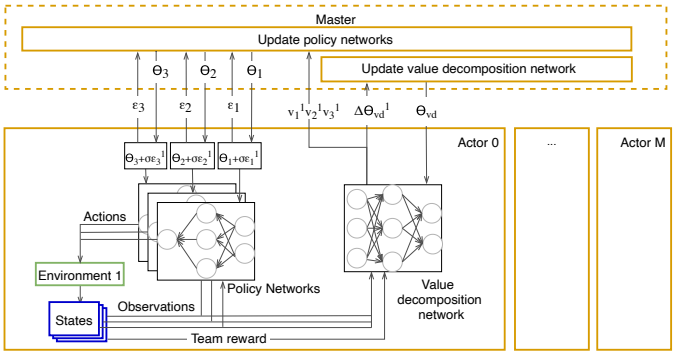


Fig. 9: PES-VD overview

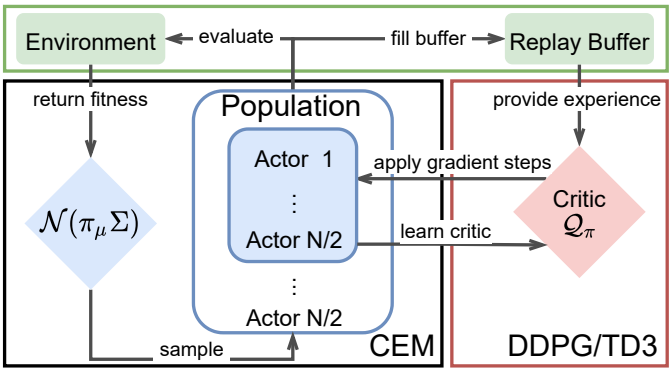


Fig. 10: CEM-RL [162]: a hybrid algorithm that combines cross-entropy method with (Twin) Deep Deterministic Gradient Policy [163].

Rais Martínez and Aznar Gregori [158] assess the performance of ESs (CMA-ES, PEPG, SES, GA, and OpenAI-ES) for multi-agent learning in the swarm aggregation task. In this problem, the robots controllers are represented by a NN with 2 hidden layers. Each has 8 infrared sensors and 4 microphones for inputs and 2 wheels and a speaker as output. Similarly, Fan et al. [159] used ESs on different multi-agent UAV swarm combat scenarios. Aznar et al. [160] developed a swarm foraging behavior using DRL and CMA-ES.

3) Comparison

Here we summarize our observations of this section

- Training under a multi-agent setting is more challenging than training a single agent for a plethora of reasons. There are usually two types of agents in MADRL: cooperative and competitive agents. Algorithms can make use of a centralized or decentralized framework and will act in a partially or fully observable environment.
- New algorithms such as PES-VD [157] propose a direct solution to some of the main challenges of MADRL. PES-VD uses a Value Decomposition Network for solving multi-agent credit assignment problems.
- Using ESs for multi-agent learning is still a growing field with a large potential as to the many advantages ESs can bring to concepts such as “collective robotic learning" and “cloud robotics" [161] with its improved approach to parallelism [4].
- Semi-centralized MADRL systems are an interesting future research direction in which a few central entities possess partial information of a set of agents.
- The literature on ESs for multi-agent scenarios seems to focus on enabling applications. We hypothesis that this is because it is less challenging to extend single-agent ES algorithms to the multi-agent domain. This is because ES algorithms do not require Markov property in the formulation of the problem, and therefore, they do not suffer from non-stationary environments.

V. Hybrid Deep Reinforcement Learning and Evolution Strategies Algorithms

Although DRL and ES have the same objective—optimizing an objective function in a potentially unknown environment—they have different strengths and weaknesses [164, 165]. For example, DRL can be sample efficient thanks to the combination of RL and deep learning; while ES have robust convergence properties and exploration strategies. The hybrid approach combines DRL and ES to get the best of both worlds. Although the idea is not new [166], hybridizing DRL and ES has gained momentum, driven by the recent success of DRL and ES [4, 167]. Combining these strengths has, among others, let to very strong play in the most challenging Real-Time Strategy (RTS) games such as StarCraft [168]. We describe in the following a few population-guided parallel learning schemes that enhance the performance of RL algorithms.

Pourchot and Sigaud [162] 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 policy gradient (TD3) [163] or the Deep Deterministic Policy Gradient DDPG [19] algorithms (Figure 10). The CEM-RL architecture consists of a population of actors that are generated using CEM, and of 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 [163], and Evolution Reinforcement Learning (ERL) [169]: a hybrid algorithm that combines a DDPG agent with an evolutionary algorithm. Shopov and Markova [170] combined ESs and multi-agent DRL (Deep Q-Networks) for Sequential Games and showcased the model’s efficiency as compared to Classical multi-agent reinforcement training with ϵ-greedy. The experiment performed by Shopov and Markova [170] aims to optimize the behaviour of a group of autonomous agents (the pursuers) in a map. Tests were performed on two cases: one map with almost no obstacles and another with many obstacles (increased probability of falling into the local minimum). Using ESs on the latter yielded better performance.

Houthooft et al. [171] devised a hybrid RL agent, Evolved Policy Gradients (EPG), that, in addition to the policy, optimizes 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. Thanks to this ability to fine tune the loss function according to the environment and agent history, EPG can learn faster than a standard RL agent.

Diqi Chen and Gao [172] proposed a hybrid agent to approximate the Pareto frontier uniformly in a multi-objective 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 achieve a well-distributed Pareto frontier, but they face difficulties optimizing a DNN. Therefore, Diqi Chen and Gao [172] proposed a two-stage multi-objective reinforcement learning (MORL) framework. In the first stage, a multi-policy soft actor-critic algorithm learns multiple policies collaboratively. And, in the second stage, a multi-objective covariance matrix adaptation evolution strategy (MO-CMA-ES) fine-tunes policy-independent parameters to approach a uniform Pareto frontier.

De Bruin et al. [173] used a hybrid approach to train and fine-tune a DNN control policy. Their approach consists of two main steps: (i) learning a state representation and initial policy from high-dimensional input data using gradient-based methods (i.e., DQN or DDPG); and (ii) fine-tuning the final action selection parameters of the DNN using CMA-ES. This architecture enables the policy to surpass in performance its gradient-based counterpart while using fewer trials compared to a pure gradient-free policy.

Several other researchers have also proposed solutions hybridizing ES and DRL for various applications. For example, Song et al. [174] proposed ES-ENAS, a neural architecture search (NAS) algorithm for identifying RL policies using ES and Efficient NAS (ENAS); Ferreira et al. [175] used ES to learn agent-agnostic synthetic environments (SEs) for Reinforcement Learning.

A. Comparison

Here we summarize our observations of this section:

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

VI. Applications

Next, we compare DRL and ESs based on the applications they support. The goal is to get an indication of their potential by tracking their application record so far. The results of querying Google Scholar is presented in Table VIII in conjunction with the keywords that were used[1].

A. Deep Reinforcement Learning applications

Gaming. Video games such as the Atari games [179] are excellent testbeds for DRL algorithms, given their well-defined problem settings and virtual environment. This makes evaluation safe and fast compared to real-world experiments [180].

There have been two important triumphs for DRL with respect to perfect information games. First, in 2015, Mnih et al. [181] developed an algorithm that could learn to play different Atari 2600 games at a superhuman level using only the image pixels as input. This work paved the way for DRL applications trained on high-dimensional data based only on the reward signal. Soon after, in 2016, Silver et al. [182] developed AlphaGo, the first program ever to beat a world champion in Go. Instead of the handcrafted rules often seen in chess programs, AlphaGo consisted of neural networks trained using a combination of Supervised Learning (SL) and RL. Only a year later, this achievement was triumphed by Silver et al. [183], whose AlphaGo Zero program beat its predecessor AlphaGo. AlphaGo Zero was based solely on RL, omitting the need for human data. More recent works have also been successful in imperfect information games which, unlike Go and Atari games, only let agents observe part of the system. In OpenAI Five [184], agents were able to defeat the world's esports champions in the game of Dota2, while AlphaStar [185] attained one of the highest rankings in the complex real-time strategy game of StarCraft II. [46, 50, 65, 76] further examined DRL algorithms' ability to scale, parallelize, and explore using Atari games. Lastly, an extensive survey on DRL in video games has been composed by Shao et al. [180].

