Off-Policy Evolutionary Reinforcement Learning with Maximum Mutations

Karush Suri
University of Toronto

ABSTRACT

Advances in Reinforcement Learning (RL) have demonstrated data efficiency and optimal control over large state spaces at the cost of scalable performance. Genetic methods, on the other hand, provide scalability but depict hyperparameter sensitivity towards evolutionary operations. A combination of the two methods has recently demonstrated success in scaling RL agents to high-dimensional action spaces. Parallel to recent developments, we present the Evolution-based Soft Actor-Critic (ESAC), a scalable RL algorithm. We abstract exploration from exploitation by combining Evolution Strategies (ES) with Soft Actor-Critic (SAC). Through this lens, we enable dominant skill transfer between offsprings by making use of soft winner selections and genetic crossovers in hindsight. Simultaneously we improve hyperparameter sensitivity in evolutions using the novel Automatic Mutation Tuning (AMT). AMT gradually replaces the entropy framework of SAC allowing the population to succeed at the task while *acting as randomly as possible*, without making use of backpropagation updates. In a study of challenging locomotion tasks consisting of high-dimensional action spaces and sparse rewards, ESAC demonstrates improved performance and sample efficiency in comparison to the Maximum Entropy framework. Additionally, ESAC presents efficacious use of hardware resources and algorithm overhead. Our implementation is available at the `Project Website`.

KEYWORDS

Reinforcement Learning, Evolution Strategies, Mutations.

ACM Reference Format:
Karush Suri. 2022. Off-Policy Evolutionary Reinforcement Learning with Maximum Mutations. In *Proc. of the 21st International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2022), Online, May 9–13, 2022*, IFAAMAS, 11 pages.

1 INTRODUCTION

Concepts and applications of Reinforcement Learning (RL) have seen a tremendous growth over the past decade [24]. These consist of applications in arcade games [24], board games [35] and lately, robot control tasks [19]. A primary reason for this growth is the usage of computationally efficient function approximators such as neural networks [16]. Modern-day RL algorithms make use of parallelization to reduce training times [23] and boost agent's performance through effective exploration giving rise to scalable methods [7, 12, 43]. However, a number of open problems such as approximation bias, lack of scalability in the case of long time horizons and lack of diverse exploration restrict the application of scalability to complex control tasks.

Proc. of the 21st International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2022), P. Faliszewski, V. Mascardi, C. Pelachaud, M.E. Taylor (eds.), May 9–13, 2022, Online. © 2022 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). All rights reserved.

Modern-day RL algorithms such as Soft Actor-Critic (SAC) [6] maximize entropy which is indicative of continued exploration. However, using a computationally expensive framework limits scalability as it increases the number of gradient-based [30] updates of the overall algorithm. Moreover, tasks consisting of long time horizons have higher computational overhead as a result of long trajectory lengths. For instance, obtaining accurate position estimates [12] over longer horizons require additional computation times which varies *linearly* with the hardware requirement. Such a variation calls for increased scalability in the RL domain.

Diverse exploration strategies are essential for the agent to navigate its way in the environment and comprehend intricate aspects of less visited states[21]. Various modern-day RL methods lack significant exploration [23, 24] which is addressed by making use of meta-controller[17] and curiosity-driven [3] strategies at the cost of sample efficiency and scalability.

Recent advances in RL have leveraged evolutionary computing for effective exploration and scalability [9, 13, 22, 32, 37]. These methods often fall short of optimal performance and depict sensitivity towards their hyperparameters. A common alternative for improving performance is to combine gradient-based objectives with evolutionary methods [13]. Such algorithms allow a population of learners to gain dominant skills [29] from modern-day RL methods and depict robust control while demonstrating scalability. However, their applications do not extend to high-dimensional tasks as a result of sensitivity to mutational hyperparameters which still remains an open problem.

We introduce the Evolution-based Soft Actor Critic (ESAC), an algorithm combining ES with SAC for improved equivalent to SAC and scalability comparable to ES. Our contributions are threefold;

- ESAC abstracts exploration from exploitation by exploring policies in *weight space* using evolutions and exploiting gradient-based knowledge using the SAC framework.
- ESAC makes use of soft winner selection function which, unlike prior selection criteria [13], does not shield winners from mutation. ESAC carries out genetic crossovers in hindsight resulting in dominant skill transfer between members of the population.
- ESAC introduces the novel Automatic Mutation Tuning (AMT) which maximizes the mutation rate of ES in a small clipped region and provides significant hyperparameter robustness without making use of backpropagation updates.

2 RELATED WORK

2.1 Scalable Reinforcement Learning

Recent advances in RL have been successful in tackling sample-efficiency [6] and approximation bias (also known as overestimation bias) which stems from value of estimates approximated by the function approximator. Overestimation bias is a common phenomenon occurring in value-based methods [8, 18, 42] and can be addressed by making use of multiple critics in [5] in the actor-critic framework [23]. This in turn limits scalability of algorithms [12] by increasing the number of gradient-based updates. Moreover, memory complexity of efficient RL methods increases linearly with the expressive power of approximators [14], which in turn hinders scalability of RL to complex control tasks.

2.2 Evolutionary Reinforcement Learning

Intersection of RL and Evolutionary methods has for long been studied in literature [9, 22, 25, 27, 28, 37]. [32] presents the large-scale parallelizable nature of Evolution Strategies (ES). Performance of ES on continuous robot control tasks is comparable to various gradient-based frameworks such as Trust Region Policy Optimization (TRPO) [33] and Proximal Policy Optimization (PPO) [34]. On the other hand, ES falls short of competitive performance resulting in local convergence and is extremely sensitive to mutation hyperparameters.

An alternative to a pure evolution-based approach is a suitable combination of an evolutionary algorithm with a gradient-based method [11], commonly referred to as Evolutionary Reinforcement Learning (ERL) [13]. ERL makes use of selective mutations and genetic crossovers which allow weak learners of the population to inherit skills from strong learners while exploring. ERL methods are scalable to high-dimensional control problems including multi-agent settings [29]. Such an approach is suitable but does not introduce mutation robustness, i.e- the ability to resist mutation noise when converged. Other methods in literature [9] follow a similar approach but are often limited to directional control tasks which require little mutation. Thus, addressing scalability and exploration while preserving higher returns and mutation robustness requires attention from a critical standpoint. Our work is parallel to prior efforts made towards this direction.

