Collaborative Evolutionary Reinforcement Learning

Shauharda Khadka [1 2] **Somdeb Majumdar** [1] **Tarek Nassar** [1] **Zach Dwiel** [1] **Evren Tumer** [1] **Santiago Miret** [1]
Yinyin Liu [1] **Kagan Tumer** [2]

Abstract

Deep reinforcement learning algorithms have been successfully applied to a range of challenging control tasks. However, these methods typically struggle with achieving effective exploration and are extremely sensitive to the choice of hyperparameters. One reason is that most approaches use a noisy version of their operating policy to explore - thereby limiting the range of exploration. In this paper, we introduce Collaborative Evolutionary Reinforcement Learning (CERL), a scalable framework that comprises a portfolio of policies that simultaneously explore and exploit diverse regions of the solution space. A collection of learners - typically proven algorithms like TD3 - optimize over varying time-horizons leading to this diverse portfolio. All learners contribute to and use a shared replay buffer to achieve greater sample efficiency. Computational resources are dynamically distributed to favor the best learners as a form of online algorithm selection. Neuroevolution binds this entire process to generate a single emergent learner that exceeds the capabilities of any individual learner. Experiments in a range of continuous control benchmarks demonstrate that the emergent learner significantly outperforms its composite learners while remaining overall more sample-efficient - notably solving the Mujoco Humanoid benchmark where all of its composite learners (TD3) fail entirely in isolation.

1. Introduction

Reinforcement learning (RL) has been successfully applied to a number of challenging tasks, ranging from arcade games (Mnih et al., 2015; 2016), board games (Silver et al., 2016) to robotic control (Andrychowicz et al., 2017; Lillicrap et al., 2015). A driving force behind the explosion of RL applications is its integration with powerful non-linear function approximators like deep neural networks. This partnership, often referred to as Deep Reinforcement Learning (DRL), has enabled RL to successfully extend to tasks with high-dimensional input and action spaces. However, widespread adoption of these techniques to real-world problems is still limited by two major challenges: the difficulty in achieving effective exploration and brittle convergence properties that require careful tuning of the hyperparameters by a designer.

First, exploration is a key component for successful reinforcement learning. It enables an agent to learn good policies and avoid converging prematurely to local optima. Designing exploration strategies that lead to a diverse set of experiences remains a key challenge for DRL operating on high dimensional action and state spaces (Plappert et al., 2017). Many methods have been formulated to address this issue, ranging from intrinsic motivation (Bellemare et al., 2016), count-based exploration (Ostrovski et al., 2017; Tang et al., 2017), curiosity (Pathak et al., 2017) and variational information maximization (Houthooft et al., 2016). Further, a parallel class of techniques emphasize exploration by adding noise directly to the parameters of the agent (Fortunato et al., 2017; Plappert et al., 2017; Khadka & Tumer, 2018). However, each of these techniques either relies on supplementary structures or introduces task-specific parameters that need to be tuned rigorously. A general exploration strategy that is universally applicable across tasks and learning algorithms remains an active area of research.

Second, DRL approaches are often notoriously sensitive to their hyperparamaters (Henderson et al., 2017; Islam et al., 2017) and demonstrate brittle convergence properties (Haarnoja et al., 2018). This is particularly true for off-policy approaches that use a replay buffer to leverage past experiences (Bhatnagar et al., 2009; Duan et al., 2016). In part, this sensitivity is coupled with the difficulty of effective exploration. A standard reinforcement learner employs a noisy version of its operating policy as its behavioral policy for exploration. This puts the burden of both exploitation and exploration onto the same set of hyperparameters.

In this paper, we introduce Collaborative Evolutionary Re-

[1]Intel AI Lab [2]Collaborative Robotics and Intelligent Systems Institute, Oregon State University. Correspondence to: Shauharda Khadka <shauharda.khadka@intel.com>, Somdeb Majumdar <somdeb.majumdar@intel.com>.

Proceedings of the 36^{th} International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019.

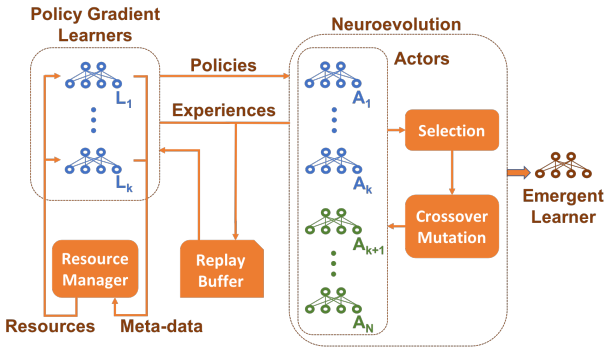


Figure 1. High level schematic of CERL. A portfolio of policy gradient learners operate in parallel to neuroevolution for collective exploration, while a shared replay buffer enables collective exploitation. Resource Manager drives this process by dynamically allocating computational resources amongst the learners.

inforcement Learning (CERL), a scalable framework that leverages a portfolio of learners that learn with different time-horizons to explore different parts of the solution space while remaining loyal to the task. This process is directed by a resource manager that dynamically re-distributes computational resources amongst the learners - favoring the best as a form of online algorithm selection. The diverse set of experiences generated by this adaptive process are stored in a shared replay buffer for collective exploration enabling better sample efficiency.

Figure 1 illustrates CERL's multi-layered learning approach where each learner exploits the data generated by a diversity of "behavioral policies" stemming from other learners in the portfolio. An evolutionary population operating in parallel augments this process by extending exploration to the parameter space of policies through mutation. Evolution also introduces redundancies in the population to stabilize learning, intermixes sub-components within policies through crossover, and binds the entire underlying process to generate an emergent learner that exceeds the sum of its parts. Experiments in a range of continuous control benchmarks demonstrate that CERL inherits the best of its composite learners while remaining overall more sample-efficient.