Robotics is another domain which forms a prominent testbed for DRL algorithms [186, 187]. DRL can provide robots with navigation, obstacle avoidance and decision making capabilities, by mapping sensory data directly to actual motor commands [188, 189]. In some cases this has enabled robots to learn complex movements such as jumping or walking [190, 191]. Tai et al. [192] proposed a mapless motion planner which relies on training in simulation, after which, physical agents were able to navigate unknown static environments without fine-tuning. While most works involved simulation, Gu et al. [161] showed that DRL can be used to learn complex robotics 3D manipulation skills from scratch on real-world robots and further reduced training time by parallelizing the training across multiple robots. Haarnoja et al. [191] demonstrated that using DRL, one can also achieve stable quadrupedal locomotion on a physical robot within a reasonable time without prior training. For an in-depth review of the use of DRL for robot manipulation, we refer the interested reader to [186, 187].

Finance. DRL also finds applications in trading [193, 194] and investment management [195], including cryptocurrency

[1]The search query template is "allintitle: "evolution strategies" OR "evolutionary strategies" key_word_1 OR key_word_1 -excluded_key_word" and "allintitle: "reinforcement learning" ...

TABLE VII: Hybrid algorithms highlights

Algorithm	Description	Experiments	Ref.
CEM-RL	combines a cross-entropy method and Twin Delayed Deep Deterministic policy gradient [163] to find robust policies	outperforms CEM, TD3, multi-actor TD3, and Evolutionary Reinforcement Learning [169]	[162]
Evolved Policy Gradients (EPG)	uses gradient descent and CMA-ES for policy and loss function optimization, respectively	achieves faster learning than policy gradient methods and provides qualitatively different behavior from other popular meta-learning algorithms	[171]
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 the hypervolume-based [176], radial [177], Pareto following [177], and Deep Neuroevolution [178] algorithm on computing the Pareto frontier	[172]
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	[173]

TABLE VIII: Searching Google Scholar for DRL and ESs applications (only papers' titles are considered).

Industry field	Search terms	Results	
		DRL	ESs
Gaming	game, games, gaming, playing, play, mahjong, atari, tetris, soccer *excluding* survey and review	1630	44
Robotics	robotics, "motion control", robots, "robot navigation", assembly, robot, grasping *excluding* survey and review	2350	39
Finance	finance, financial, trading, portfolio, stock, price, liquidation, hedging, banking, trader, cryptocurrency, underpricing *excluding* survey and review	475	34
Communications	network, routing, communications, wireless, 5g, LTE, MAC, "access control", "network slicing", *excluding* "neural network" survey and review	2020	53
Energy	energy, power *excluding* survey and review	1470	41
Transportation	transportation, transport, vehicle, traffic, fleet, driving *excluding* survey and review	1580	25

[196]. Moody and Saffell [197] built a DRL agent for stock trading using raw financial data as the DNN input. Carapuço et al. [198] described a system for short-term speculation in the foreign exchange market, based on DRL. Wu et al. [199] proposed adaptive stock trading strategies leveraging DRL. A more recent DRL work by Lei et al. [200], adaptively selects between historical data and the changing trend of a stock, depending on the current state.

Communications. Upcoming networks such as the 5G network, emphasize the need for efficient dynamic and large-scale solutions [201]. DRL has been emerging as an effective tool to tackle various problems and challenges within the field of networking [202]. For example, Wang et al. [203] applied a DQN to automatically optimize data transmission and reception in a multi-wireless-channel access problem. Ye and Li [204] developed a similar system for vehicle-to-vehicle communication. The optimal transmission bitrate can change over time. DRL can dynamically optimize the bitrate based on the quality of the last segment, the current buffer state [205, 206]and other channel statistics [207, 208]. Proactive caching can greatly reduce the number of transmissions over the network. However, deciding which content to cache is not trivial. Researchers have used DQNs to determine which information to keep in a cache based on observations of the channel state [209], cache state [210], request history [211, 212] and available base stations [213, 214, 215].

Energy. Within the energy sector, *smart grids* make intelligent decisions with respect to electricity generation, transmission, distribution, consumption and control. DRL has been used in a variety of settings to tackle electric power system decision and control problems [216], such as in the context of microgrids [217] or buildings energy optimization [218, 219].

Transportation. Congestion, safety and efficiency are important aspects of transportation. DRL is often used for adaptive traffic signal control to reduce waiting times [220, 221, 222]. Chen et al. [223] expanded upon this and conceived the first DRL control system which scales to thousands of traffic lights. Wang and Sun [224] developed a MADRL framework to prevent 'bus bunching' and streamline the flow of public transport. Manchella et al. [225] proposed a model-free DRL algorithm which packs ride-sharing passengers together with goods delivery to optimize fleet utilization and fuel efficiency.

B. Evolution Strategy applications

ESs applications are categorized and highlighted next.

Gaming. similarly to DRL, gaming represents one of the main testbeds for ESs. Most of the literature on ESs reviewed in this survey test their algorithms on Atari games [4, 5, 29, 32, 226, 227, 228, 229]. These are considered to be challenging as they present the agents with high dimensional visual inputs and a diverse and interesting set of tasks that were designed to be difficult for humans players [14].

Robotics. The ability of ESs to continuously control actuators has been leverage in controlling simulated and real robotic systems [4, 58, 80, 86, 230, 231, 232]. Hu et al. [233] used CMA-ES to make a robot learns how to grasp object under uncertainty. Uchitane et al. [234] augmented the $(\mu + \lambda)$-ES algorithm with a mask operation during the mutation step to tune the controller's parameters of humanoid robots. With help of ESs Li et al. [235] design an indoor mobile robot navigation using monocular vision.

Finance. Korczak and Lipinski [236] presented a portfolio optimization algorithm using ESs. Rimcharoen et al. [237], Sutheebanjard and Premchaiswadi [238] proposed the Adaptive and (1+1)-ES methods for predicting the Stock Exchange of Thailand index movement. Bonde and Khaled [239] predicted the changes (increase or decrease) of stock prices for different companies using ESs and Genetic Algorithms. Pai and Michel [240] proposed ESs with hall of fame (ES-HOF) for optimizing long–short portfolios with the 130-30-strategy-based constraint. Pai and Michel [241] used multi-objective

ESs for futures portfolio optimization. Yu [242] proposed an ESs method for the multi-asset multi-period portfolio optimization. Sable et al. [243] proposed an ESs approach for predicting the short time prices of stocks. Sorensen et al. [244] applied meta-learning algorithms to ES for stock trading.

Communications. Different methods are proposed throughout the literature that used ESs for communication. Pérez-Pérez et al. [245] used ESs with NSGAII (ESN) to approximate the Pareto frontier of the mobile adhoc network (MANETs). Krulikovska et al. [246] used ESs for the routing of multipoint connections. Additionally, they proposed methods for improving ESs. Nissen and Gold [247] used ESs for designing a survivable network while taking economics and reliability into consideration. He et al. [248] analyzed the data characteristics of wireless sensor network (WSN), and proposed a method for fault diagnosis of WSN based on a belief rule base (BRB) model which is optimized using CMA-ES.Srivastava and Singh [249] used ESs for solving the the total rotation minimization problem (TRMP) in directional sensor networks. Srivastava et al. [250] presented an ESs method for solving the Cover scheduling problem in wireless sensor networks (WSN-CSP). Gu and Potkonjak [251] proposed an ESs method to search for a network configuration able to produce and stabilize responses of a Physical Unclonable Functions (PUFs).

Energy. ESs have been used to optimize many energy-related systems. For example, Mendoza et al. [252] used ESs to select optimal size of feeders in radial power distribution system. Lezama et al. [253] used differential ESs for large-scale energy resource management in smart grids. Coelho et al. [254] used it for energy load forecasting in electric grids. Versloot et al. [255] optimized near-field wireless power transfer using ESs.