3 BACKGROUND

3.1 Reinforcement Learning and Soft Actor-Critic

We review the RL setup wherein an agent interacts with the environment in order to transition to new states and observe rewards by following a sequence of actions. The problem is modeled as a finite-horizon Markov Decision Process(MDP) [39] defined by the tuple $(\mathcal{S}, \mathcal{A}, r, P, \gamma)$ where the state space is denoted by $\mathcal{S}$ and action space by $\mathcal{A}$, r presents the reward observed by agent such that $r : \mathcal{S} \times \mathcal{A} \rightarrow [r_{min}, r_{max}]$, $P : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \rightarrow [0, \infty)$ presents the unknown transition model consisting of the transition probability to the next state $s_{t+1} \in \mathcal{S}$ given the current state $s_t \in \mathcal{S}$ and action $a_t \in \mathcal{A}$ at time step t and γ is the discount factor. We consider a policy $\pi_\theta(a_t|s_t)$ as a function of model parameters θ. Standard RL defines the agent's objective to maximize the expected discounted reward $\mathbb{E}_{\pi_\theta}[\sum_{t=0}^{T} \gamma^t r(s_t, a_t)]$ as a function of the parameters θ. SAC [6] defines an entropy-based[44] objective expressed as in Equation 1.

$$J(\pi_\theta) = \sum_{t=0}^{T} \gamma^t [r(s_t, a_t) + \lambda \mathcal{H}(\pi_\theta(\cdot|s_t))] \tag{1}$$

wherein λ is the temperature coefficient and $\mathcal{H}(\pi_\theta(\cdot|s_t))$ is the entropy exhibited by the policy $\pi(\cdot|s_t)$ in s_t. For a fixed policy, the soft Q-value function can be computed iteratively, starting from any function $Q : \mathcal{S} \times \mathcal{A}$ and repeatedly applying a modified Bellman backup operator $\mathcal{T}^\pi$ given by Equation 2

$$\mathcal{T}^\pi Q(s_t, a_t) = r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1} \sim P}[V(s_{t+1})] \tag{2}$$

where $V(s_t)$ is the soft state value function expressed in Equation 3.

$$V(s_t) = \mathbb{E}_{a_t \sim \pi}[Q(s_t, a_t) - \log(\pi(a_t|s_t))] \tag{3}$$

We consider a parameterized state value function $V_\psi(s_t)$, a soft Q-function $Q_\phi(s_t, a_t)$ and a policy $\pi_\theta(a_t|s_t)$ which can be represented with nonlinear function approximators such as neural networks with ψ, ϕ and θ being the parameters of these networks.

3.2 Evolution Strategies

We review the Evolution Strategies [32] framework which is motivated by natural evolution. ES is a heuristic search procedure in which a population of offsprings is mutated using random perturbations. Upon mutation, the fitness objective corresponding to each member of the population is evaluated and offsprings with greater scores are recombined to form the population for the next generation. Let n be the number of offsprings in the population. The parameter vectors of the model can then be represented as $\theta_{es,(i)}$ such that $i = 1, 2, ..n$. A total of n random perturbations $\epsilon_{(i)}$, $i = 1, 2, ..n$ are sampled from a Gaussian distribution $\mathcal{N}(0, 1)$ in order to mutate $\theta_{es,(i)}$ and evaluate the fitness objective $\mathbb{E}_{\epsilon_{(i)} \sim \mathcal{N}(0,1)}[O(\theta_{es,(i)} + \sigma\epsilon_{(i)})] = \frac{1}{\sigma}\mathbb{E}_{\epsilon_{(i)} \sim \mathcal{N}(0,1)}[O(\theta_{es,(i)} + \sigma\epsilon_{(i)})\epsilon_{(i)}]$. Here, σ is the mutation rate which controls the extent of mutation. In the case of RL, the fitness objecive $O(\theta_{es,(i)})$ is the episodic reward observed by members of the population.

4 A MOTIVATING EXAMPLE: THE DISCRETE CYCLIC MDP

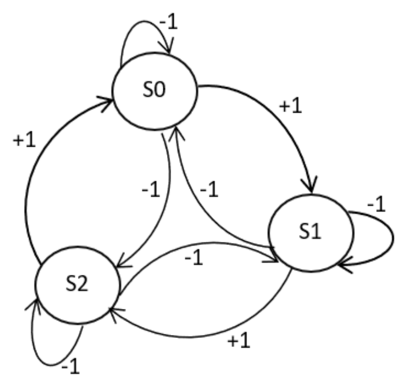

Figure 1: The long-horizon discrete Cyclic MDP. The agent observes a reward of +1 for moving clockwise and -1 otherwise.

We consider a long-horizon discrete cyclic MDP as our motivation for the work. The MDP has a state space $\mathcal{S}^3$ consisting of 3 states- $S0, S1$ and $S2$ and a discrete action space $\mathcal{A}^3$ consisting of 3 actions-clockwise, anticlockwise and stay. The agent starts in state $S0$. The reward function $r : \mathcal{S}^3 \times \mathcal{A}^3$ assigns a reward of +1 for moving clockwise and −1 otherwise. Each episode lasts 2000 timesteps and terminates if the agent reaches the end of horizon or incurs a negative reward. Figure 1 presents the long-horizon discrete cyclic MDP.

The cyclic MDP, being a long-horizon problem, serves as a suitable benchmark for agent's behavior consisting of minimum computational overhead and is a small-scale replication of policy-search for scalable policy-based agents. The environment consists of a global objective which the agents can achieve if they solve the environment by obtaining the maximum reward of +2000. In order to assess evolution-based behavior, we compare the performance of a population of 50 offsprings utilizing ES with PPO [34] and Deep Deterministic Policy Gradient (DDPG) [19], an efficient off-policy RL method. Actions are sampled by the DDPG policy based on a running average of mean and variance of the categorical action distribution. Figure 3 (left) presents the performance of ES in comparison to gradient-based agents in the cyclic MDP averaged over 5 runs. The ES population presents sample efficiency by solving the task within the first 100 episodes. DDPG, on the other hand, starts solving the task much later during training. The use of a deterministic policy allows DDPG to continuously move left whereas in the case of ES, the population carries out exploration in the weight space and moves along the direction of the strong learners. Lastly, PPO finds a local solution and does not converge towards solving the task. Driven by clipped updates, PPO restricts the search horizon in policy space leading to a sub-optimal policy.

ES has proven to be scalable to large-scale and high dimensional control tasks [32]. We assess this property of ES in the cyclic MDP by varying the operational hardware (number of CPUs) [4] and algorithm overhead (population size). We measure the average wall-clock time per episode [1]. As shown in Figure 3 (center), ES is parallelizable in nature and can be scaled up to larger population sizes by reducing the computation time. The large-scale readily parallelizable nature of ES is a convincing characteristic for utilizing CPU-based hardware. However, ES relies on excessively sensitive hyperparameters such as mutation rate. Figure 3 (right) presents the sensitivity of ES to mutation rate within a small range with a constant population size of 50. Varying population size does not present a trend in sensitivity indicating that mutation rate is the dominant hyperparameter governing policy behavior among offsprings. Hyperparameter sensitivity requires attention in the case of RL applications such as for real-world continuous control [20]. These include excessive tuning of parameters and detailed ablation studies. The cyclic MDP highlights this sensitive nature of ES and serves as a motivating example for tackling sensitivity while preserving optimal performance in a scalable manner.

5 EVOLUTION-BASED CONTINUOUS CONTROL

The motivation behind ESAC stems from translating the scalability and tackling the mutation sensitivity of ES observed in discrete cyclic MDP to continuous control tasks. ESAC combines the scalable nature of ES with the limited approximation bias of SAC to yield a CPU-friendly improved algorithm.