2. Background

A standard reinforcement learning setting is often formalized as a Markov Decision Process (MDP) and consists of an agent interacting with an environment over a finite number of discrete time steps. At each time step t, the agent observes a state s_t and maps it to an action a_t using its policy π. The agent receives a scalar reward r_t and transitions to the next state s_{t+1}. The process continues until the agent reaches a terminal state marking the end of an episode. The return, $R_t = \sum_{n=1}^{\infty} \gamma^k r_{t+k}$ is the total return from time step t with discount factor $\gamma \in (0, 1]$. The goal of the agent is to maximize this expected return. The state-value function $\mathcal{Q}^\pi(s, a)$ describes the expected return from state s after taking action a and subsequently following policy π.

2.1. Twin Delayed Deep Deterministic Policy Gradients

Policy gradient methods re-frame the goal of maximizing the expected return as the minimization of a loss function $L(\theta)$ where θ encapsulates the agent parameters. A widely used policy gradient method is Deep Deterministic Policy Gradient (DDPG) (Lillicrap et al., 2015), a model-free RL algorithm developed for working with continuous, high dimensional actions spaces. Recently, Fujimoto et al. extended DDPG to Twin Delayed DDPG (TD3), (Fujimoto et al., 2018b) addressing the well-known overestimation problem of the former. TD3 was shown to significantly improve upon DDPG and is the state-of-the-art, off-policy algorithm for model-free deep reinforcement learning in continuous action spaces. TD3 uses an actor-critic architecture (Sutton & Barto, 1998) maintaining a deterministic policy (actor) $\pi : \mathcal{S} \rightarrow \mathcal{A}$, and two distinct action-value function approximations (critics) $\mathcal{Q} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}_i$.

Each critic independently approximates the actor's action-value function $\mathcal{Q}^\pi$. The actor and the critics are parameterized by (deep) neural networks with θ^π, $\theta_a^\mathcal{Q}$, and $\theta_b^\mathcal{Q}$ respectively. A separate copy of the actor π' and critics: $\mathcal{Q}'_a$ and $\mathcal{Q}'_b$ are kept as target networks for stability. These networks are updated periodically using the actor π and critic networks: $\mathcal{Q}_a$ and $\mathcal{Q}_b$ regulated by a weighting parameter τ and a delayed policy update frequency d.

A behavioral policy is used to explore the environment during training. The behavioral policy is simply a noisy version of the policy: $\pi_b(s) = \pi(s) + \mathcal{N}(0, 1)$ where $\mathcal{N}$ is white Gaussian noise. After each action, the tuple (s_t, a_t, r_t, s_{t+1}) containing the current state, actor's action, observed reward and the next state, respectively, is saved into a **replay buffer** $\mathcal{R}$. The actor and critic networks are updated by randomly sampling mini-batches from $\mathcal{R}$. The critic is trained by minimizing the loss function:

$$L_i = \frac{1}{T} \sum_i (y_i - \mathcal{Q}_i(s_i, a_i|\theta^\mathcal{Q}))^2$$

$$\text{where } y_i = r_i + \gamma \min_{j=1,2} \mathcal{Q}'_j(s_{i+1}, \tilde{a} |\theta^{\mathcal{Q}'_j})$$

where $\tilde{a}$ is the noisy action computed by adding Gaussian noise clipped to between $-c$ and c. $\tilde{a} = \pi'(s_{i+1}|\theta^{\pi'}) + \epsilon$, $clip\big(\epsilon \sim \mathcal{N}(\mu, \sigma^2) - c, c\big)$

This noisy action used for the Bellman update smoothens the value estimate by bootstrapping from similar state-action value estimates. It serves to make the policy smooth and addresses overfitting of the deterministic policy. The actor is trained using the sampled policy gradient:

$$\nabla_{\theta^\pi} J \sim \frac{1}{T} \sum \nabla_a \mathcal{Q}(s, a|\theta_a^\mathcal{Q})|_{s=s_i, a=a_i} \nabla_{\theta^\pi} \pi(s|\theta^\pi)|_{s=s_i}$$

The sampled policy gradient with respect to the actor's parameters θ^π is computed by backpropagation through the combined actor and critic network.

2.2. Evolutionary Algorithms

Evolutionary algorithms (EAs) are a class of search algorithms characterized by three primary operators: new solution generation, solution alteration and selection (Fogel, 2006; Spears et al., 1993). These operations are applied on a population of candidate solutions to continually generate new solutions while retaining promising ones. The selection operation is generally probabilistic, where better solutions with higher fitness values have a higher probability of being selected. Assuming that higher fitness values are representative of good solution quality, the overall quality of solutions will improve with each generation. In this work, each individual in the evolutionary algorithm is a deep neural network representing a policy π. Mutation is implemented as random perturbations to the weights (genes) of these neural networks. The evolutionary framework used here is closely related to evolving neural networks and is often referred to as neuroevolution (Floreano et al., 2008; Lüders et al., 2017; Risi & Togelius, 2017; Stanley & Miikkulainen, 2002).

3. Related Work

A closely related work to CERL is Population-based Training (PBT) (Jaderberg et al., 2017), which employs a population to jointly optimize models and its associated hyperparameters online. However, unlike CERL, PBT does not dynamically redistribute computational resources amongst its learners; instead, it relies entirely on its evolutionary process for learner selection. Additionally, learners in PBT are isolated and do not share experiences with each other for collective exploitation - a key mechanism in CERL for the retention of sample-efficiency. Collective exploitation of a diverse set of experiences is a popular idea, particularly in recent literature. Colas et al. used an evolutionary method (Goal Exploration Process) to generate diverse samples followed by a policy gradient method for fine-tuning the policy parameters (Colas et al., 2018) while Khadka and Tumer incorporated the two processes to run concurrently formulating a Lamarckian framework (Khadka & Tumer, 2018). From an evolutionary perspective, this is closely related to the idea of incorporating learning with evolution (Ackley & Littman, 1991; Drugan, 2018; Turney et al., 1996).