Transportation. A number of papers that use ESs for managing and optimizing vehicle traffic have been proposed. Balaji et al. [256] proposed a multi-agent-based real-time centralized evolutionary optimization technique for urban traffic management in the area of traffic signal control. Mester and Bräysy [257] combined ESs with guided local search to tackle large-scale vehicle routing problems with time windows. Mester and Bräysy [257] simulated and optimized traffic flow with help of ESs.

C. Comparison

Next we list our observations about this section.

- Both DRL and ES algorithms have found adoption in many domains such as robotics, games, and finance.
- DRL-based solutions seem to excel in situations that require scalable and adaptive behavior.
- DRL receives much more attention than ESs within the scientific (Table VIII). We suspect that two of the main reasons for this gap are (i) DRL has a richer structure; therefore, it naturally allows for more research, and (ii) there are similar algorithmic families to ESs (e.g., genetic algorithms) which may result in reduced focus on ESs.
- Many great-performing DRL and ES algorithms are benchmarked in simulated environments. Consequently, their performance in real-world applications are still questionable.

VII. 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. We briefly list some of them in the sequel.

Sample Efficiency. DRL agents require a large number of samples (i.e., interactions with environments) to learn good-performing policies. Collecting so many samples is not always feasible due to either computational reasons or because the quantity of interactions with the environment is limited. Although this problem has been tackled in different ways (e.g., transfer learning, meta-learning), more innovation and research are still needed [258, 259]. One promising research direction to tackle this problem is model-based RL. However, getting an accurate model of the environment is usually hard.

ESs can provide more robust policies as compared to DRL; however, they are even less sample efficient, as they work with full-length episodes [260, 261], and they do not use any type of memory [260]. Approaches to improve sample efficiency in ESs such sample resue and importance mixing [260, 262] have been proposed; however, more research and innovation are still required.

Exploration versus exploitation. The exploration versus exploitation dilemma is one of the most prominent problems in RL. Beyond classical balancing approaches such as ε-greedy [1], Upper Confidence Bound (UCB) [59], and Thompson Sampling [60], recent breakthroughs enable the exploration of novel environments. For example, Osband et al. [68] observed the importance of temporal correlation and proposed the bootstrapped DQN; and Bellemare et al. [65] used density models to scale UCB to problems with high-dimensional input data. In spite of that, exploring complex environments is still a very active field of research.

ESs realize exploration through the recombination and mutation steps. Despite their effectiveness in exploration, ESs may still get trapped in local optima [58, 80]. Proposals have been made to enhance ESs exploration capabilities [82, 83]; however, more work in this direction is needed. In general, DRL and ESs are proposed to tackle ever more novel environments, and consequently, the exploration versus exploitation dilemma still poses a challenge that requires innovation.

Sparse reward. A reward signal guides the learning process of an RL agent. When this signal is sparse learning becomes much harder. Although, different solutions have been introduced (e.g., reward shaping [1], curiosity-driven methods [73], curriculum learning [263], hierarchical learning [264] and inverse RL [265]), learning with sparse rewards still represents an open challenge.

Direct the exploration of ES algorithms to counter the sparsity and/or deceptiveness of an RL task is one of the most important challenges to scale ESs to more complex environments and make them more efficient. A detailed summary of the challenges related to ESs, such as differential evolution and swarm optimization, is presented in [21].

Simulation-to-reality gap. Despite the benefits of simulations, they give rise to the sim-to-real gap: policies that are learned in simulations often do not work as expected in the real world. Different techniques are being adapted to mitigate the effect of this gap. For example, [266, 267] randomized the simulated environment to produce more generalized models. Rao et al. [268] noted that such randomization requires manually specifying which aspects of the simulator to randomize. Therefore, they used domain adaptation (i.e., many simulated examples and a few real ones) to train a robot on grasping tasks without manually instrumenting the simulator. Despite such efforts, the sim-to-real gap is still an open challenge to be addressed.

VIII. Conclusion

Deep Reinforcement Learning (DRL) and Evolution Strategies (ESs) have the same objective but make use of different mechanisms for learning sequential decision-making tasks. In this paper, we provided the necessary background of DRL and ESs in order to understand their relative strengths and weaknesses, which may lead to developing an algorithmic family that is superior to each one of them. Instead of focusing on individual algorithms, we considered major learning aspects such as parallelism, exploration, meta-learning, and multi-agent learning. We believe that hybridizing DRL and ESs has a high potential to drive the development of agents that operate reliably and efficiently in the real world.

Acknowledgment

This work has been undertaken in the Internet of Swarms project sponsored by Cognizant Technology Solutions and Rijksdienst voor Ondernemend Nederland under PPS O&I.

References

[1] R. S. Sutton and A. G. Barto, Reinforcement learning: An introduction. MIT press, 2018.

[2] V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, and J. Pineau, "An introduction to deep reinforcement learning," arXiv preprint arXiv:1811.12560, 2018.

[3] A. E. Eiben and J. E. Smith, Introduction to evolutionary computing. Springer, 2003.

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

[5] P. Chrabaszcz, I. Loshchilov, and F. Hutter, "Back to basics: Benchmarking canonical evolution strategies for playing atari," 2018.

[6] H. Qian and Y. Yu, "Derivative-free reinforcement learning: A review," Frontiers of Computer Science (electronic), 2021.

[7] B. R. Kiran, I. Sobh, V. Talpaert, P. Mannion, A. A. A. Sallab, S. Yogamani, and P. Pérez, "Deep reinforcement learning for autonomous driving: A survey," IEEE Transactions on Intelligent Transportation Systems, 2021.

[8] T. T. Nguyen, N. D. Nguyen, and S. Nahavandi, "Deep reinforcement learning for multiagent systems: A review of challenges, solutions, and applications," IEEE transactions on cybernetics, 2020.

[9] D. P. Bertsekas, "Dynamic programming and optimal control 3rd edition, volume ii," Belmont, MA: Athena Scientific, 2011.

[10] R. S. Sutton, D. A. McAllester, S. P. Singh, Y. Mansour et al., "Policy gradient methods for reinforcement learning with function approximation." in NIPs. Citeseer, 1999.

[11] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," Machine learning, 1992.

[12] R. BELLMAN, "A markovian decision process," Journal of Mathematics and Mechanics, 1957.

[13] G. Rummery and M. Niranjan, "On-line q-learning using connectionist systems," Technical Report CUED/F-INFENG/TR 166, 1994.

[14] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," 2013.

[15] V. R. Konda and J. N. Tsitsiklis, "Actor-critic algorithms," in Advances in neural information processing systems, 2000.

[16] L. Kaiser, M. Babaeizadeh, P. Milos, B. Osinski, R. H. Campbell, K. Czechowski, D. Erhan, C. Finn, P. Kozakowski, S. Levine et al., "Model-based reinforcement learning for atari," arXiv preprint arXiv:1903.00374, 2019.

[17] D. Silver, A. Huang, C. Maddison, A. Guez, L. Sifre, G. Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," Nature, 2016.

[18] A. S. Polydoros and L. Nalpantidis, "Survey of model-based reinforcement learning: Applications on robotics," Journal of Intelligent & Robotic Systems, 2017.

[19] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," 2019.

[20] N. Hansen, D. V. Arnold, and A. Auger, Evolution Strategies. Springer, 2015.

[21] Z. Li, X. Lin, Q. Zhang, and H. Liu, "Evolution strategies for continuous optimization: A survey of the state-of-the-art," Swarm and Evolutionary Computation, 2020.

[22] V. Heidrich-Meisner and C. Igel, "Similarities and differences between policy gradient methods and evolution strategies," in ESANN 2008, 2008.

[23] V. Heidrich-Meisner and C. Igel, "Evolution strategies for direct policy search," in International Conference on Parallel Problem Solving from Nature, 2008.

[24] C. Heidrich-Meisner, Verena and Igel, "Neuroevolution strategies for episodic reinforcement learning," Journal of Algorithms, 2009.

[25] V. Heidrich-Meisner and C. Igel, "Hoeffding and bernstein races for selecting policies in evolutionary direct policy search," in Proceedings of the 26th Annual International Conference on Machine Learning, 2009.