5.1 Overview

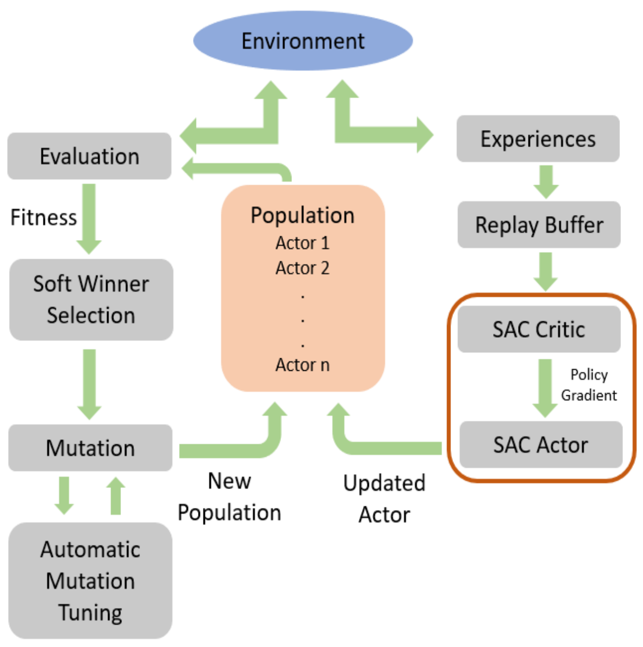

Figure 2: Workflow of ESAC combining ES with SAC. ESAC makes use of soft winner selections, hindsight crossovers and AMT for scalable performance.

Figure 2 provides a high-level schematic of the ESAC algorithm and its components. The population is evaluated in the environment with the fitness metric as episodic rewards obtained by each offspring. Top w winners are then segregated for mutation consisting of ES update followed by crossovers between perturbed offsprings and winners. The new population is formed using crossed-over offsprings and SAC agent. The SAC agent executes its own episodes at fixed timesteps and stores these experiences in a dedicated replay-buffer following policy update. During the SAC update timesteps, ESAC utilizes AMT which maximizes the mutation rate in a clipped region. SAC update timesteps are exponentially annealed to reduce entropy noise and abstract exploration in weight space.

5.2 Algorithm

Algorithm 1 presents the ESAC algorithm. We begin by initializing $\psi, \theta, \theta_{es}, \phi$ being the parameters of state-value function, SAC policy, ES policy and Q-function respectively. We then initialize learning rate for SAC agent α, learning rate of ES population α_{es}, mutation rate σ, p_{sac} which is the probability of SAC updates, $\bar{\psi}$ is the parameter vector of the target value function, ζ is the clip parameter, τ is the target smoothing coefficient, e is the fraction of winners and g is the gradient interval. A population of n actors pop_n is initialized along with an empty replay buffer R. Following the main loop, for each offspring i in the population, we draw a noise vector ϵ_i from $\mathcal{N}(0, 1)$ and perturb the ES policy vector θ_{es} to yield the perturbed parameter vector $\theta_{es,(i)}$ as per the expression $\theta_{es,(i)} + \sigma\epsilon_{(i)}$. $\theta_{es,(i)}$ is then evaluated to yield the fitness $F_{(i)}$ as episodic rewards. These

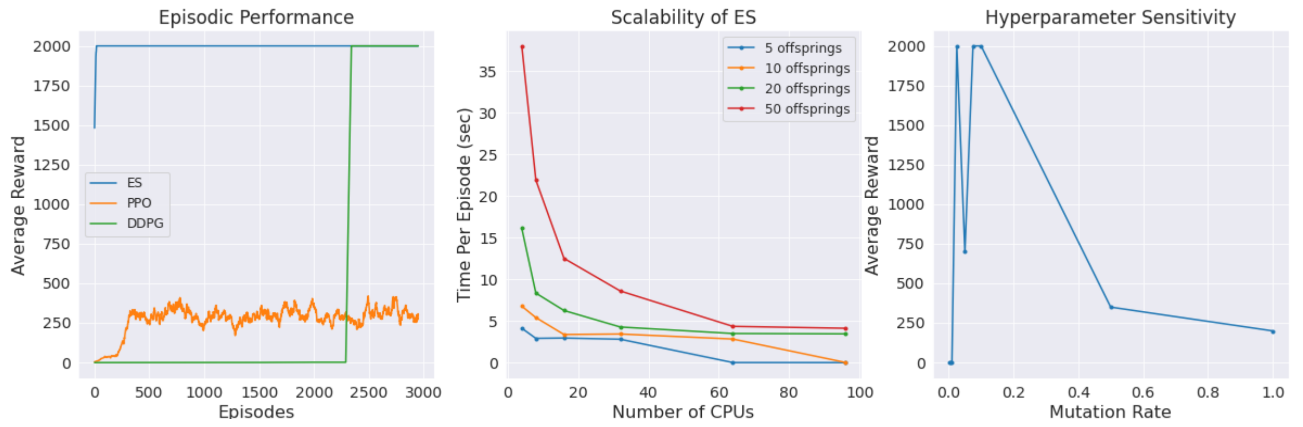

Figure 3: Comparison of ES, DDPG and PPO in the discrete cyclic MDP. ES depicts sample-efficient behavior (left) due to the presence of strong learners in the population. Scalable nature of ES (center) with readily available computational resources allows in reduction of average episode execution time. However, ES is sensitive to hyperparameters (right) which results in inconsistency across different seeds and rigorous fine-tuning.

are collected in a normalized and ranked set F. We now execute soft winner selection wherein the first $w = (n * e)$ offsprings from F are selected for crossovers by forming the set W. The soft winner selection allows dominant skill transfer between winners and next generation offsprings. Mutation is carried out using the ES update [32]. SAC gradient updates are executed at selective gradient intervals g during the training process. p_{sac} is exponentially annealed to reduce entropy noise and direct exploration in the weight space. During each g, the agent executes its own episodes by sampling $a_t \sim \pi_\theta(a_t|s_t)$, observing $r(s_t|a_t)$ and s_{t+1} and storing these experiences in R as a tuple $(s_t, a_t, r(s_t, a_t), s_{t+1})$. Following the collection of experiences, we update the parameter vectors ψ, ϕ and θ by computing $\nabla_\psi J_V(\psi)$, $\nabla_\phi J_Q(\phi_{(i)})$ and $\nabla_\theta J_\pi(\theta)$ where $J_V(\psi)$, $J_Q(\phi_{(i)})$ and $J_\pi(\theta)$ are the objectives of the state-value function, each of the two Q-functions $i \in \{1, 2\}$ and policy as presented in [6] respectively. Gradient updates are followed by AMT update(section 6) which leads to hindsight crossovers between winners in W and ES policy parameter vector θ_{es}. Crossovers are carried out as random replacements between elements of weight vectors. In the case of hindsight crossovers, replacements between weight vector elements of current & immediate previous generations are carried out. This allows the generation to preserve traits of dominant offsprings in hindsight. Finally, the new population is formed using θ, θ_{es} and W.