Another facet of CERL is algorithm selection (Gagliolo & Schmidhuber, 2006; Smith-Miles, 2009; Rice, 1976) - an idea that has been explored extensively in past literature. Lagoudakis and Littman formulated algorithm selection as an MDP and used Q-learning to solve classic order statistic selection and sorting problems (Lagoudakis & Littman, 2000). Cauwet et al. addressed noisy optimization using a portfolio of online reinforcement learning algorithms (Cauwet et al., 2014). Conversely, Laroche and Feraud introduced Epochal Stochastic Bandit Algorithm Selection (ESBAS), which tackled algorithm selection in reinforcement learning itself, formulating it as a K-armed stochastic bandit problem (Laroche & Feraud, 2017). The resource-manager in CERL closely builds on this formulation to inherit its good exploration-exploitation trade-off properties. However, unlike ESBAS, CERL leads to soft algorithm selection - carried out through the allocation of computation resources rather than a hard binary selection.

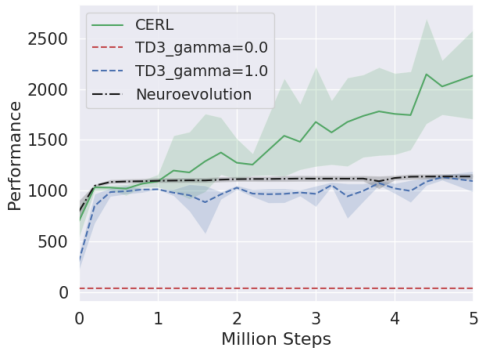


Figure 2. Comparative performance of Neuroeovlution, TD3 ($\gamma = 0.0, 1.0$) and CERL (built using them) in the Hopper benchmark.

4. Motivating Example

Consider the Hopper task from OpenAI gym (Brockman et al., 2016), a classic continuous control benchmark used widely in recent DRL literature (Duan et al., 2016; Islam et al., 2017; Henderson et al., 2017; Schulman et al., 2015). Here, the goal is to make a two-dimensional, one-legged robot hop as fast as possible without falling. The task has a state space dimension of $\mathcal{S} = 11$ and action space dimension of $\mathcal{A} = 3$. TD3 has been shown to solve this problem fairly easily (Fujimoto et al., 2018a) (also shown in Figure 5 in Section 6). However, TD3 solves this problem with a tuned discount rate ($\gamma = 0.99$). It is interesting how sensitive this performance would be to varying choices of a discount rate (γ), including ones that are clearly sub-optimal.

Figure 2 shows the comparative performance of TD3 ($\gamma = 0.0$), TD3 ($\gamma = 1.0$), neuroevolution and our proposed approach: CERL built using the two TD3 variations as its learners. TD3 ($\gamma = 0.0$) represents an extremely greedy learner whose optimization horizon is limited to its immediate reward. On the contrary, TD3 ($\gamma = 1.0$) represents a long-term learner whose optimization horizon is virtually infinite. However, since it seeks to optimize a return which is a function of all future action and states in the trajectory, it learns with significant amount of variance. Both learners represent the extreme ends of the spectrum and would

not be expected to learn well. Figure 2 corroborates this expectation: TD3 ($\gamma = 0.0$) fails to entirely learn as the most greedy action with respect to the immediate reward is rarely aligned with the cumulative episode-wide return. TD3 ($\gamma = 1.0$), on the other hand, has a reward ceiling of 1000 - imposed by the variance of its computed return. Similarly, neuroevolution on its own also fails to solve the task within the 5 million steps tested. However, CERL, which is built directly on top of these learners, is able to continue learning beyond this - reaching a score of 2136 ± 512.

While each of the learners fails to solve the problem individually, they collaboratively succeed in solving it under the CERL framework. A key reason here is that each learner fails when required to simultaneously exploit well and produce good behavioral policies that explore the space well. Being able to do both is key to solving the problem and tuning the discount rate is akin to finding this trade-off. CERL provides an alternate approach to finding this trade-off - by employing both learners to explore the space while dynamically distributing the resources to the better performer for effective exploitation. Even when a learner is ill-suited for solving the task by itself, it can serve to be a key 'behavioral policy' that explores critical parts of the search space and generates experiences which are key to learning well on the task. CERL exploits these diversities to define an emergent learner that surpasses the sum of its parts.

5. Collaborative Evolutionary Reinforcement Learning

Algorithm 1 Object Learner

procedure INITIALIZE(γ)
  Set discount rate=γ, $count$=0 and value v=0
  Initialize actor π and critic $\mathcal{Q}$ with weights θ^{π} and $\theta^{\mathcal{Q}}$, respectively
  Initialize target actor π' and critic $\mathcal{Q}'$ with weights $\theta^{\pi'}$ and $\theta^{\mathcal{Q}'}$, respectively

The principal idea behind Collaborative Evolutionary Reinforcement Learning (CERL) is to incorporate the strengths of multiple learners, each optimizing over varying time-horizons of the underlying task (MDP). While a specific learner is unlikely to be an optimal choice for the task throughout the learning process, a diverse collection of learners is significantly more likely to be so. This is particularly true for exploration, where different learners can contribute a diverse set of behavioral policies while remaining loyal to the task. A shared replay buffer ensures that all learners exploit this diverse data generated. A resource manager supervises this process by dynamically re-distributing computational resources to favor the better performing learners. Finally, this entire underlying apparatus is bound together by evolution which serves to integrate the best policies, explore in the parameter space and exploit any decomposition in the policy space with crossover operands. The emergent learner combines the best of its underlying composite processes, leading to a whole larger than the sum of its parts.

