Efficient and Stable Off-policy Training via Behavior-aware Evolutionary Learning

Anonymous Author(s)
Affiliation
Address
email

Abstract: Applying reinforcement learning (RL) algorithms to real-world continuos control problems faces many challenges in terms of sample efficiency, stability and exploration. Off-policy RL algorithms show great sample efficiency but can be unstable to train and require effective exploration techniques for sparse reward environments. A simple yet effective approach to address these challenges is to train a population of policies and ensemble them in certain ways. In this work, a novel population based evolutionary training framework inspired by evolution strategies (ES) called Behavior-aware Evolutionary Learning (BEL) is proposed. The main idea is to train a population of behaviorally diverse policies in parallel and conduct selection with simple linear recombination. BEL consists of two mechanisms called behavior-regularized perturbation (BRP) and behavior-targeted training (BTT) to accomplish stable and fine control of the population behavior divergence. Experimental studies showed that BEL not only has superior sample efficiency and stability compared to existing methods, but can also produce diverse agents in sparse reward environments. Due to the parallel implementation, BEL also exhibits relatively good computation efficiency, making it a practical and competitive method to train policies for real-world robots.

Keywords: Continuos control, Reinforcement learning, Evolution strategies

1 Introduction

Recent advances in reinforcement learning (RL) have proven that off-policy deep reinforcement learning (DRL) algorithms possess great potential in solving continuos control problems, especially in terms of sample efficiency [1] [2] [3]. However, off-policy algorithms are also known to be unstable or brittle [4] [5], which to some extent hinders the application of these algorithms to real-world robots. On the other hand, environments with sparse rewards which are common in real-world scenarios also present challenges in terms of exploration.

Along with DRL algorithms, another line of direct policy search method called evolutionary algorithms (EA) also showed significant success as a result of improved computation efficiency of modern hardwares and clever implementations [6] [7] [8]. Different from DRL algorithms which exploit the sequential structure of the Markov Decision Process (MDP), EA algorithms treat policy search as a black-box optimization problem and utilize a population of randomly perturbed policies to search for better policies. While being less sample efficient, EA methods tend to enjoy properties like improved stability, efficient parallelization and better diversity due to the utilization of the EA population and EA operations.

Naturally, combining those two paradigms to get the best of both worlds has attracted lots of efforts over the years [9] [10] [11] [12] [13]. The motivation behind these works is to inject the gradient-trained DRL agents into the population and drive the population with policy gradient signals while enjoying the benefits of the EA population. Another perspective is that EA mutation can serve

Submitted to the 6th Conference on Robot Learning (CoRL 2022). Do not distribute.

as parameter space noise and improve the exploration ability of RL agents [14]. Such kind of combination turned out to be very successful, the resulting hybrid algorithms can beat both of their EA and DRL components.

However, we identified two important problems remain unsolved by previous methods. The first problem is: how to randomly perturb the policy in a meaningful and controlled manner? A too small perturbation would result in no significant changes, while a too large one can lead to divergent training. Inspired by previous works [15] [12], we propose to solve this problem by Behavior-Regularized Perturbation (BRP) which can randomly perturb a policy network within a specified behavior divergence range in an online fashion. The second problem is: although perturbed networks result in different policies, once policies undergo the same RL training process and sample from the sample replay buffer, they may end up being similar and reduce the overall diversity. Ideally, we would like to have a population that is not only diverse after random perturbation, but also diverse after RL training. To accomplish this goal, Behavior-Targeted Training (BTT), is proposed to assign a randomly sampled target behavior divergence and inject it into the actor training process. The final proposed training framework is called Behavior-aware Evolutionary Learning (BEL). In BEL, training is conducted in a generational fashion which closely resembles a traditional evolution strategy (ES). In each generation, policies are first randomly generated by applying BRP to a central mean policy. Then, those offspring policies are trained in parallel with BTT. Finally, all trained policies undergo a weighted linear combination [16] and form the new mean policy for the next generation.

2 Related works

Model-free off-policy RL algorithms are a class of sample efficient algorithms for continuos control tasks with relatively high dimensional action spaces [1] [17]. Built upon the actor-critic (AC) paradigm, where a pair of actor network and critic network are trained simultaneously, the Twin Delayed Deep Deterministic (TD3) algorithm [2] and the Soft Actor-Critic (SAC) algorithm [3] showed great success, and quickly become the go-to algorithms for sample efficient RL training.

Evolutionary algorithms have also gained attention due to the fact that they prove to be competitive alternatives to MDP-based methods [6] [7]. In [6], authors developed a simplified natural evolution strategies (NES) [18]. The resulting OpenAI ES offers massive scalability while matching the performance of MDP-based methods. In [7], it was shown that genetic algorithm (GA) was able to evolve networks with four million parameters and achieved competitive performance compared to gradient-based methods.