Algorithm 1 Evolution-based Soft Actor-Critic (ESAC)

```
1:  Initialize parameter vectors $\psi, \bar{\psi}, \theta, \theta_{es}, \phi$
2:  Initialize $\alpha, \alpha_{es}, \sigma, \zeta, \tau, e, g, p_{sac}$
3:  Initialize a population of $n$ actors $pop_n$ and an empty replay buffer $R$
4:  for generation=1,$\infty$ do
5:      for $i \in pop_n$ do
6:          sample $\epsilon_{(i)} \sim \mathcal{N}(0, 1)$
7:          $F_{(i)} \leftarrow$ evaluate $(\theta_{es,(i)} + \sigma\epsilon_{(i)})$ in the environment
8:      end for
9:      normalize and rank $F_{(i)} \in F$
10:     select the first $w = (n * e)$ offsprings from $F$ to form the set of winners $W$
11:     set $\theta_{es} \leftarrow \theta_{es} + \frac{\alpha_{es}}{n\sigma} \sum_{i=1}^{n} F_{(i)}\epsilon_{(i)}$
12:     if $generation$ mod $g == 0$ & $\mu \sim \mathcal{N}(0, 1) < p_{sac}$ then
13:         for each environment step do
14:             $a_t \sim \pi_\theta(a_t|s_t)$
15:             observe $r(s_t|a_t)$ and $s_{t+1} \sim P$
16:             $R \leftarrow R \cup (s_t, a_t, r(s_t, a_t), s_{t+1})$
17:         end for
18:         for each gradient step do
19:             $\psi \leftarrow \psi - \alpha\nabla_\psi J_V(\psi)$
20:             $\phi \leftarrow \phi - \alpha\nabla_\phi J_Q(\phi_{(i)})$ for $i \in \{1, 2\}$
21:             $\theta \leftarrow \theta - \alpha\nabla_\theta J_\pi(\theta)$
22:             $\bar{\psi} \leftarrow \tau\psi + (1 - \tau)\psi$
23:         end for
24:         Update $\sigma$ using Equation 6
25:     end if
26:     crossover between $\theta_{es,(i)}$ and $\theta_{es}$ for $i = 1, 2, ..w$
27:     Form new population $pop_n$ using $\theta, \theta_{es}, W$
28: end for
```

6 AUTOMATIC MUTATION TUNING (AMT)

6.1 Maximization in Weight Space

Maximization of randomness in the policy space is akin to maximization in the weight space as both formulations are a multi-step replica of generalized policy improvement algorithm. This allows one to leverage the more suitable weight space for parallel computations. Policy updates during execution of offsprings require tuning the exploration scheme. To this end, we automatically tune σ starting with the intial value $\sigma_{(1)}$. σ is updated at fixed timesteps in a gradient-ascent manner without making use of backpropagation updates. AMT motivates guided exploration towards the objective as a result of the expansion of the search horizon of population. This in turn enables the agent to maximize rewards *as randomly as* *possible.* AMT makes use of the SmoothL1 (Huber) [10] loss function provided in Equation 4 and the update rule is mathematically expressed in Equation 5.

$$SmoothL1(x_i, y_i) = \begin{cases} 0.5(x_i - y_i)^2, \text{if } |x_i - y_i| < 1 \\ |x_i - y_i| - 0.5, \text{otherwise} \end{cases} \tag{4}$$

$$\sigma_{(t+1)} \leftarrow \sigma_{(t)} + \frac{\alpha_{es}}{n\sigma_{(t)}} SmoothL1(R_{max}, R_{avg}) \tag{5}$$

Here, R_{max} is the reward observed by winner offspring, R_{avg} is the mean reward of the population with $\sigma_{(t)}$ and $\sigma_{(t+1)}$ the mutation rates at timesteps t and $t+1$ respectively. While exploring in weight space, the SmoothL1 loss tends to take up large values. This is indicative of the fact that the deviation between the winner and other learners of the population is significantly high. In order to reduce excessive noise from weight perturbations ϵ_i, we clip the update in a small region parameterized by the new clip parameter ζ. Suitable values for ζ range between 10^{-6} to 10^{-2}. The clipped update is mathematically expressed in Equation 6.

$$\sigma_{(t+1)} \leftarrow \sigma_{(t)} + clip(\frac{\alpha_{es}}{n\sigma_{(t)}} SmoothL1(R_{max}, R_{avg}), 0, \zeta) \tag{6}$$

6.2 Relation to Initial Mutations

The update can be expanded recursively and written in terms of the initial mutation rate $\sigma_{(1)}$. We derive this expression using the AMT and ES update rules,

$$\sigma_{(t)} \leftarrow \sigma_{(t-1)} + \frac{\alpha_{es}}{n\sigma_{(t-1)}} SmoothL1(R_{max,(t-1)}, R_{avg,(t-1)})$$

$$\theta_{(t+1)} \leftarrow \theta_{(t)} + \frac{\alpha_{es}}{n\sigma_{(t)}} \sum_{i=1}^{N} R_i \epsilon_i \tag{7}$$

Using the expression for $\sigma_{(t)}$ in the ES update yields the following,

$$\theta_{(t+1)} \leftarrow \theta_{(t)} + \frac{\alpha_{es}}{n(\sigma_{(t-1)} + \frac{\alpha_{es}}{n\sigma_{(t-1)}} SmoothL1(R_{max,(t-1)}, R_{avg,(t-1)}))} \sum_{i=1}^{n} R_i \epsilon_i \tag{8}$$

$$= \theta_{(t+1)} \leftarrow \theta_{(t)} + \frac{\alpha_{es}}{n\sigma_{(t-1)}(1 + \frac{\alpha_{es}}{n\sigma^2_{(t-1)}} SmoothL1(R_{max,(t-1)}, R_{avg,(t-1)}))} \sum_{i=1}^{n} R_i \epsilon_i \tag{9}$$

$$= \theta_{(t+1)} \leftarrow \theta_{(t)} + \frac{\alpha_{es}}{n\sigma_{(t-1)}\Lambda_{(t-1)}} \sum_{i=1}^{n} R_i \epsilon_i \tag{10}$$

where,

$$\Lambda_{(t-1)} = 1 + \frac{\alpha_{es}}{n\sigma^2_{(t-1)}} SmoothL1(R_{max,(t-1)}, R_{avg,(t-1)}) \tag{11}$$