Algorithm 2 CERL Algorithm

Initialize portfolio $\mathcal{P}$ with q learners (Alg 1) - varying γ
Start allocation $\mathcal{A}$ uniformly, and set # roll-out H = 0
Initialize a population of k actors pop_{π}
Initialize an empty cyclic replay buffer $\mathcal{R}$
Define a Gaussian noise generator $O = \mathcal{N}(0, \sigma)$
Define a random number generator $r() \in [0, 1)$
for generation = 1, ∞ **do**
  for actor $\pi \in pop_{\pi}$ **do**
    fitness, R = Evaluate(π, R, noise=None)
  Rank the population based on fitness scores
  Select the first e actors $\pi \in pop_{\pi}$ as elites
  Select $(k - e)$ actors π from pop_{π}, to form Set S using tournament selection with replacement
  while $|S| < (k - e)$ **do**
    Use single-point crossover between a randomly sampled $\pi \in e$ and $\pi \in S$ and append to S
  for Actor $\pi \in$ Set S **do**
    if $r() < mut_{prob}$ **then**
      Mutate(θ^{π})
  for Learner $\mathcal{L} \in \mathcal{P}$ **do**
    for ii =1,$\mathcal{A}_i$ **do**
      $score$, R = Evaluate(L_{π},R, $noise = O$)
      $\mathcal{L}_v$ = α * score + (1 - α) * $\mathcal{L}_v$
      $\mathcal{L}_{count}$ += 1
  ups = # of environment steps taken this generation
  for ii = 1, ups **do**
    for Learner $\mathcal{L} \in \mathcal{P}$ **do**
      Sample a random minibatch of T transitions (s_i, a_i, r_i, s_{i+1}) from $\mathcal{R}$
      Update the critic via a Bellman update using the min of $\mathcal{L}_{\mathcal{Q}_j'}(s_{i+1}$ (see 2.1)
      Update L_{π} using the sampled policy gradient with noisy actions (see 2.1))
      Soft update target networks:
      $L_{\theta^{\pi'}} \Leftarrow \tau L_{\theta^{\pi}} + (1 - \tau) L_{\theta^{\pi'}}$ and
      $L_{\theta^{\mathcal{Q}'}} \Leftarrow \tau L_{\theta^{\mathcal{Q}}} + (1 - \tau) L_{\theta^{\mathcal{Q}'}}$
  Compute the UCB scores $\mathcal{U}$ using
  for Learner $\mathcal{L} \in \mathcal{P}$ **do**

$$\mathcal{U}_i = \mathcal{L}_v + c * \sqrt{\frac{\log_e H}{\mathcal{L}_{count}}}$$

  Normalize U to be within $[0, 1)$ and set $\mathcal{A}$ = []
  Sample from U to fill up $\mathcal{A}$
  if generation mod $\omega = 0$ **then**
    for Learner $\mathcal{L} \in \mathcal{P}$ **do**
      Copy L_{π} into the population: for weakest $\pi \in pop_{\pi} : \theta^{\pi} \Leftarrow L_{\theta^{\pi}}$

Algorithm 3 Function Evaluate

procedure EVALUATE(π, R, noise)
  $fitness = 0$
  Reset environment and get initial state s_0
  while env is not done **do**
    Select action $a_t = \pi(s_t|\theta^\pi) + noise_t$
    Execute action a_t and observe reward r_t and new state s_{t+1}
    Append transition (s_t, a_t, r_t, s_{t+1}) to R
    $fitness \leftarrow fitness + r_t$ and $s = s_{t+1}$
  Return $fitness$, R

Algorithm 4 Function Mutate

procedure MUTATE(θ^π)
  for Weight Matrix $\mathcal{M} \in \theta^\pi$ **do**
    for iteration = 1, $mut_{frac} * |\mathcal{M}|$ **do**
      Randomly sample indices i and j from $\mathcal{M}'s$ first and second axis, respectively
      if $r() < supermut_{prob}$ **then**
        $\mathcal{M}[i,j] = \mathcal{M}[i,j] * \mathcal{N}(0, 100 * mut_{strength})$
      else if $r() < reset_{prob}$ **then**
        $\mathcal{M}[i,j] = \mathcal{N}(0, 1)$
      else
        $\mathcal{M}[i,j] = \mathcal{M}[i,j] * \mathcal{N}(0, mut_{strength})$

A general flow of the CERL algorithm proceeds as follows: a population of actor networks is initialized with random weights. The population is then *evaluated* in an episode of interaction with the environment (*roll-out*). The fitness for each actor is computed as the cumulative sum of the rewards received in the roll-out. A *selection* operator selects a portion of the population for survival with probability commensurate with their relative fitness scores. The weights of the actors in the population are then probabilistically *perturbed* through mutation and crossover operators to create the next **generation** of actors. A select portion of actors with the highest relative fitness are shielded from the mutation step and are preserved as elites.

Portfolio: The procedure described so far is reminiscent of a standard EA. However, in addition to the population of actors, CERL initializes a collection of *learners* (henceforth referred to as a *portfolio*). Each learner is initialized with its own actor, critic and has an associated learning algorithm defined with its own distinct hyperparameters. In this paper, the variation across learners is realized through varying discount rates (γ). However, in general, this can be any other variation in the hyperparameters, including a difference in the learning algorithm itself. The variation in discount rate used in this work can be interpreted as each learner optimizing over a distinct time-horizon of the underlying MDP. Learners with lower discount rates optimize a "greedier" objective than the ones with larger discount rates (long-term optimizers). The greedier objective has the benefit of being highly learnable but is not guaranteed to be aligned with the true learning goal. On the other hand, the long-term objective is more aligned to the true learning goal but is not as learnable - suffering from high variance due to its returns being conditioned on a longer time horizon. Thus, the portfolio represents a diverse set of learners, each with its own strengths and weaknesses.