Combining evolutionary methods and policy gradient-based methods in order to benefit from the best of two worlds soon attracted much attention when [9] first proposed to evolve a population of agents with GA and periodically inject gradient information into the population. Their resulting algorithm ERL outperformed both GA and Deep Deterministic Policy Gradient (DDPG). In [11], authors managed to use a variant of ES called cross entropy method (CEM) to evolve the population half of which was composed of EA agents, and the other half was composed of TD3 agents. Their hybrid algorithm CEM-RL turned out to be very competitive and served as a strong baseline for derivative works. Later, [12] pointed out that traditional crossover and mutation operators widely used in GA can be detrimental in the sense that they could destroy learned behaviors. As a remedy, [12] proposed to conduct crossover with network distillation and mutation with SM-G-SUM [15] which proved to be able to keep learned network behaviors.

3 Background

3.1 Evolution Strategies (ES)

Evolution strategies (ES) belongs to the gradient-free black-box optimization algorithm family. It was first proposed by Rechenberg [19], and later developed by Schwefel [20]. Mimicking the natural evolution process, ES randomly generates a population of solution vectors (usually with a Gaussian

distribution) whose fitness value will be evaluated in a problem-specific manner (for example episodic reward). In its canonical form, ES can be classified into two major versions: the $(\mu, \lambda) - ES$, where μ parents of the next generation are selected from the current λ offsprings, and the $(\mu + \lambda) - ES$, where the selection pool contains both the current parents and offsprings. The selection operation is usually a simple weighted linear recombination of the population vectors according to their fitness ranks. In this work, we adopt the simplest $(1, \lambda) - ES$ scheme, and model the population with a uniform distribution in terms of behavior divergence.

3.2 Twin Delayed Deep Deterministic Policy Gradient (TD3)

As an RL algorithm, TD3 is built upon the Markov Decision Process (MDP) which is described by $< S, A, P, R, \gamma >$. In this formulation, S is the state space, A is the action space, P is the transition function, R is the reward function and γ is a discount factor [4]. The goal is to learn an optimal policy function π to maximize the expected return $J(\theta) = \mathbb{E}_{s \sim p_\pi, a \sim \pi}[R_0]$. TD3 solves this problem by adopting the actor-critic deterministic policy gradient [21] [1] [2], where a Q-function Q_ϕ is learned through the Bellman equation:

$$Q^\pi(s, a) = r + \gamma \mathbb{E}_{s', a'}\left[Q^\pi(s', a')\right], \quad a' \sim \pi(s') \tag{1}$$

And then the policy function π_θ is optimized by the deterministic policy gradient:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s \sim p_\pi}\left[\nabla_a Q^\pi(s, a)|_{a=\pi(s)} \nabla_\theta \pi_\theta(s)\right] \tag{2}$$

For implementation, both Q_ϕ and π_θ are optimized with Monte-Carlo estimation with the help of a replay buffer D, the loss function of Q_ϕ and π_θ are defined as follows:

$$\mathcal{L}_{Q_\phi}^{TD3} = \mathop{\mathrm{E}}_{(s,a,r,s') \sim \mathcal{D}}\left[\left(Q_\phi(s, a) - \left(r + \gamma \max_{a'} Q_\phi(s', a')\right)\right)^2\right] \tag{3}$$

$$\mathcal{L}_{\pi_\theta}^{TD3} = -\mathop{\mathbb{E}}_{s \sim \mathcal{D}}\left[\mathop{\mathbb{E}}_{a \sim \pi_\theta} Q_\phi(s, a)\right] \tag{4}$$

In TD3, three tricks are used to make the above learning process more stable and alleviate the overestimation bias. The first trick is to learn two Q functions and uses the smaller Q-value to form the target Q in eq. (3). The second trick is to delay the target networks updates with regard to Q network updates. The third trick is to add noise to target actions to smooth out Q along changes in action [2].

4 Behavior-aware Evolutionary Learning (BEL)

4.1 BEL framework

The overall structure of BEL is outlined in fig. 1. In BEL, we maintain a center actor as the population center, and λ actors as offsprings. In each generation, first, all offspring actors will be initialized around the center actor with Behavior-Regularized Perturbation smoothing (BRP), which will be introduced in detail in section 4.2. Then each offspring actor will undergo Behavior-Targeted Training (BTT) as will be described in section 4.3. After these two processes, all offspring actors will interact with the environment and save their experiences to the replay buffer. Finally, the population selection is conducted with a weighted linear recombination of network parameters according to episodic rewards of the trained offspring actors to form the center actor for next generation. This process is repeated until termination criterion is met.

4.2 Behavior-Regularized Perturbation (BRP)

Similar to the SM-G-SUM mutation operator used in [12] and [15], BRP relies on the calculation of the so-called parameter sensitivity with regard to network outputs. Given an actor network π_θ and a batch of transitions i , BRP approximately measures how the overall output will vary with regard to

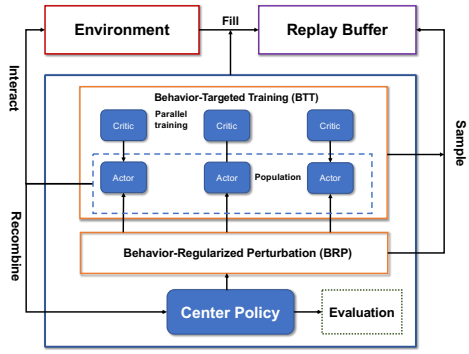


Figure 1: An illustration of the BEL framework.