Expanding $\sigma_{(t-1)}$ using the AMT update rule leads to the following expression for the denominator,

$$n\Lambda_{(t-1)}(\sigma_{(t-2)} + \frac{\alpha_{es}}{n\sigma_{(t-2)}} SmoothL1(R_{max,(t-2)}, R_{avg,(t-2)})) \tag{12}$$

$$n\Lambda_{(t-1)}\sigma_{(t-2)}(1 + \frac{\alpha_{es}}{n\sigma^2_{(t-2)}} SmoothL1(R_{max,(t-2)}, R_{avg,(t-2)})) \tag{13}$$

$$= \theta_{(t+1)} \leftarrow \theta_{(t)} + \frac{\alpha_{es}}{n\Lambda_{(t-1)}\sigma_{(t-2)}\Lambda_{(t-2)}} \sum_{i=1}^{n} R_i \epsilon_i \tag{14}$$

where,

$$\Lambda_{(t-2)} = 1 + \frac{\alpha_{es}}{n\sigma^2_{(t-2)}} SmoothL1(R_{max,(t-2)}, R_{avg,(t-2)}) \tag{15}$$

Expanding this recursively gives us the following,

$$\theta_{(t+1)} \leftarrow \theta_{(t)} + \frac{\alpha_{es}}{n\sigma_{(1)}\Lambda_{(t-1)}\Lambda_{(t-2)}...\Lambda_{(1)}} \sum_{i=1}^{n} R_i \epsilon_i \tag{16}$$

$$= \theta_{(t+1)} \leftarrow \theta_{(t)} + \frac{\alpha_{es}}{n\sigma_{(1)} \prod_{t'=1}^{t-1} \Lambda_{(t')}} \sum_{i=1}^{n} R_i \epsilon_i \tag{17}$$

$$= \theta_{(t+1)} \leftarrow \theta_{(t)} + \frac{\alpha_{es}}{n\sigma_{(1)}\hat{\Lambda}} \sum_{i=1}^{n} R_i \epsilon_i \tag{18}$$

Hence, yielding the AMT update in the form of initial mutation rate $\sigma_{(1)}$. Here, $\hat{\Lambda}$ is defined as the Tuning Multiplier.

6.3 Policy Improvement

It can be additionally shown that AMT, when combined with soft winner selection, leads to policy improvement with high probability among the set of winners. To see this, consider two successive gradient intervals g indexed by (l) and $(l-1)$. Let $p_{(l)}$ and $p_{(l-1)}$ be the probabilities of convergence to the optimal policy $\pi^*_{\theta_{es}}(a_t|s_t)$ in the weight space at (l) and $(l-1)$ respectively.

We start by evaluating the mutation rates at (l) and $(l-1)$ which are given as $\sigma_{(l)} > \sigma_{(l-1)}$. We can now evaluate the probabilities of convergence to $\pi^*_{\theta_{es}}(a_t|s_t)$ as

$$p_{(l)} \geq p_{(l-1)} \tag{19}$$

Using this fact, we can evaluate the winners (indexed by q) in the sorted reward population F.

$$\sum_{q=1}^{w} p^{(q)}_{(l)} F^{(q)}_{(l)} \geq \sum_{q=1}^{w} p^{(q)}_{(l-1)} F^{(q)}_{(l-1)} \tag{20}$$

$$= \mathbf{E}_{F^{(q)}_{(l)} \sim F_{(l)}}[F^{(q)}_{(l)}] \geq \mathbf{E}_{F^{(q)}_{(l-1)} \sim F_{(l-1)}}[F^{(q)}_{(l-1)}] \tag{21}$$

$$= \mathbf{E}[W_{(l)}] \geq \mathbf{E}[W_{(l-1)}] \tag{22}$$

Here, $p^{(q)}_{(l)}$ is the probability of convergence of actor q (having observed reward $F^{(q)}_{(l)}$) to its optimal policy $\pi^{(q),*}_{\theta_{es}}(a_t|s_t)$ at interval (l). $W_{(l)}$ represents the set of winners at (l). The mathematical expression obtained represents that the set of winners $W_{(l)}$ formed at the next gradient interval (l) is at least as good as the previous set of winners $W_{(l-1)}$, i.e.- $\pi^{(q)}_{\theta_{es},(l)}(a_t|s_t) \geq \pi^{(q)}_{\theta_{es},(l-1)}(a_t|s_t)$. This guarantees policy improvement among winners of the population.

Domain	Tasks	ESAC	SAC	TD3	PPO	ES
MuJoCo	HalfCheetah-v2	10277.16±403.63	**10985.90±319.56**	7887.32±532.60	1148.54±1455.64	3721.85±371.36
	Humanoid-v2	5426.82±229.24	**5888.55±44.66**	5392.89±363.11	455.09±213.88	751.65±95.64
	Ant-v2	3465.57±337.81	3693.08±708.56	**3951.76±370.00**	822.34±15.76	1197.69±132.01
	Walker2d-v2	**3862.82±49.80**	3642.27±512.59	3714.89±90.35	402.33±27.38	1275.93±243.78
	Swimmer-v2	**345.44±17.89**	31.68±0.41	110.85±23.02	116.96±0.74	254.42±109.91
	Hopper-v2	**3461.63±118.61**	3048.69±467.21	3255.27±184.18	1296.17±1011.95	1205.73±185.25
	LunarLanderContinuous-v2	**285.79±9.60**	66.52±26.75	273.75±4.51	124.47±11.58	74.41±109.69
	Reacher-v2	-2.01±0.07	-0.50±0.05	-5.12±0.17	**-0.21±0.07**	-4.43±2.06
	InvertedDoublePendulum-v2	**9359.35±0.60**	9257.96±86.54	5603.72±3213.51	88.52±4.73	259.39±36.75
DeepMind Control Suite	HumanoidStand	**805.08±135.67**	759.08±125.67	745.15±291.377	8.41±3.33	10.57±0.30
	HumanoidWalk	**883.00±21.97**	843.00±7.97	686.33±56.23	2.20±0.18	10.59±0.34
	HumanoidRun	**358.82±101.12**	341.45±18.14	291.82±2101.12	2.29±0.16	10.55±0.30
	CheetahRun	**773.14±3.00**	227.66±13.07	765.22±27.93	371.70±19.82	368.62±32.87
	WalkerWalk	**971.02±2.87**	175.75±15.51	941.45±27.01	316.54±79.54	308.94±44.36
	FishUpright	914.96±2.04	285.69±21.03	838.32±34.86	561.39±111.59	**997.58±0.26**

Table 1: Average returns on 15 locomotion tasks from MuJoCo & DeepMind Control Suite. Results are averaged over 5 random seeds with the best performance highlighted in bold. ESAC demonstrates improved performance on 10 out of 15 tasks. Furthermore, ESAC presents consistency across different seeds in the case of large action spaces and sparse rewards indicating the suitability of evolutionary methods to RL and control tasks.