[26] P. Chrabaszcz, I. Loshchilov, and F. Hutter, "Back to basics: Benchmarking canonical evolution strategies for playing atari," CoRR, 2018. [Online]. Available: http://arxiv.org/abs/1802.08842

[27] C. J. Watkins and P. Dayan, "Q-learning," Machine learning, 1992.

[28] V. Mnih, K. Kavukcuoglu, D. Silver, A. Rusu, J. Veness, M. Bellemare, A. Graves, M. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," Nature, 2015.

[29] L. Fuks, N. Awad, F. Hutter, and M. Lindauer, "An evolution strategy with progressive episode lengths for playing games," in Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19. International Joint Conferences on Artificial Intelligence Organization, 2019.

[30] K. Varelas, A. Auger, D. Brockhoff, N. Hansen, O. A. ElHara, Y. Semet, R. Kassab, and F. Barbaresco, "A comparative study of large-scale variants of cma-es," in International Conference on Parallel Problem Solving from Nature, 2018.

[31] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training gans," 2016.

[32] E. Conti, V. Madhavan, F. P. Such, J. Lehman, K. O. Stanley, and J. Clune, "Improving exploration in evolution strategies for deep reinforcement learning via a population of novelty-seeking agents," 2018.

[33] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," Advances in neural information processing systems, 2012.

[34] J. N. Tsitsiklis and B. Van Roy, "An analysis of temporal-difference learning with function approximation," IEEE transactions on automatic control, 1997.

[35] L.-J. Lin, "Reinforcement learning for robots using neural networks," Carnegie-Mellon Univ Pittsburgh PA School of Computer Science, Tech. Rep., 1993.

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

[37] A. Slowik and H. Kwasnicka, "Evolutionary algorithms and their applications to engineering problems," Neural Computing and Applications, 2020.

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

[39] H.-P. Schwefel, Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie. Birkh "a user, 1977.

[40] N. Hansen and A. Ostermeier, "Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation," in Proceedings of IEEE international conference on evolutionary computation, 1996.

[41] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," Evolutionary computation, 2001.

[42] N. Hansen, S. D. Müller, and P. Koumoutsakos, "Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es)," Evolutionary computation, 2003.

[43] N. Hansen, "The cma evolution strategy: A tutorial," 2016.

[44] D. Wierstra, T. Schaul, T. Glasmachers, Y. Sun, J. Peters, and J. Schmidhuber, "Natural evolution strategies," The Journal of Machine Learning Research, 2014.

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

[46] A. Nair, P. Srinivasan, S. Blackwell, C. Alcicek, R. Fearon, A. D. Maria, V. Panneershelvam, M. Suleyman, C. Beattie, S. Petersen, S. Legg, V. Mnih, K. Kavukcuoglu, and D. Silver, "Massively parallel methods for deep reinforcement learning," 2015.

[47] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," 2016.

[48] A. V. Clemente, H. N. Castejón, and A. Chandra, "Efficient parallel methods for deep reinforcement learning," 2017.

[49] D. Horgan, J. Quan, D. Budden, G. Barth-Maron, M. Hessel, H. van Hasselt, and D. Silver, "Distributed prioritized experience replay," 2018.

[50] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, S. Legg, and K. Kavukcuoglu, "Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures," 2018.

[51] L. Espeholt, R. Marinier, P. Stanczyk, K. Wang, and M. Michalski, "Seed rl: Scalable and efficient deep-rl with accelerated central inference," 2020.

[52] M. Grounds and D. Kudenko, "Parallel reinforcement learning with linear function approximation," in Adaptive Agents and Multi-Agent Systems III. Adaptation and Multi-Agent Learning. Springer, 2005.

[53] J. Dean, G. S. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, and A. Y. Ng, "Large scale distributed deep networks," in NIPS, 2012.

[54] M. Babaeizadeh, I. Frosio, S. Tyree, J. Clemons, and J. Kautz, "Reinforcement learning through asynchronous advantage actor-critic on a gpu," 2017.

[55] N. Heess, D. TB, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, S. Eslami et al., "Emergence of locomotion behaviours in rich environments," arXiv preprint arXiv:1707.02286, 2017.

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

[57] S. Kapturowski, G. Ostrovski, J. Quan, R. Munos, and W. Dabney, "Recurrent experience replay in distributed reinforcement learning," in International conference on learning representations, 2018.

[58] G. Liu, L. Zhao, F. Yang, J. Bian, T. Qin, N. Yu, and T. Liu, "Trust region evolution strategies," in AAAI, 2019.

[59] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," Machine learning, 2002.

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

[61] J. Achiam and S. Sastry, "Surprise-based intrinsic motivation for deep reinforcement learning," 2017.

[62] D. Pathak, D. Gandhi, and A. Gupta, "Self-supervised exploration via disagreement," 2019.

[63] P. Shyam, W. Jaśkowski, and F. Gomez, "Model-based active exploration," 2019.

[64] H. Kim, J. Kim, Y. Jeong, S. Levine, and H. O. Song, "Emi: Exploration with mutual information," 2019.

[65] M. G. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos, "Unifying count-based exploration and intrinsic motivation," 2016.

[66] G. Ostrovski, M. G. Bellemare, A. van den Oord, and R. Munos, "Count-based exploration with neural density models," 2017.

[67] H. Tang, R. Houthooft, D. Foote, A. Stooke, X. Chen, Y. Duan, J. Schulman, F. De Turck, and P. Abbeel, "# exploration: A study of count-based exploration for deep reinforcement learning," in 31st Conference on Neural Information Processing Systems (NIPS), 2017.

[68] I. Osband, C. Blundell, A. Pritzel, and B. Van Roy, "Deep exploration via bootstrapped dqn," arXiv preprint arXiv:1602.04621, 2016.

[69] R. Y. Chen, J. Schulman, P. Abbeel, and S. Sidor, "Ucb and infogain exploration via q-ensembles," arXiv preprint arXiv:1706.01502, 2017.

[70] B. O'Donoghue, I. Osband, R. Munos, and V. Mnih, "The uncertainty bellman equation and exploration," in International Conference on Machine Learning, 2018.

[71] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in International conference on machine learning. PMLR, 2015.

[72] R. Houthooft, X. Chen, Y. Duan, J. Schulman, F. D. Turck, and P. Abbeel, "Vime: Variational information maximizing exploration," 2017.

[73] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, "Curiosity-driven exploration by self-supervised prediction," 2017.

[74] N. Savinov, A. Raichuk, R. Marinier, D. Vincent, M. Pollefeys, T. Lillicrap, and S. Gelly, "Episodic curiosity through reachability," arXiv preprint arXiv:1810.02274, 2018.

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

[76] A. P. Badia, P. Sprechmann, A. Vitvitskyi, D. Guo, B. Piot, S. Kapturowski, O. Tieleman, M. Arjovsky, A. Pritzel, A. Bolt et al., "Never give up: Learning directed exploration strategies," arXiv preprint arXiv:2002.06038, 2020.

[77] A. P. Badia, B. Piot, S. Kapturowski, P. Sprechmann, A. Vitvitskyi, D. Guo, and C. Blundell, "Agent57: Outperforming the atari human benchmark," 2020.

[78] J. Schmidhuber, "Formal theory of creativity, fun, and intrinsic motivation (1990–2010)," IEEE Transactions on Autonomous Mental Development, 2010.

[79] Y. Burda, H. Edwards, A. Storkey, and O. Klimov, "Exploration by random network distillation," 2018.

[80] J. Zhang, H. Tran, and G. Zhang, "Accelerating reinforcement learning with a directional-gaussian-smoothing evolution strategy," 2020.

[81] J. Geweke, "Antithetic acceleration of monte carlo integration in bayesian inference," Journal of Econometrics, 1988.

[82] K. Choromanski, M. Rowland, V. Sindhwani, R. Turner, and A. Weller, "Structured evolution with compact architectures for scalable policy optimization," 2018.

[83] N. Maheswaranathan, L. Metz, G. Tucker, D. Choi, and J. Sohl-Dickstein, "Guided evolutionary strategies: Augmenting random search with surrogate gradients," 2019.

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

[85] K. Choromanski, A. Pacchiano, J. Parker-Holder, and Y. Tang, "From complexity to simplicity: Adaptive es-active subspaces for blackbox optimization," 2019.