Adaptive Resource Allocation: CERL is initialized with a computation resource budget of b workers dedicated to running roll-outs for its learner portfolio (separate from the resources used to evaluate the evolutionary population of actors). Allocation $\mathcal{A}$ defines the allotment of this resource budget amongst the learners within the portfolio for each generation of learning. This is initialized uniformly - each learner gets an equal number of dedicated workers to run roll-outs using its actor as the behavioral policy. Each learner stores statistics about the number of cumulative roll-outs it has run y, and a value metric v, defined as the discounted sum of the cumulative returns received from its own roll-outs. v is updated after every roll-out as:

$$v' \Leftarrow \alpha * return + (1 - \alpha) * v$$

After each generation, an upper confidence bound (UCB) (Auer, 2002) score $\mathcal{U}$ is computed for each learner based on its node statistics using Equation 1. This formulation is commonly used in solving multi-bandit problems (Bubeck et al., 2012; Karnin et al., 2013). The UCB score is known to provide good trade-offs between exploitation and exploration and has been extensively used for reinforcement learning in the form of tree searches (Anthony et al., 2017; Silver et al., 2016) and algorithms selection (Laroche & Feraud, 2017).

$$\mathcal{U}_i = v_i^n + c * \sqrt{\frac{\log(\sum_{i=1}^{b} y_i)}{y_i}} \quad (1)$$

where, v^n is v normalized to be $\in (0, 1)$

The UCB scores are normalized to form a probability distribution, and allocation A is re-populated by iterative sampling from this distribution. The allocation describes the new allotment of resources (roll-out workers) amongst the learners for the next generation. The process can be seen as a meta-operation that adaptively distributes resources across the learners dynamically during the course of learning. The underlying UCB technique used to control this distribution ensures a systematic approach to balancing exploitation and exploration when allocating resources across learners.

Shared Experiences: The collective replay buffer is the principal mechanism that enables the sharing of information across the evolutionary population and amongst the learners

in the portfolio. In contrast to a standard EA which would extract the fitness metric from each of its roll-outs and disregard them immediately, or ensemble methods that treat different learners separately, CERL pools all experiences defined by the tuple *(current state, action, next state, reward)* in its collective replay buffer. This is done for every interaction, at every time-step, for every episode and for each of its actors (including the evolutionary population and each roll-out conducted by the portfolio of learners). All learners are then able to sample experiences from this collective buffer and use it to update its parameters repeatedly using gradient descent. This mechanism allows for increased information extraction from each individual experiences leading to improved sample efficiency.

Diverse Exploration: In contrast to most methods where a learner learns based on data that its behavioral policy generates, CERL enables its portfolio of learners to leverage the data generated by a diverse set of actors. This includes the actors within the neuroevolutionary population and the actors stemming from other learners in the portfolio. Since each learner optimizes over varying time-horizons of the same underlying MDP, the associated actors lead to diverse behavioral policies exploring different regions of the solution space while remaining aligned with the task at hand. Additionally, in contrast to the learners which explore in their *action* space, the neuroevolutionary population explores in its *parameter* (neural weights) space using the mutation operator. The two processes complement each other and collectively lead to an effective strategy that is able to better explore the policy space.

Portfolio $\rightarrow$ EA: Periodically, each learner network is copied into the evolutionary population of actors, a process referred to as *Lamarckian transfer*. The frequency of *Lamarckian transfer* controls the flow of information from the gradient-based learners in the portfolio to the gradient-free evolutionary population. This is the core mechanism that enables the evolutionary framework to directly leverage the information learned through gradient-based optimization. The evolutionary process also acts as an amplifier in the realization of adaptive resource allocation. Good learner policies are selected to survive and reproduce - extending their influence in the population over subsequent generations. These policies and their descendants contribute increasingly more data experiences into the collective replay buffer and influence the learning of the all portfolio learners. Bad learner policies, on the other hand, are rejected to minimize their influence. Finally, crossover serves to exploit any decomposability in the policy space and combines good "sub-components of the policies" present in the diverse evolutionary population.

Algorithm 2, 3 and 4 provide a detailed pseudo-code of the CERL algorithm using a portfolio of TD3 learners. The choice of hyperparameters is explained in the Appendix. Additionally, our source code [1] is available online.

6. Results

Domain: CERL is evaluated on 5 continuous control tasks on Mujoco (Todorov et al., 2012). These benchmarks are used widely in the field (Khadka & Tumer, 2018; Such et al., 2017; Schulman et al., 2017) and are hosted on OpenAI gym (Brockman et al., 2016).

Compared Baselines: For each benchmark, we compare the performance of CERL with its composite learners ran in isolation. While not constrained to this arrangement, CERL here is built using a combination of a neuroevolutionary algorithm (EA) and 4 policy gradient based learners. We use TD3 (Fujimoto et al., 2018b) as our policy gradient learner as it is the current state-of-the-art off-policy algorithm for these benchmarks. The 4 TD3 learners are identical with each other apart from their discount rates which are 0.9, 0.99, 0.997, and 0.9995. These were not tuned for performance.

We also ran CERL with a single learner - picking the best TD3 learner for each task. This is equivalent to ERL (Khadka & Tumer, 2018) with the exception of the resource manager. However, the resource manager does not have any functional effect when there is only one learner.

Methodology for Reported Metrics: For TD3, the actor network was periodically tested on 10 task instances without any exploratory noise. The average score was logged as its performance. During each training generation, the actor network with the highest fitness was selected as the champion. The champion was then tested on 10 task instances, and the average score was logged. This protocol shielded the reported metrics from any bias of the population size. We conduct 5 statistically independent runs with random seeds from $\{2018, 2022\}$ and report the average with error bars showing a 95% confidence interval.