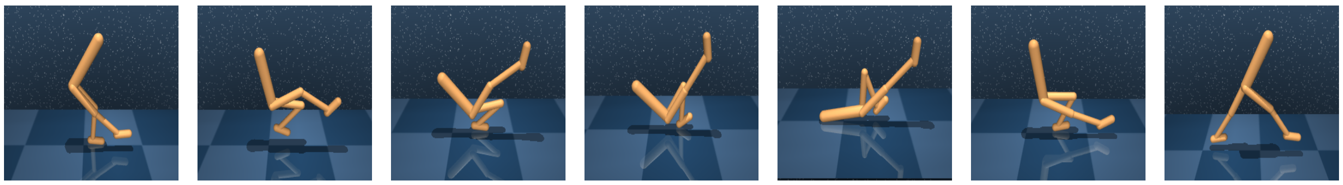

Figure 4: Robust behavior of ESAC observed on the WalkerWalk task. The ESAC policy prevents the walking robot from falling down when the robot loses its balance while walking. The robot successfully retains its initial posture within 50 timesteps. ESAC exhibits robust policies on complex tasks as a result of successive evolutions and hindsight genetic crossovers between winners and actors of the population.

7 EXPERIMENTS

Our experiments aim to evaluate performance, sample efficiency, scalability and mutation sensitivity of ESAC. Specifically, we aim to answer the following questions-

- How does the algorithm compare to modern-day RL methods for complex tasks?
- What kind of behaviors do evolution-driven policies present under varying task dynamics?
- How do evolutionary operations impact scalability in the presence of gradients?
- Which components of the method contribute to sensitivity and scalability?

7.1 Performance

We assess performance and sample efficiency of ESAC with state-of-the-art RL techniques including on-policy and off-policy algorithms. We compare our method to ES [32]; SAC [6]; Twin-Delayed Deep Deterministic Policy Gradient (TD3)[5] and PPO [34] on a total of 9 MuJoCo [41] and 6 DeepMind Control Suite [40] tasks. We refer the reader to Appendix A.1 for complete results. The tasks considered consist of sparse rewards and high-dimensional action spaces including 4 different versions of Humanoid. Additionally, we consider the LunarLander continuous task as a result of its narrow basin of learning. All methods were implemented using author-provided implementations except for ES in which Virtual Batch Normalization [31] was omitted as it did not provide significant performance boosts and hindered scalability.

Agents were trained in OpenAI's Gym environments [2] framework for a total of 5 random seeds. Training steps were interleaved with validation over 10 episodes. For all agents we use nonlinear function approximators as neural networks in the form of a multilayer architecture consisting of 2 hidden layers of 512 hidden units each activated with ReLU [26] nonlinearity and an output layer with tanh activation. We use this architecture as a result of its consistency in baseline implementations. We use Adam [15] as the optimizer (refer to Appendix B for hyperparameters). For ESAC and SAC, we use a Diagonal Gaussian (DG) policy [36] without automatic entropy tuning for a fair comparison. Training of gradient-based methods was conducted on 4 NVIDIA RTX2070 GPUs whereas for ES and ESAC, a total of 64 AMD Ryzen 2990WX CPUs were used.

Table 1 presents total average returns of agents on all 15 tasks considered for our experiments. ESAC demonstrates improved returns on 10 out of 15 tasks. ESAC makes use of evolution-based weight-space exploration to converge to robust policies in tasks where SAC often learns a sub-optimal policy. Moreover, utilization of evolutionary operations demonstrates consistency across different seeds for high-dimensional Humanoid tasks indicating large-scale suitability of the method to complex control.

7.2 Behaviors

Combination of RL and evolutionary methods provides suitable performance on control benchmarks. It is essential to assess the behaviors learned by agents as a result of weight-space exploration. We turn our attention to observe meaningful patterns in agent's behavior during its execution in the environment. More specfically, we aim to evaluate the robustness of ESAC scheme which promises efficacious policy as a result of effective exploration. We initialize a learned ESAC policy on the WalkerWalk task and place it in a challenging starting position. The Walker agent stands at an angle and must prevent a fall in order to complete the task of walking suitably as per the learned policy.

Figure 4 demonstrates the behavior of the Walker agent during its first 100 steps of initialization. The agent, on its brink of experiencing a fall, is able to gain back its balance and retain the correct posture for completing the walking task. More importantly, the agent carries out this manoeuvre within 50 timesteps and quickly gets back on its feet to start walking. Figure 4 is an apt demonstration of robust policies learned by the ESAC agent. Dominant skill transfer arising from hindsight crossovers between winners and offsprings provisions effective exploration in weight space.

7.3 Scalability

We assess scalability of our method with ES on the basis of hardware resources and algorithm overhead. We vary the number of CPUs by keeping other training parameters constant. Parallelization on multiple CPU-based resources is readily available and cost-efficient in comparison to a single efficient GPU resource. We also vary number of offsprings in the population by fixing CPU resources. Out of mutation rate σ and population size n, n governs the computational complexity of ES with σ being a scalar value. Thus, assessing variation w.r.t n provides a better understanding of resource utility and duration. For both experiments, we train the population for 10^6 steps and average the wall-clock time per episode.

Note that another effective way to demonstrate scalability is by monitoring overall time taken to complete the training tasks [32]. However, this often tends to vary as initial learning periods have smaller episode lengths which does not compensate for fixed horizons of 1000 steps in MuJoCo.

Figure 5 presents the scalable nature of ESAC equivalent to ES on the MuJoCo and LunarLanderContinuous tasks. Average wall-clock time per episode is reduced utilizing CPU resources which is found to be favourable for evolution-based methods. Moreover, the variation depicts consistency with the increasing number of members in the population indicating large-scale utility of the proposed method. A notable finding here is that although ESAC incorporates gradient-based backpropagation updates, it is able to preserve its scalable nature by making use of evolutions as dominant operations during the learning process. This is in direct contradiction to prior methods [13]which demonstrate reduced sample-efficiency and the need for significant tuning when combining RL with scalable evolutionary methods. Reduction in the number of SAC updates by exponentially annealing the gradient interval allows ESAC to reduce computation times and simultaneously explore using AMT.

8 ABLATION STUDY

8.1 Mutation Sensitivity

ES presents sensitivity to σ which is addressed by making use of AMT in ESAC. The AMT update gradually increases mutation rate σ using clip parameter ζ as learning progresses. To assess the robustness of policies to mutations, we vary the clip parameter ζ and study its impact on task returns.

Figure 6 (left) presents the variation of average normalized rewards with different values of ζ for HalfCheetah-v2 and Ant-v2 tasks. Each experiment was run for 1 million steps. The population presents robustness and performance improvement for small values with the optimal range being 10^{-4} to 10^{-2}. On the other hand, sensitivity is observed in the 10^{-1} to 1 region which accounts for larger updates with high variance. Hyperparameter variation is limited to a smaller region, in contrast to a wider spread of σ in the ES update. Offsprings remain robust to significantly large values of ζ due to early convergence of the population at the cost of poor performance among weak learners of the population. However, this is addressed by making use of hindsight crossovers which allow simultaneous transfer of dominant traits.