[86] F.-Y. Liu, Z.-N. Li, and C. Qian, "Self-guided evolution strategies with historical estimated gradients," in Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20, 2020.

[87] J. Zhang, H. Tran, D. Lu, and G. Zhang, "A novel evolution strategy with directional gaussian smoothing for blackbox optimization," 2020.

[88] J. Lehman and K. Stanley, Novelty Search and the Problem with Objectives. Springer, 11 2011.

[89] J. Pugh, L. Soros, and K. Stanley, "Quality diversity: A new frontier for evolutionary computation," Frontiers in Robotics and AI, 2016.

[90] R. McCallum, "Reinforcement learning with selective perception and hidden state," 1997.

[91] R. Ortner, O.-A. Maillard, and D. Ryabko, "Selecting near-optimal approximate state representations in reinforcement learning," in International Conference on Algorithmic Learning Theory, 2014.

[92] V. François-Lavet, G. Rabusseau, J. Pineau, D. Ernst, and R. Fonteneau, "On overfitting and asymptotic bias in batch reinforcement learning with partial observability," Journal of Artificial Intelligence Research, 2019.

[93] M. Hausknecht and P. Stone, "Deep recurrent q-learning for partially observable mdps," arXiv preprint arXiv:1507.06527, 2015.

[94] C. Igel, "Neuroevolution for reinforcement learning using evolution strategies," in The 2003 Congress on Evolutionary Computation, 2003. CEC'03., 2003.

[95] T. Schaul and J. Schmidhuber, "Metalearning," Scholarpedia, 2010.

[96] A. Gupta, R. Mendonca, Y. Liu, P. Abbeel, and S. Levine, "Meta-reinforcement learning of structured exploration strategies," 2018.

[97] N. Mishra, M. Rohaninejad, X. Chen, and P. Abbeel, "A simple neural attentive meta-learner," 2017.

[98] A. Nagabandi, I. Clavera, S. Liu, R. S. Fearing, P. Abbeel, S. Levine, and C. Finn, "Learning to adapt in dynamic, real-world environments through meta-reinforcement learning," 2019.

[99] F. M. Garcia and P. S. Thomas, "A meta-mdp approach to exploration for lifelong reinforcement learning," 2019.

[100] J. X. Wang, Z. Kurth-Nelson, D. Tirumala, H. Soyer, J. Z. Leibo, R. Munos, C. Blundell, D. Kumaran, and M. Botvinick, "Learning to reinforcement learn," 2017.

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

[102] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," 2017.

[103] Z. Li, F. Zhou, F. Chen, and H. Li, "Meta-sgd: Learning to learn quickly for few-shot learning," 2017.

[104] S. Hochreiter, A. S. Younger, and P. R. Conwell, "Learning to learn using gradient descent," in International Conference on Artificial Neural Networks. Springer, 2001.

[105] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. Lillicrap, "One-shot learning with memory-augmented neural networks," 2016.

[106] M. Botvinick, S. Ritter, J. Wang, Z. Kurth-Nelson, C. Blundell, and D. Hassabis, "Reinforcement learning, fast and slow," Trends in Cognitive Sciences, 2019.

[107] J. G. Robles and J. Vanschoren, "Learning to reinforcement learn for neural architecture search," 2019.

[108] M. Huisman, J. N. van Rijn, and A. Plaat, "A survey of deep meta-learning," 2020.

[109] A. Nichol, J. Achiam, and J. Schulman, "On first-order meta-learning algorithms," 2018.

[110] A. Gajewski, J. Clune, K. O. Stanley, and J. Lehman, "Evolvability es: Scalable and direct optimization of evolvability," 2019.

[111] H. Mengistu, J. Lehman, and J. Clune, "Evolvability search: directly selecting for evolvability in order to study and produce it," in Proceedings of the Genetic and Evolutionary Computation Conference 2016, 2016.

[112] A. Katona, D. W. Franks, and J. A. Walker, "Quality evolvability es: Evolving individuals with a distribution of well performing and diverse offspring," 2021.

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

[114] X. Song, W. Gao, Y. Yang, K. Choromanski, A. Pacchiano, and Y. Tang, "Es-maml: Simple hessian-free meta learning," 2020.

[115] X. Song, Y. Yang, K. Choromanski, K. Caluwaerts, W. Gao, C. Finn, and J. Tan, "Rapidly adaptable legged robots via evolutionary meta-learning," 2020.

[116] Z. Wang, C. Chen, and D. Dong, "Instance weighted incremental evolution strategies for reinforcement learning in dynamic environments," 2020.

[117] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke, "Sim-to-real: Learning agile locomotion for quadruped robots," 2018.

[118] K. Arndt, M. Hazara, A. Ghadirzadeh, and V. Kyrki, "Meta reinforcement learning for sim-to-real domain adaptation," 2019.

[119] A. Nowe, P. Vrancx, and Y.-M. De Hauwere, Game Theory and Multi-agent Reinforcement Learning, 2012.

[120] 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 Transactions on Vehicular Technology, 2020.

[121] L. S. Shapley, "Stochastic games," Proceedings of the national academy of sciences, 1953.

[122] L. Panait and S. Luke, "Cooperative multi-agent learning: The state of the art," Autonomous agents and multi-agent systems, vol. 11, no. 3, pp. 387–434, 2005.

[123] K. Zhang, Z. Yang, and T. Başar, "Multi-agent reinforcement learning: A selective overview of theories and algorithms," 2019.

[124] G. Sartoretti, J. Kerr, Y. Shi, G. Wagner, T. K. S. Kumar, S. Koenig, and H. Choset, "Primal: Pathfinding via reinforcement and imitation multi-agent learning," IEEE Robotics and Automation Letters, 2019.

[125] G. Chen, "A new framework for multi-agent reinforcement learning – centralized training and exploration with decentralized execution via policy distillation," 2019.

[126] D. Lee, N. He, P. Kamalaruban, and V. Cevher, "Optimization for reinforcement learning: From a single agent to cooperative agents," IEEE Signal Processing Magazine, 2020.

[127] P. Hernandez-Leal, M. Kaisers, T. Baarslag, and E. M.