The "Steps" Metric: All scores reported are compared against the number of environment steps. A step is defined as an agent taking an action and receiving a reward back from the environment. To make the comparisons fair across single-agent and population-based algorithms, all steps taken by all actors in the population, and by all learners in the portfolio are counted cumulatively.

Hyperparameter Selection: The hyperparameters used for CERL were **not** tuned to generate the results, unless specifically stated. The parameters used for the TD3 learners were simply inherited from (Fujimoto et al., 2018b), while the evolutionary parameters were inherited from (Khadka & Tumer, 2018). The computational budget of b workers

[1] github.com/intelai/cerl

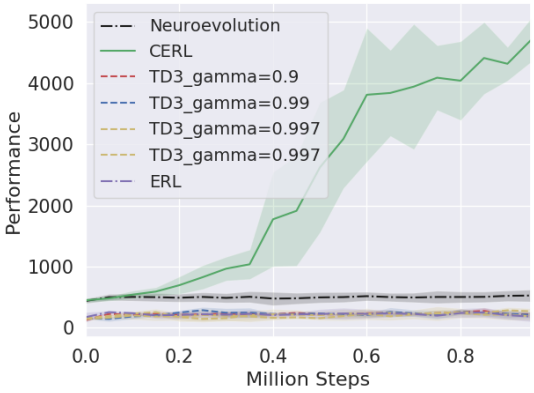


Figure 3. Comparative Results for CERL tested against its composite learners in the Humanoid benchmark.

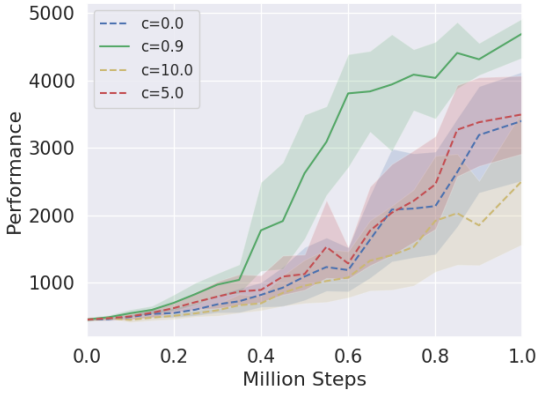


Figure 4. Sensitivity analysis for resource-manager exploration (c) in the Humanoid benchmark

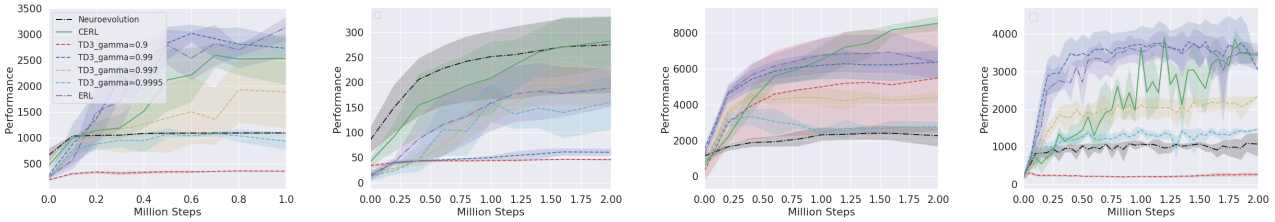
was set to 10 to match the evolutionary population size. The UCB exploration coefficient was set to 0.9 which numerically makes the relative weight of exploration and exploitation terms in Equation 1 close to equilibrium at the start.

Humanoid: Figure 3 shows the comparative performance of CERL, alongside its composite learners. CERL significantly outperforms neuroevolution, as well as all versions of TD3 with varying discount rates. The TD3 learners fail to learn at all, which is consistent with reports in previous literature (Haarnoja et al., 2018). On the other hand, neuroevolution alone was shown to solve Humanoid, but required 62.5 millions roll-outs (Lehman et al., 2018). CERL is able to achieve a score of 4702.0 ± 356.5 within 1 million environment steps (approximately 4000 roll-outs). Considering that CERL only uses a combination of these learners, this is a significant result. Each learner in isolation fails to learn on the task entirely, while the same learners when incorporated under the CERL framework, are able to solve it jointly. This is because none of the learners are able to succeed when burdened with both exploring the solution space to generate an expansive set of data, and exploiting it aptly. However, when the learners collectively explore diverse regions of the solution space, and collectively exploit these experiences, they succeed. The single-learner ERL also fails to learn this task. Since the key difference between ERL and CERL is the use of multiple learners, this demonstrates that the performance gains of CERL come primarily from this collaboration.

Resource-manager's Sensitivity to Exploration: Figure 4 shows the comparative performance for CERL tested with varying c (exploration coefficient in Equation 1) for the Humanoid benchmark. CERL with $c = 0.9$ performs the best as it provides a good balance of exploration and exploitation for the resource-manager. However, CERL with $c = 0.0$ and $c = 5.0$ both are also able to learn well the benchmark, but are less sample-efficient. An important point to note is that $c = 0.0$ does not lead to the complete lack of exploration. As all learners start with random weights, the returns are close to random at the beginning of learning and serves to bootstrap exploration. On the other hand, a c of 10 does lead to extremely high exploration. As expected, this prolonged exploration leads to even lower sample efficiency. This highlights the role that the resource-manager plays in dynamically redistributing resource and finding the balance between exploration and exploitation.

Additional Mujoco Experiments: Figure 5 shows the comparative performance of CERL, alongside its composite learners in 4 additional environments simulated using Mujoco. Unlike the 3D humanoid benchmark, these domains are 2D, have considerably smaller state and action spaces, and are relatively simpler to solve. One of the four TD3 learners: TD3 with a discount rate of 0.99 (TD3-0.99) is able to solve 3 out of the 4 benchmarks, with the exception of Swimmer. CERL is also able to solve these benchmarks but is less sample-efficient that TD3-0.99. However, on the Swimmer benchmark, while all of the TD3 learners fail to solve the task, CERL successfully solves it similar to neuroevolution. This emphasizes the key strength of CERL: the ability to inherit the best of its composite approaches.