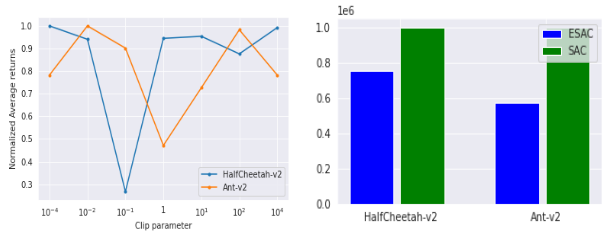

Figure 6: Left: Mutation sensitivity of ESAC, Right: Number of backprop updates in ESAC compared to SAC

8.2 Number of Updates

The main computation bottleneck in SAC arises from the number of backprop updates. This is tackled by exponentially annealing these updates and increasing winner-based evolutions and crossovers for transferring skills between SAC agent and ES offspings.

Figure 6 (right) presents a comparison between the number of backprop updates carried out using SAC and ESAC during the training phase of HalfCheetah-v2 and Ant-v2 tasks. Results for the updates are averaged over 3 random seeds. ESAC executes lesser number of updates highlighting its computationally efficient nature and low dependency on a gradient-based scheme for monotonic improvement. Complete results can be found in subsection A.2.

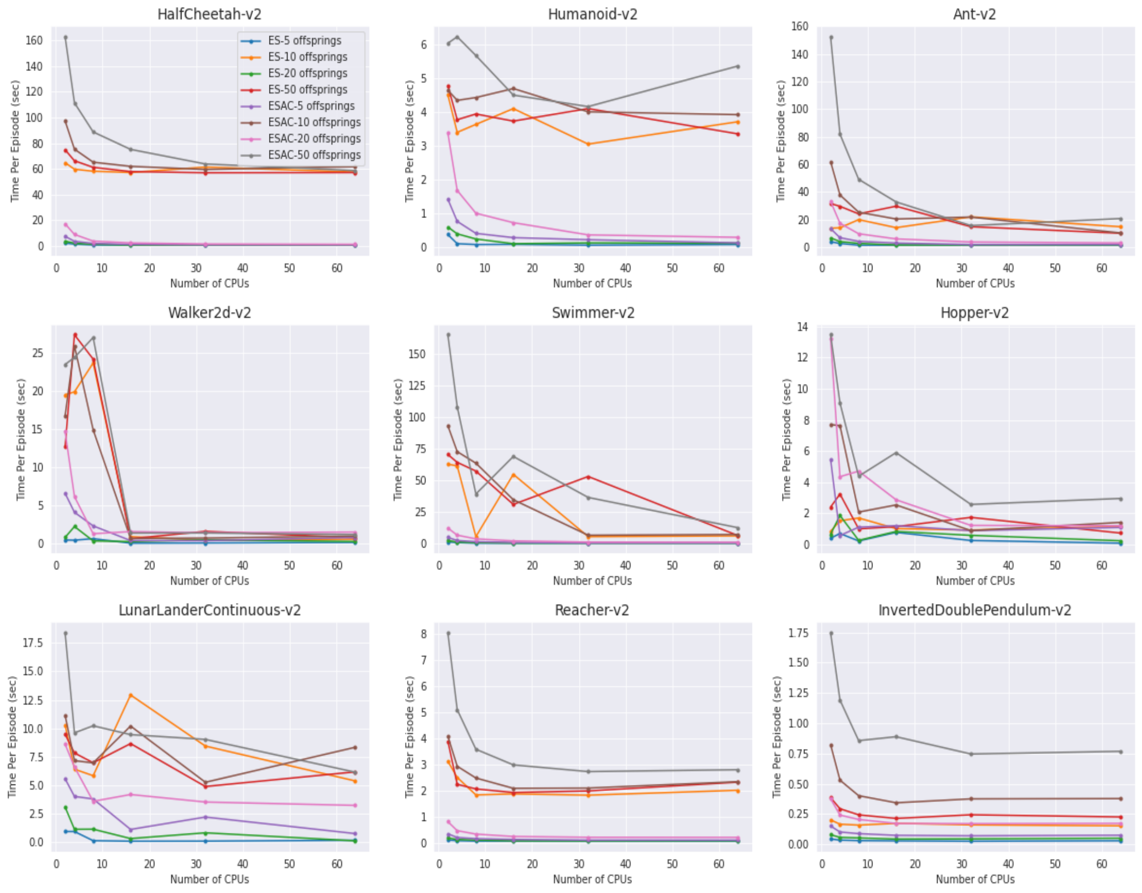

Figure 5: Variation of average time per episode (in seconds) with the number of operational CPUs and population size (in legend) for locomotion tasks from the MuJoCo benchmark. ESAC demonstrates equivalent scalability as ES by providing reductions in episode execution times. Reduction in computational time is found to be approximately quadratic which is computationally efficient for RL in comparison to linear variations for high-end GPU machines.

9 DISCUSSION

9.1 Conclusion

In this paper, we introduced ESAC which combines the scalable nature of ES with low approximation bias and improved performance of SAC. The scheme abstracts exploration from exploitation by searching for policies in the parameter space and learning behaviors using gradient signals. ESAC addresses the problem of mutation-sensitive evolutions by introducing AMT which maximizes the mutation rate of evolutions in a small clipped region as the SAC updates are exponentially decayed. The scheme further incorporates soft winner selection and genetic hindsight crossovers between current and previous generations. This leads to preservation of dominant traits within the offsprings. ESAC demonstrates improved performance on 10 out of 15 locomotion tasks including different versions of Humanoid. Additionally, the method yields robust policies and highlights scalability comparable to ES by reducing the average wall-clock time.

9.2 Limitations

While a combination of evolutionary and gradient operations is suitable from the computational perspective, its effect on generalization needs to be well understood. High variance of the SAC agent under sparse rewards requires consistent optimal behavior. This often hinders learning of optimal policies with evolutions. The above can be addressed by combining the framework with a meta-controller or using a more sophisticated architecture such as a master-slave framework [38]. We leave this for future work.

ACKNOWLEDGMENTS

We thank Xiao Qi Shi, Konstantinos Plataniotis and Yuri Lawryshyn for helpful discussions during early stages of the project. The authors are also thankful to Seyed Kamyar Seyed Ghasemipour and anonymous reviewers for providing valuable feedback on the paper. KS was supported by the Edward S. Rogers Graduate Scholarship and Borelis AI, RBC's research institute.

REFERENCES

[1] Prasanna Balaprakash, Romain Egele, Misha Salim, Stefan Wild, Venkatram Vishwanath, Fangfang Xia, Tom Brettin, and Rick Stevens. 2019. Scalable reinforcement-learning-based neural architecture search for cancer deep learning research. *Proceedings of the SC* (Nov 2019).

[2] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. OpenAI Gym. cite arxiv:1606.01540.

[3] Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. 2019. Exploration by random network distillation. In *ICLR*.

[4] Lasse Espeholt, Raphaël Marinier, Piotr Stanczyk, Ke Wang, and Marcin Michalski. 2020. SEED RL: Scalable and Efficient Deep-RL with Accelerated Central Inference. In *ICLR*.