de Cote, "A survey of learning in multiagent environments: Dealing with non-stationarity," arXiv preprint arXiv:1707.09183, 2017.
[128] M. Tan, "Multi-agent reinforcement learning: Independent vs. cooperative agents," in Proceedings of the tenth international conference on machine learning, 1993.
[129] A. O. Castaneda, "Deep reinforcement learning variants of multi-agent learning algorithms," Edinburgh: School of Informatics, University of Edinburgh, 2016.
[130] G. Palmer, K. Tuyls, D. Bloembergen, and R. Savani, "Lenient multi-agent deep reinforcement learning," arXiv preprint arXiv:1707.04402, 2017.
[131] B. Kartal, J. Godoy, I. Karamouzas, and S. J. Guy, "Stochastic tree search with useful cycles for patrolling problems," in 2015 IEEE International Conference on Robotics and Automation (ICRA), 2015.
[132] P. Hernandez-Leal, B. Kartal, and M. E. Taylor, "A survey and critique of multiagent deep reinforcement learning," Autonomous Agents and Multi-Agent Systems, 2019.
[133] Y. Yang and J. Wang, "An overview of multi-agent reinforcement learning from game theoretical perspective," 2020.
[134] P. Sunehag, G. Lever, A. Gruslys, W. M. Czarnecki, V. Zambaldi, M. Jaderberg, M. Lanctot, N. Sonnerat, J. Z. Leibo, K. Tuyls, and T. Graepel, "Value-decomposition networks for cooperative multi-agent learning," 2017.
[135] 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 International Conference on Machine Learning, 2018.
[136] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," arXiv preprint arXiv:1706.02275, 2017.
[137] J. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson, "Counterfactual multi-agent policy gradients," 2018.
[138] G. Chen, "A new framework for multi-agent reinforcement learning–centralized training and exploration with decentralized execution via policy distillation," 2019.
[139] M. Lauer and M. Riedmiller, "An algorithm for distributed reinforcement learning in cooperative multi-agent systems," in In Proceedings of the Seventeenth International Conference on Machine Learning, 2000.
[140] J. Foerster, N. Nardelli, G. Farquhar, T. Afouras, P. H. Torr, P. Kohli, and S. Whiteson, "Stabilising experience replay for deep multi-agent reinforcement learning," in International conference on machine learning, 2017.
[141] A. OroojlooyJadid and D. Hajinezhad, "A review of cooperative multi-agent deep reinforcement learning," arXiv preprint arXiv:1908.03963, 2019.
[142] X. Chu and H. Ye, "Parameter sharing deep deterministic policy gradient for cooperative multi-agent reinforcement learning," arXiv preprint arXiv:1710.00336, 2017.
[143] H. Ryu, H. Shin, and J. Park, "Multi-agent actor-critic with generative cooperative policy network," arXiv preprint arXiv:1810.09206, 2018.
[144] H. Mao, Z. Zhang, Z. Xiao, and Z. Gong, "Modelling the dynamic joint policy of teammates with attention multi-agent ddpg," arXiv preprint arXiv:1811.07029, 2018.
[145] R. E. Wang, M. Everett, and J. P. How, "R-maddpg for partially observable environments and limited communication," arXiv preprint arXiv:2002.06684, 2020.
[146] T. Kasai, H. Tenmoto, and A. Kamiya, "Learning of communication codes in multi-agent reinforcement learning problem," in 2008 IEEE Conference on Soft Computing in Industrial Applications, 2008.
[147] C. L. Giles and K.-C. Jim, "Learning communication for multi-agent systems," in Workshop on Radical Agent Concepts, 2002.
[148] J. N. Foerster, Y. M. Assael, N. de Freitas, and S. Whiteson, "Learning to communicate to solve riddles with deep distributed recurrent q-networks," arXiv preprint arXiv:1602.02672, 2016.
[149] S. Sukhbaatar, A. Szlam, and R. Fergus, "Learning multiagent communication with backpropagation," arXiv preprint arXiv:1605.07736, 2016.
[150] J. K. Gupta, M. Egorov, and M. Kochenderfer, "Cooperative multi-agent control using deep reinforcement learning," in International Conference on Autonomous Agents and Multiagent Systems, 2017.
[151] F. L. Da Silva and A. H. R. Costa, "A survey on transfer learning for multiagent reinforcement learning systems," Journal of Artificial Intelligence Research, 2019.
[152] W. Du and S. Ding, "A survey on multi-agent deep reinforcement learning: from the perspective of challenges and applications," Artificial Intelligence Review, 2020.
[153] M. Hiraga, Y. Wei, T. Yasuda, and K. Ohkura, "Evolving autonomous specialization in congested path formation task of robotic swarms," Artificial Life and Robotics, 2018.
[154] M. Hiraga, Y. Wei, and K. Ohkura, "Evolving collective cognition of robotic swarms in the foraging task with poison," in 2019 IEEE Congress on Evolutionary Computation (CEC), 2019.
[155] Y. Tang, J. Tan, and T. Harada, "Learning agile locomotion via adversarial training," arXiv preprint arXiv:2008.00603, 2020.
[156] J. Chen and Z. Gao, "A framework for learning predator-prey agents from simulation to real world," 2020.
[157] G. Li, Q. Duan, and Y. Shi, A Parallel Evolutionary Algorithm with Value Decomposition for Multi-agent Problems, 2020.
[158] J. Rais Martínez and F. Aznar Gregori, "Comparison of evolutionary strategies for reinforcement learning

in a swarm aggregation behaviour," in 2020 The 3rd International Conference on Machine Learning and Machine Intelligence, 2020.
[159] D. D. Fan, E. Theodorou, and J. Reeder, "Model-based stochastic search for large scale optimization of multi-agent uav swarms," 2018.
[160] F. Aznar, M. Pujol, and R. Rizo, "Learning a swarm foraging behavior with microscopic fuzzy controllers using deep reinforcement learning," Applied Sciences, 2021.
[161] S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," 2016.
[162] A. Pourchot and O. Sigaud, "Cem-rl: Combining evolutionary and gradient-based methods for policy search," 2019.
[163] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in International Conference on Machine Learning, 2018.
[164] K. Hansel, J. Moos, and C. Derstroff, Benchmarking the Natural Gradient in Policy Gradient Methods and Evolution Strategies. Springer International Publishing, 2021.
[165] P. Ecoffet, N. Fontbonne, J.-B. André, and N. Bredeche, "Policy search with rare significant events: Choosing the right partner to cooperate with," 2021.
[166] A. Stafylopatis and K. Blekas, "Autonomous vehicle navigation using evolutionary reinforcement learning," European Journal of Operational Research, 1998.
[167] M. Jaderberg, W. M. Czarnecki, I. Dunning, L. Marris, G. Lever, A. G. Castaneda, C. Beattie, N. C. Rabinowitz, A. S. Morcos, A. Ruderman et al., "Human-level performance in 3d multiplayer games with population-based reinforcement learning," Science, 2019.
[168] O. Vinyals, I. Babuschkin, J. Chung, M. Mathieu, M. Jaderberg, W. Czarnecki, A. Dudzik, A. Huang, P. Georgiev, R. Powell, T. Ewalds, D. Horgan, M. Kroiss, I. Danihelka, J. Agapiou, J. Oh, V. Dalibard, D. Choi, L. Sifre, Y. Sulsky, S. Vezhnevets, J. Molloy, T. Cai, D. Budden, T. Paine, C. Gulcehre, Z. Wang, T. Pfaff, T. Pohlen, D. Yogatama, J. Cohen, K. McKinney, O. Smith, T. Schaul, T. Lillicrap, C. Apps, K. Kavukcuoglu, D. Hassabis, and D. Silver, "AlphaStar: Mastering the Real-Time Strategy Game StarCraft II," https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/, 2019.
[169] S. Khadka and K. Tumer, "Evolutionary reinforcement learning," arXiv preprint arXiv:1805.07917, 2018.
[170] V. Shopov and V. Markova, "A study of the impact of evolutionary strategies on performance of reinforcement learning autonomous agents," ICAS 2018, 2018.
[171] R. Houthooft, R. Y. Chen, P. Isola, B. C. Stadie, F. Wolski, J. Ho, and P. Abbeel, "Evolved policy gradients," 2018.
[172] Y. W. Diqi Chen and W. Gao, "Combining a gradient-based method and an evolution strategy for multi-objective reinforcement learning," 2020.
[173] T. De Bruin, J. Kober, K. Tuyls, and R. Babuška, "Fine-tuning deep rl with gradient-free optimization*," in Proceedings of the IFAC World Congress, 2020.
[174] X. Song, K. Choromanski, J. Parker-Holder, Y. Tang, D. Peng, D. Jain, W. Gao, A. Pacchiano, T. Sarlos, and Y. Yang, "Es-enas: Combining evolution strategies with neural architecture search at no extra cost for reinforcement learning," 2021.
[175] F. Ferreira, T. Nierhoff, and F. Hutter, "Learning synthetic environments for reinforcement learning with evolution strategies," 2021.
[176] K. Van Moffaert, M. M. Drugan, and A. Nowé, "Hypervolume-based multi-objective reinforcement learning," in International Conference on Evolutionary Multi-Criterion Optimization. Springer, 2013.
[177] S. Parisi, M. Pirotta, N. Smacchia, L. Bascetta, and M. Restelli, "Policy gradient approaches for multi-objective sequential decision making," in 2014 International Joint Conference on Neural Networks (IJCNN), 2014.
[178] F. P. 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," 2018.
[179] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," Journal of Artificial Intelligence Research, 2013.
[180] K. Shao, Z. Tang, Y. Zhu, N. Li, and D. Zhao, "A survey of deep reinforcement learning in video games," arXiv preprint arXiv:1912.10944, 2019.
[181] V. Mnih, K. Kavukcuoglu, D. Silver, A. Rusu, J. Veness, M. Bellemare, A. Graves, M. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," Nature, 2015.
[182] D. Silver, A. Huang, C. Maddison, A. Guez, L. Sifre, G. Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," Nature, 2016.
[183] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton et al., "Mastering the game of go without human knowledge," nature, 2017.
[184] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Dębiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse et al., "Dota 2 with large scale deep reinforcement learning," arXiv preprint arXiv:1912.06680, 2019.
[185] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Math-

ieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev et al., "Grandmaster level in starcraft ii using multi-agent reinforcement learning," Nature, 2019.
[186] J. Kober, J. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," The International Journal of Robotics Research, 2013.
[187] H. Nguyen and H. La, "Review of deep reinforcement learning for robot manipulation," in 2019 Third IEEE International Conference on Robotic Computing (IRC). IEEE, 2019.
[188] Y. F. Chen, M. Liu, M. Everett, and J. P. How, "Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning," in 2017 IEEE international conference on robotics and automation (ICRA), 2017.
[189] G. Kahn, A. Villaflor, B. Ding, P. Abbeel, and S. Levine, "Self-supervised deep reinforcement learning with generalized computation graphs for robot navigation," in 2018 IEEE International Conference on Robotics and Automation (ICRA), 2018.
[190] G. Bellegarda and Q. Nguyen, "Robust quadruped jumping via deep reinforcement learning," arXiv preprint arXiv:2011.07089, 2020.
[191] T. Haarnoja, A. Zhou, S. Ha, J. Tan, G. Tucker, and S. Levine, "Learning to walk via deep reinforcement learning," CoRR, 2018.
[192] L. Tai, G. Paolo, and M. Liu, "Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation," in 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2017.
[193] Y. Li, W. Zheng, and Z. Zheng, "Deep robust reinforcement learning for practical algorithmic trading," IEEE Access, 2019.
[194] Y. Deng, F. Bao, Y. Kong, Z. Ren, and Q. Dai, "Deep direct reinforcement learning for financial signal representation and trading," IEEE Transactions on Neural Networks and Learning Systems, 2017.
[195] Z. Jiang, D. Xu, and J. Liang, "A deep reinforcement learning framework for the financial portfolio management problem," 2017.
[196] Z. Jiang and J. Liang, "Cryptocurrency portfolio management with deep reinforcement learning," in 2017 Intelligent Systems Conference (IntelliSys), 2017.
[197] J. Moody and M. Saffell, "Reinforcement learning for trading systems and portfolios," in KDD, 1998.
[198] J. Carapuço, R. Neves, and N. Horta, "Reinforcement learning applied to forex trading," Applied Soft Computing, 2018.
[199] X. Wu, H. Chen, J. Wang, L. Troiano, V. Loia, and H. Fujita, "Adaptive stock trading strategies with deep reinforcement learning methods," Information Sciences, 2020.
[200] K. Lei, B. Zhang, Y. Li, M. Yang, and Y. Shen, "Time-driven feature-aware jointly deep reinforcement learning for financial signal representation and algorithmic trading," Expert Systems with Applications, 2020.
[201] Z. Xiong, Y. Zhang, D. Niyato, R. Deng, P. Wang, and L. Wang, "Deep reinforcement learning for mobile 5g and beyond: Fundamentals, applications, and challenges," IEEE Vehicular Technology Magazine, 2019.
[202] N. C. Luong, D. T. Hoang, S. Gong, D. Niyato, P. Wang, Y. Liang, and D. I. Kim, "Applications of deep reinforcement learning in communications and networking: A survey," IEEE Communications Surveys Tutorials, 2019.
[203] S. Wang, H. Liu, P. H. Gomes, and B. Krishnamachari, "Deep reinforcement learning for dynamic multichannel access in wireless networks," CoRR, 2018.
[204] H. Ye and G. Y. Li, "Deep reinforcement learning for resource allocation in v2v communications," 2018.
[205] M. Gadaleta, F. Chiariotti, M. Rossi, and A. Zanella, "D-dash: A deep q-learning framework for dash video streaming," IEEE Transactions on Cognitive Communications and Networking, 2017.
[206] H. Mao, R. Netravali, and M. Alizadeh, "Neural adaptive video streaming with pensieve," in Proceedings of the Conference of the ACM Special Interest Group on Data Communication, 2017.
[207] S. Chinchali, P. Hu, T. Chu, M. Sharma, M. Bansal, R. Misra, M. Pavone, and S. Katti, "Cellular network traffic scheduling with deep reinforcement learning," in AAAI, 2018.
[208] P. Ferreira, R. Paffenroth, A. Wyglinski, T. Hackett, S. Bilén, R. Reinhart, and D. Mortensen, "Multi-objective reinforcement learning for cognitive satellite communications using deep neural network ensembles," IEEE Journal on Selected Areas in Communications, 2018.
[209] Y. He and S. Hu, "Cache-enabled wireless networks with opportunistic interference alignment," arXiv preprint arXiv:1706.09024, 2017.
[210] Y. He, C. Liang, F. R. Yu, N. Zhao, and H. Yin, "Optimization of cache-enabled opportunistic interference alignment wireless networks: A big data deep reinforcement learning approach," in 2017 IEEE International Conference on Communications (ICC), 2017.
[211] C. Zhong, M. C. Gursoy, and S. Velipasalar, "A deep reinforcement learning-based framework for content caching," in 2018 52nd Annual Conference on Information Sciences and Systems (CISS), 2018.
[212] Y. He, Z. Zhang, F. R. Yu, N. Zhao, H. Yin, V. C. Leung, and Y. Zhang, "Deep-reinforcement-learning-based optimization for cache-enabled opportunistic interference alignment wireless networks," IEEE Transactions on Vehicular Technology, 2017.
[213] Y. He, F. R. Yu, N. Zhao, H. Yin, and A. Boukerche, "Deep reinforcement learning (drl)-based resource management in software-defined and virtualized vehicular ad hoc networks," in Proceedings of the 6th ACM Symposium on Development and Analysis of Intelligent