While TD3-0.99 is more sample-efficient in 3 out of the 4 benchmarks, CERL is more sample-efficient than all the other TD3-based learners. This suggests that 0.99 is an ideal discount rate for these tasks. Any deviation from this value leads to considerable loss in performance for TD3. In other words, this is a sensitive hyperparameter that has to be rigorously tuned. CERL achieves this functionality online through its resource-manager, which adaptively re-distributes computational resources across the learners. While this invariably leads to the use of more samples when compared to an ideal hyperparameter that is known *a priori*, CERL is able to identify and exploit the best hyperparam-



(a) Hopper (b) Swimmer (c) HalfCheetah (d) Walker2D

Figure 5. Comparative results of CERL with 4 learners (TD3 with discount rates of 0.9, 0.99, 0.997 and 0.9995) against the learners in isolation, and neuroevolution.

Table 1. Average cumulative resource-allocation rate for CERL across benchmarks. (error intervals omitted as all were < 0.04)

Task	L1	L2	L3	L4
Humanoid	0.24	0.35	0.20	0.20
Hopper	0.14	0.27	0.32	0.27
Swimmer	0.17	0.20	0.36	0.27
HalfCheetah	0.29	0.32	0.24	0.15
Walker	0.14	0.28	0.33	0.25

eters online via joint exploration. Additionally, as demonstrated in the cases of Swimmer and Humanoid (Figure 3), this exploration itself is critical to successful learning as there does not exist one hyperparameter that can solve the task all by itself. Overall, CERL enables an arguably simpler alternative to network design compared to complex hyperparameter tuning methodologies.

Allocation: Table 1 reports the final cumulative resource-allocation rate across the four learners for CERL in the five Mujoco benchmarks tested. L1, L2, L3 and L4 correspond to learners with $\gamma = 0.9, 0.99, 0.997$ and 0.9995, respectively. L2 seems to be the learner that is generally preferred across most tasks. This is not surprising as this value for γ is the hyperparameter used in (Fujimoto et al., 2018b) after tuning. However, in the Swimmer benchmark, this choice of hyperparameter is not ideal. Learners with higher γ perform significantly better on the task (Figure 5). CERL is able to identify this online and allocates more resources to L3 and L4 with higher γ. This flexibility for online algorithm selection, in combination with its evolutionary population, enables CERL to solve the Swimmer benchmark effectively.

7. Discussion

We presented CERL, a scalable platform that allows gradient-based learners to jointly explore and exploit solutions in a gradient-free evolutionary framework. Experiments in continuous control demonstrate that CERL's emergent learner can outperform its composite learners while remaining overall sample-efficient compared to traditional approaches.

Strengths: CERL is generally insensitive to its hyperparameters and to those of the individual learners. The Humanoid and Swimmer problems are examples where state-of-the-art algorithms show high sensitivity to their hyperparameters while CERL required no hyperparameter tuning. Significantly, the Humanoid problem demonstrates that CERL is able to find effective solutions using participating learners that fail completely on their own. This makes CERL a simpler design alternative to complex hyperparameter tuning and one that seems to generalize well across multiple tasks.

A practical consideration for CERL is the parallel operation of gradient-based and gradient-free methods. The former, involving backpropagation, is typically suited for GPUs. The latter, involving forward-propagation, is typically suited for CPUs and is highly scalable, leading to impressive wall-clock performances (Salimans et al., 2017; Such et al., 2017). By leveraging both modes simultaneously, CERL provides a principled way to parallelize learning and to cater one's learning algorithm to the available hardware.

Limitations: CERL can be less sample-efficient for simple tasks where the ideal hyperparameters are known *a priori*. This is apparent in the case of Walker2d (Fig 5) and can be attributed to the exploration involved in selecting learners. However, CERL does eventually match the performance shown by the learner with the known ideal hyperparameter. This weakness of CERL is contingent on the ability to derive the ideal parameters for a learner - a process which by itself generally consumes significant resources that are often not reported in literature.

Future Work: Here, we explored homogeneous learners optimizing over varying time-horizons of a task. Future work will extend this to learners that are different algorithms themselves. Incorporating stochastic actors from SAC (Haarnoja et al., 2018) with the deterministic TD3 actors is an exciting area. Another promising line of work would be to incorporate learning within the resource manager to augment the current UCB formulation.

References

Ackley, D. and Littman, M. Interactions between learning and evolution. *Artificial life II*, 10:487–509, 1991.

Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Abbeel, O. P., and Zaremba, W. Hindsight experience replay. In *Advances in Neural Information Processing Systems*, pp. 5048–5058, 2017.

Anthony, T., Tian, Z., and Barber, D. Thinking fast and slow with deep learning and tree search. In *Advances in Neural Information Processing Systems*, pp. 5360–5370, 2017.

Auer, P. Using confidence bounds for exploitation-exploration trade-offs. *Journal of Machine Learning Research*, 3(Nov):397–422, 2002.

Bellemare, M., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., and Munos, R. Unifying count-based exploration and intrinsic motivation. In *Advances in Neural Information Processing Systems*, pp. 1471–1479, 2016.

Bhatnagar, S., Precup, D., Silver, D., Sutton, R. S., Maei, H. R., and Szepesvári, C. Convergent temporal-difference learning with arbitrary smooth function approximation. In *Advances in Neural Information Processing Systems*, pp. 1204–1212, 2009.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Bubeck, S., Cesa-Bianchi, N., et al. Regret analysis of stochastic and nonstochastic multi-armed bandit problems. *Foundations and Trends® in Machine Learning*, 5 (1):1–122, 2012.