[5] Scott Fujimoto, Herke van Hoof, and David Meger. 2018. Addressing Function Approximation Error in Actor-Critic Methods. *CoRR* abs/1802.09477 (2018). arXiv:1802.09477

[6] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. 2018. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. *CoRR* abs/1801.01290 (2018). arXiv:1801.01290

[7] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. 2019. Dream to Control: Learning Behaviors by Latent Imagination. arXiv:1912.01603 [cs.LG]

[8] Hado V. Hasselt. 2010. Double Q-learning. In *NIPS 23*.

[9] Rein Houthooft, Yuhua Chen, Phillip Isola, Bradly Stadie, Filip Wolski, Jonathan Ho, and Pieter Abbeel. 2018. Evolved Policy Gradients. In *NIPS*. 5400–5409.

[10] Peter J. Huber. 1964. Robust estimation of a location parameter. *Annals of Mathematical Statistics* 35, 1 (March 1964), 73–101.

[11] Perttu Hämäläinen, Amin Babadi, Xiaoxiao Ma, and Jaakko Lehtinen. 2018. PPO-CMA: Proximal Policy Optimization with Covariance Matrix Adaptation. *CoRR* abs/1810.02541 (2018). arXiv:1810.02541

[12] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, and Sergey Levine. 2018. Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation. In *CoRL (PMLR, Vol. 87)*. PMLR, 651–673.

[13] Shauharda Khadka and Kagan Tumer. 2018. Evolution-Guided Policy Gradient in Reinforcement Learning. In *NIPS (NIPS'18)*. Curran Associates Inc., Red Hook, NY, USA, 1196–1208.

[14] Shauharda Khadka, Connor Yates, and Kagan Tumer. 2018. A Memory-Based Multiagent Framework for Adaptive Decision Making. In *17th ICAAMS*.

[15] Diederik P. Kingma and Jimmy Ba. [n.d.]. Adam: A Method for Stochastic Optimization. 3rd International Conference for Learning Representations, San Diego, 2015.

[16] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *NIPS*. Curran Associates, Inc., 1097–1105.

[17] Tejas D. Kulkarni, Karthik R. Narasimhan, Ardavan Saeedi, and Joshua B. Tenenbaum. 2016. Hierarchical Deep Reinforcement Learning: Integrating Temporal Abstraction and Intrinsic Motivation. In *NIPS*. 3682–3690.

[18] Qingfeng Lan, Yangchen Pan, Alona Fyshe, and Martha White. 2020. Maxmin Q-learning: Controlling the Estimation Bias of Q-learning. In *International Conference on Learning Representations*.

[19] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Manfred Otto Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2015. Continuous control with deep reinforcement learning. *CoRR* abs/1509.02971 (2015).

[20] A. Rupam Mahmood, Dmytro Korenkevych, Gautham Vasan, William Ma, and James Bergstra. 2018. Benchmarking Reinforcement Learning Algorithms on Real-World Robots. *CoRR* abs/1809.07731 (2018). arXiv:1809.07731

[21] Parvin Malekzadeh, Mohammad Salimibeni, Arash Mohammadi, Akbar Assa, and Konstantinos N. Plataniotis. 2020. MM-KTD: Multiple Model Kalman Temporal Differences for Reinforcement Learning. arXiv:2006.00195 [cs.LG]

[22] Thomas Miconi, Aditya Rawal, Jeff Clune, and Kenneth O. Stanley. 2020. Backpropamine: training self-modifying neural networks with differentiable neuromodulated plasticity. arXiv:2002.10585

[23] Volodymyr Mnih, Adriá Puigdoménech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous Methods for Deep Reinforcement Learning. *CoRR* abs/1602.01783 (2016). arXiv:1602.01783

[24] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. 2013. Playing Atari with Deep Reinforcement Learning. *CoRR* abs/1312.5602 (2013). arXiv:1312.5602

[25] David E Moriarty and Risto Mikkulainen. 1996. Efficient reinforcement learning through symbiotic evolution. *Machine Learning* 22, 1-3 (1996), 11–32.

[26] Vinod Nair and Geoffrey E. Hinton. 2010. Rectified Linear Units Improve Restricted Boltzmann Machines. In *ICML*, Johannes Fürnkranz and Thorsten Joachims (Eds.). 807–814.

[27] Paolo Pagliuca, Nicola Milano, and Stefano Nolfi. 2019. Efficacy of Modern Neuro-Evolutionary Strategies for Continuous Control Optimization. arXiv:1912.05239

[28] Aloïs Pourchot and Olivier Sigaud. 2018. CEM-RL: Combining evolutionary and gradient-based methods for policy search. arXiv:1810.01222

[29] Golden Rockefeller, Shauharda Khadka, and Kagan Tumer. 2020. Multi-Level Fitness Critics for Cooperative Coevolution. In *ICAAMS*.

[30] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. 1986. Learning Representations by Back-propagating Errors. *Nature* 323, 6088 (1986), 533–536.

[31] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, Xi Chen, and Xi Chen. 2016. Improved Techniques for Training GANs. In *29th NIPS*.

[32] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. 2017. Evolution Strategies as a Scalable Alternative to Reinforcement Learning. arXiv:1703.03864

[33] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. 2015. Trust Region Policy Optimization. In *ICML (PMLR, Vol. 37)*, Francis Bach and David Blei (Eds.). PMLR, Lille, France, 1889–1897.

[34] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. *CoRR* abs/1707.06347 (2017).

[35] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, and et. al. 2016. Mastering the Game of Go with Deep Neural Networks and Tree Search. *Nature* 529, 7587 (Jan. 2016), 484–489.

[36] Aravind Srinivas, Michael Laskin, and Pieter Abbeel. 2020. Curl: Contrastive unsupervised representations for reinforcement learning. *arXiv preprint arXiv:2004.04136* (2020).

[37] Kenneth O. Stanley and Risto Miikkulainen. 2002. Evolving Neural Networks through Augmenting Topologies. *Evolutionary Computing* 10, 2 (June 2002), 99–127.

[38] Karush Suri and Rinki Gupta. 2018. Transfer Learning for sEMG-based Hand Gesture Classification using Deep Learning in a Master- Slave Architecture. *IC3I* (Oct 2018).

[39] Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction.*

[40] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, Timothy P. Lillicrap, and Martin A. Riedmiller. 2018. DeepMind Control Suite. *CoRR* abs/1801.00690 (2018).

[41] Emanuel Todorov, Tom Erez, and Yuval Tassa. 2012. Mujoco: A physics engine for model-based control. In *IROS, 2012 IEEE/RSJ International Conference on*. IEEE, 5026–5033.

[42] Hado van Hasselt, Arthur Guez, and David Silver. 2015. Deep Reinforcement Learning with Double Q-learning. (2015). cite arxiv:1509.06461Comment: AAAI 2016.

[43] Yuhuai Wu, Elman Mansimov, Shun Liao, Roger Grosse, and Jimmy Ba. 2017. Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation. arXiv:1708.05144 [cs.LG]

[44] Brian D. Ziebart. 2010. *Modeling Purposeful Adaptive Behavior with the Principle of Maximum Causal Entropy.* Ph.D. Dissertation. USA. Advisor(s) Bagnell, J. Andrew.