Vehicular Networks and Applications, 2017.
[214] Y. He, C. Liang, Z. Zhang, F. R. Yu, N. Zhao, H. Yin, and Y. Zhang, "Resource allocation in software-defined and information-centric vehicular networks with mobile edge computing," in 2017 IEEE 86th Vehicular Technology Conference (VTC-Fall), 2017.
[215] Y. He, N. Zhao, and H. Yin, "Integrated networking, caching and computing for connected vehicles: A deep reinforcement learning approach," IEEE Transactions on Vehicular Technology, 2018.
[216] M. Glavic, R. Fonteneau, and D. Ernst, "Reinforcement learning for electric power system decision and control: Past considerations and perspectives," IFAC-PapersOnLine.
[217] V. François-Lavet, D. Taralla, D. Ernst, and R. Fonteneau, "Deep reinforcement learning solutions for energy microgrids management," in European Workshop on Reinforcement Learning (EWRL 2016), 2016.
[218] E. Mocanu, D. C. Mocanu, P. H. Nguyen, A. Liotta, M. E. Webber, M. Gibescu, and J. G. Slootweg, "Online building energy optimization using deep reinforcement learning," IEEE transactions on smart grid, 2018.
[219] F. Ruelens, B. J. Claessens, P. Vrancx, F. Spiessens, and G. Deconinck, "Direct load control of thermostatically controlled loads based on sparse observations using deep reinforcement learning," CSEE Journal of Power and Energy Systems, 2019.
[220] L. Li, Y. Lv, and F. Wang, "Traffic signal timing via deep reinforcement learning," IEEE/CAA Journal of Automatica Sinica, 2016.
[221] X. Liang, X. Du, G. Wang, and Z. Han, "A deep reinforcement learning network for traffic light cycle control," IEEE Transactions on Vehicular Technology, 2019.
[222] T. Chu, J. Wang, L. Codecà, and Z. Li, "Multi-agent deep reinforcement learning for large-scale traffic signal control," IEEE Transactions on Intelligent Transportation Systems, 2020.
[223] C. Chen, H. Wei, N. Xu, G. Zheng, M. Yang, Y. Xiong, K. Xu, and Z. Li, "Toward a thousand lights: Decentralized deep reinforcement learning for large-scale traffic signal control," Proceedings of the AAAI Conference on Artificial Intelligence, 2020.
[224] J. Wang and L. Sun, "Dynamic holding control to avoid bus bunching: A multi-agent deep reinforcement learning framework," Transportation Research Part C: Emerging Technologies, 2020.
[225] K. Manchella, A. K. Umrawal, and V. Aggarwal, "Flexpool: A distributed model-free deep reinforcement learning algorithm for joint passengers and goods transportation," IEEE Transactions on Intelligent Transportation Systems, 2021.
[226] P. Pagliuca, N. Milano, and S. Nolfi, "Efficacy of modern neuro-evolutionary strategies for continuous control optimization," 2020.
[227] B. Zhou and J. Feng, "Sample efficient deep neuroevolution in low dimensional latent space," 2018.
[228] Z. Chen, Y. Zhou, X. He, and S. Jiang, "A restart-based rank-1 evolution strategy for reinforcement learning." in IJCAI, 2019.
[229] S. Risi and K. O. Stanley, "Deep neuroevolution of recurrent and discrete world models," 2019.
[230] S. Veer and A. Majumdar, "Cones: Convex natural evolutionary strategies," 2020.
[231] L. Shi, S. Li, Q. Zheng, L. Cao, L. Yang, and G. Pan, "Maximum entropy reinforcement learning with evolution strategies," in 2020 International Joint Conference on Neural Networks (IJCNN), 2020.
[232] B. Jackson and A. Channon, "Neuroevolution of humanoids that walk further and faster with robust gaits," 2019.
[233] Y. Hu, X. Wu, P. Geng, and Z. Li, "Evolution strategies learning with variable impedance control for grasping under uncertainty," IEEE Transactions on Industrial Electronics, vol. 66, no. 10, pp. 7788–7799, 2018.
[234] T. Uchitane, T. Hatanaka, and K. Uosaki, "Evolution strategies for biped locomotion learning using nonlinear oscillators," in Proceedings of SICE Annual Conference 2010. IEEE, 2010, pp. 1458–1461.
[235] M.-H. Li, B.-R. Hong, Z.-S. Cai, S.-H. Piao, and Q.-C. Huang, "Novel indoor mobile robot navigation using monocular vision," Engineering Applications of Artificial Intelligence, vol. 21, no. 3, pp. 485–497, 2008.
[236] J. Korczak and P. Lipinski, "Evolutionary approach to portfolio optimization," in Proceedings of Workshop on Artificial Intelligence for Financial Time Series Analysis, Porto, 2001.
[237] S. Rimcharoen, D. Sutivong, and P. Chongstitvatana, "Prediction of the stock exchange of thailand using adaptive evolution strategies," in 17th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'05), 2005.
[238] P. Sutheebanjard and W. Premchaiswadi, "Factors analysis on stock exchange of thailand (set) index movement," in 2009 7th International Conference on ICT and Knowledge Engineering, 2009.
[239] G. Bonde and R. Khaled, "Stock price prediction using genetic algorithms and evolution strategies," in Proceedings of the International Conference on Genetic and Evolutionary Methods (GEM), 2012.
[240] G. A. V. Pai and T. Michel, "Integrated metaheuristic optimization of 130–30 investment-strategy-based long–short portfolios," International Journal of Intelligent Systems in Accounting and Finance Management, 2012.
[241] G. V. Pai and T. Michel, "Metaheuristic multi-objective optimization of constrained futures portfolios for effective risk management," Swarm and Evolutionary Computation, vol. 19, pp. 1–14, 2014.
[242] K. Yu, "Dynamic portfolio optimization using evolution strategy," 2017.
[243] S. Sable, A. Porwal, and U. Singh, "Stock price prediction using genetic algorithms and evolution strate-

gies," in 2017 International conference of Electronics, Communication and Aerospace Technology (ICECA), 2017.
[244] E. Sorensen, R. Ozzello, R. Rogan, E. Baker, N. Parks, and W. Hu, "Meta-learning of evolutionary strategy for stock trading," Journal of Data Analysis and Information Processing, 2020.
[245] R. Pérez-Pérez, C. Luque, A. Cervantes, and P. Isasi, "Multiobjective algorithms to optimize broadcasting parameters in mobile ad-hoc networks," in 2007 IEEE Congress on Evolutionary Computation, 2007.
[246] L. Krulikovska, J. Filanová, and J. Pavlovic, "Evolution strategies in the multipoint connections routing," Radioengineering, 2010.
[247] V. Nissen and S. Gold, Survivable network design with an evolution strategy. Springer, 2008.
[248] W. He, P. Qiao, Z. Zhou, G. Hu, Z. Feng, and H. Wei, "A new belief-rule-based method for fault diagnosis of wireless sensor network," IEEE Access, 2018.
[249] G. Srivastava and A. Singh, "Boosting an evolution strategy with a preprocessing step: application to group scheduling problem in directional sensor networks," Applied Intelligence, 2018.
[250] G. Srivastava, P. Venkatesh, and A. Singh, "An evolution strategy based approach for cover scheduling problem in wireless sensor networks," Int. J. Mach. Learn. Cybern., 2020.
[251] H. Gu and M. Potkonjak, "Evolution-strategies-driven optimization on secure and reconfigurable interconnection puf networks," Electronics, 2021.
[252] F. Mendoza, D. Requena, J. L. Bemal-Agustin, and J. A. Domínguez-Navarro, "Optimal conductor size selection in radial power distribution systems using evolutionary strategies," in 2006 IEEE/PES Transmission & Distribution Conference and Exposition: Latin America. IEEE, 2006, pp. 1–5.
[253] F. Lezama, L. E. Sucar, E. M. de Cote, J. Soares, and Z. Vale, "Differential evolution strategies for large-scale energy resource management in smart grids," in Proceedings of the Genetic and Evolutionary Computation Conference Companion, 2017, pp. 1279–1286.
[254] V. N. Coelho, F. G. Guimarães, A. J. Reis, I. M. Coelho, B. N. Coelho, and M. J. Souza, "A heuristic fuzzy algorithm bio-inspired by evolution strategies for energy forecasting problems," in 2014 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE). IEEE, 2014, pp. 338–345.
[255] T. W. Versloot, D. J. Barker, and X. O. One, "Optimization of near-field wireless power transfer using evolutionary strategies," in The 8th European Conference on Antennas and Propagation, Netherlands, 2014.
[256] P. G. Balaji, G. Sachdeva, D. Srinivasan, and C. Tham, "Multi-agent system based urban traffic management," in 2007 IEEE Congress on Evolutionary Computation, 2007.
[257] D. Mester and O. Bräysy, "Active guided evolution strategies for large-scale vehicle routing problems with time windows," Computers & Operations Research, vol. 32, no. 6, pp. 1593–1614, 2005.
[258] A. Nagabandi, K. Konolige, S. Levine, and V. Kumar, "Deep dynamics models for learning dexterous manipulation," in Conference on Robot Learning, 2020.
[259] J. Tebbe, L. Krauch, Y. Gao, and A. Zell, "Sample-efficient reinforcement learning in robotic table tennis," arXiv preprint arXiv:2011.03275, 2020.
[260] A. Pourchot, N. Perrin, and O. Sigaud, "Importance mixing: Improving sample reuse in evolutionary policy search methods," 2018.
[261] O. Sigaud and F. Stulp, "Policy search in continuous action domains: an overview," 2019.
[262] Y. Sun, D. Wierstra, T. Schaul, and J. Schmidhuber, "Efficient natural evolution strategies," 2012.
[263] R. Portelas, C. Colas, L. Weng, K. Hofmann, and P.-Y. Oudeyer, "Automatic curriculum learning for deep rl: A short survey," arXiv preprint arXiv:2003.04664, 2020.
[264] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," IEEE Signal Processing Magazine, 2017.
[265] A. Y. Ng and S. J. Russell, "Algorithms for inverse reinforcement learning," in Proceedings of the Seventeenth International Conference on Machine Learning, 2000.
[266] F. Sadeghi and S. Levine, "Cad2rl: Real single-image flight without a single real image," arXiv preprint arXiv:1611.04201, 2016.
[267] J. Matas, S. James, and A. J. Davison, "Sim-to-real reinforcement learning for deformable object manipulation," in Conference on Robot Learning, 2018.
[268] K. Rao, C. Harris, A. Irpan, S. Levine, J. Ibarz, and M. Khansari, "Rl-cyclegan: Reinforcement learning aware simulation-to-real," in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020.