Cauwet, M.-L., Liu, J., and Teytaud, O. Algorithm portfolios for noisy optimization: Compare solvers early. In *International Conference on Learning and Intelligent Optimization*, pp. 1–15. Springer, 2014.

Colas, C., Sigaud, O., and Oudeyer, P.-Y. Gep-pg: Decoupling exploration and exploitation in deep reinforcement learning algorithms. *arXiv preprint arXiv:1802.05054*, 2018.

Drugan, M. M. Reinforcement learning versus evolutionary computation: A survey on hybrid algorithms. *Swarm and Evolutionary Computation*, 2018.

Duan, Y., Chen, X., Houthooft, R., Schulman, J., and Abbeel, P. Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*, pp. 1329–1338, 2016.

Floreano, D., Dürr, P., and Mattiussi, C. Neuroevolution: from architectures to learning. *Evolutionary Intelligence*, 1(1):47–62, 2008.

Fogel, D. B. *Evolutionary computation: toward a new philosophy of machine intelligence*, volume 1. John Wiley & Sons, 2006.

Fortunato, M., Azar, M. G., Piot, B., Menick, J., Osband, I., Graves, A., Mnih, V., Munos, R., Hassabis, D., Pietquin, O., et al. Noisy networks for exploration. *arXiv preprint arXiv:1706.10295*, 2017.

Fujimoto, S., van Hoof, H., and Meger, D. Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477*, 2018a.

Fujimoto, S., van Hoof, H., and Meger, D. Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477*, 2018b.

Gagliolo, M. and Schmidhuber, J. Learning dynamic algorithm portfolios. *Annals of Mathematics and Artificial Intelligence*, 47(3-4):295–328, 2006.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., and Meger, D. Deep reinforcement learning that matters. *arXiv preprint arXiv:1709.06560*, 2017.

Houthooft, R., Chen, X., Duan, Y., Schulman, J., De Turck, F., and Abbeel, P. Vime: Variational information maximizing exploration. In *Advances in Neural Information Processing Systems*, pp. 1109–1117, 2016.

Islam, R., Henderson, P., Gomrokchi, M., and Precup, D. Reproducibility of benchmarked deep reinforcement learning tasks for continuous control. *arXiv preprint arXiv:1708.04133*, 2017.

Jaderberg, M., Dalibard, V., Osindero, S., Czarnecki, W. M., Donahue, J., Razavi, A., Vinyals, O., Green, T., Dunning, I., Simonyan, K., et al. Population based training of neural networks. *arXiv preprint arXiv:1711.09846*, 2017.

Karnin, Z., Koren, T., and Somekh, O. Almost optimal exploration in multi-armed bandits. In *International Conference on Machine Learning*, pp. 1238–1246, 2013.

Khadka, S. and Tumer, K. Evolution-guided policy gradient in reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 1196–1208, 2018.

Lagoudakis, M. G. and Littman, M. L. Algorithm selection using reinforcement learning. In *ICML*, pp. 511–518. Citeseer, 2000.

Laroche, R. and Feraud, R. Reinforcement learning algorithm selection. *arXiv preprint arXiv:1701.08810*, 2017.

Lehman, J., Chen, J., Clune, J., and Stanley, K. O. Es is more than just a traditional finite-difference approximator. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 450–457. ACM, 2018.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Lüders, B., Schläger, M., Korach, A., and Risi, S. Continual and one-shot learning through neural networks with dynamic external memory. In *European Conference on the Applications of Evolutionary Computation*, pp. 886–901. Springer, 2017.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529, 2015.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pp. 1928–1937, 2016.

Ostrovski, G., Bellemare, M. G., Oord, A. v. d., and Munos, R. Count-based exploration with neural density models. *arXiv preprint arXiv:1703.01310*, 2017.

Pathak, D., Agrawal, P., Efros, A. A., and Darrell, T. Curiosity-driven exploration by self-supervised prediction. In *International Conference on Machine Learning (ICML)*, volume 2017, 2017.

Plappert, M., Houthooft, R., Dhariwal, P., Sidor, S., Chen, R. Y., Chen, X., Asfour, T., Abbeel, P., and Andrychowicz, M. Parameter space noise for exploration. *arXiv preprint arXiv:1706.01905*, 2017.

Rice, J. R. The algorithm selection problem. In *Advances in computers*, volume 15, pp. 65–118. Elsevier, 1976.

Risi, S. and Togelius, J. Neuroevolution in games: State of the art and open challenges. *IEEE Transactions on Computational Intelligence and AI in Games*, 9(1):25–41, 2017.

Salimans, T., Ho, J., Chen, X., and Sutskever, I. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*, 2017.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. Trust region policy optimization. In *International Conference on Machine Learning*, pp. 1889–1897, 2015.

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

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

Smith-Miles, K. A. Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Computing Surveys (CSUR)*, 41(1):6, 2009.

Spears, W. M., De Jong, K. A., Bäck, T., Fogel, D. B., and De Garis, H. An overview of evolutionary computation. In *European Conference on Machine Learning*, pp. 442–459. Springer, 1993.

Stanley, K. O. and Miikkulainen, R. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.

Such, F. P., Madhavan, V., Conti, E., Lehman, J., Stanley, K. O., and Clune, J. Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. *arXiv preprint arXiv:1712.06567*, 2017.

Sutton, R. S. and Barto, A. G. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

Tang, H., Houthooft, R., Foote, D., Stooke, A., Chen, O. X., Duan, Y., Schulman, J., DeTurck, F., and Abbeel, P. # exploration: A study of count-based exploration for deep reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 2750–2759, 2017.

Todorov, E., Erez, T., and Tassa, Y. Mujoco: A physics engine for model-based control. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pp. 5026–5033. IEEE, 2012.

Turney, P., Whitley, D., and Anderson, R. W. Evolution, learning, and instinct: 100 years of the baldwin effect. *Evolutionary Computation*, 4(3):iv–viii, 1996.