small changes of the neural network's weights θ through the aggregation of backward gradients of each output node k on data batch i. For each parameter in π_θ, its sensitivity $sens$ is calculated by:

$$sens = \sqrt{\sum_k \left(\sum_i \left| \nabla_\theta \pi_\theta \left(s_i \right)_k \right| \right)^2} \tag{5}$$

A large value of $sens$ indicates that the corresponding parameter will lead to a large change of the action output and vice versa. Denote the overall sensitivity for all parameters as $Sens$, it is then used as the coefficient of the below linear transformation:

$$Vec(\tilde{\pi}) = Vec(\pi_\theta) + \frac{\delta}{Sens} \tag{6}$$

In eq. (6), $Vec(\pi_\theta)$ means network parameters represented as a one-dimensional vector. $\tilde{\pi_\theta}$ is the perturbed policy network and δ is a random vector that determines the perturbation magnitude and direction.

Unlike previous methods [22] [12] where δ is randomly sampled from a constant-scaled Gaussian distribution, BRP instead tries to adaptively search for a proper δ within a certain magnitude that can bound the behavior divergence of the perturbed network. This idea is similar to the parameter noise adaption method in [14]. BRP adopts the Euclidean norm as the behavior divergence measure: $d(\pi_\theta(s), \tilde{\pi}(s)) = \sqrt{\frac{1}{N} \sum_{k=1}^{N} \mathbb{E}_s \left[\left(\pi_\theta(s) - \tilde{\pi}(s) \right)^2 \right]}$. Given a behavior divergence Δ_{max}^{BRP} upper bound, to generate one perturbed network $\tilde{\pi}$, BRP first randomly samples a divergence Δ_i, and then conduct a simple iterative linear search to find a proper δ, detailed procedure is summarized in algorithm 1. The final output is a set of λ randomly perturbed policy networks following a uniform distribution in the behavior divergence space. Note that unlike previous implementations which only approximately calculated $\mathbf{S}$, our implementation precisely calculated $\mathbf{S}$ with the help of Pytorch hooks, and generating five perturbed networks can be done within 0.2 seconds.

4.3 Behavior-Targeted Training (BTT)

BRP generates policy networks through random local perturbation, BTT on the other hand generates trained policies that are within a behavior divergence range to the center policy. Consider one offspring policy $\tilde{\pi_i}$ generated by BRP, we would like its behavior divergence after training $\Delta_i^{trained}$ to lie in the range defined by an upper bound: $\Delta_i^{trained} \in \left[0, \Delta_{max}^{BTT}\right]$.

To achieve this goal, we gained inspiration from imitation learning. In each generation of policy gradient training, the actor network aims to optimize two objectives. The first objective is the traditional RL objective as in eq. (4). For the second objective, consider a batch of states s sampled from the replay buffer, the actions of both of the center policy π_θ and the offspring policy $\tilde{\pi_i}$ in those states are calculated as $a_\theta = \pi_\theta(s)$ and $a_i = \tilde{\pi_i}(s)$. Then as depicted in fig. 2(a), we construct a

Algorithm 1 Behavior-Regularized Perturbation

Input: Population center policy π_θ, population size λ, error bound ϵ, initial magnitude scalar δ_{init}, a batch of states s, and $\beta \in [0, 1]$
Calculate $Sens$ for π_θ according to eq. (5)
for $i = 1$ **to** λ **do**
  Sample a target divergence $\Delta_i \sim U_{[0,\Delta_{max}^{BRP}]}$, sample a random direction from $\delta \sim N(0, 1)$
  Get initial perturbed network $\tilde{\pi}_i$ with $Sens$ according to eq. (6)
  while $|d(\pi_\theta(s), \tilde{\pi}_i(s)) - \Delta_i| > \epsilon$ **do**
    if $d(\pi_\theta(s), \tilde{\pi}_i(s)) < \Delta_i$ **then**
      $\delta = \frac{\delta}{\beta}$
    else
      $\delta = \delta * \beta$
    end if
    Get perturbed network $\tilde{\pi}_i$ with $Sens$ and δ
  end while
end for
Output: Perturbed policies $\{\tilde{\pi}_i | i = 1, ..., \lambda\}$

"behavior potential well" with a one dimensional Gaussian distribution to force the negative log-likelihood of the Euclidean distance between a_θ and a_i to stay close to the bottom of the Gaussian whose mean is defined by a sampled and fixed Δ_i^{target}, standard deviation is defined by a predefined σ_{BTT}. This process results in the following training objective for BTT:

$$L_{\tilde{\pi}_i}^{BTT} = L_{\tilde{\pi}_i}^{TD3} - \alpha \ln \left[\frac{1}{\sigma_{BTT}\sqrt{2\pi}} e^{-\frac{\left[d(\pi_\theta(s), \tilde{\pi}(s)) - \Delta_i^{target}\right]^2}{2\sigma_{BTT}^2}} \right] \tag{7}$$

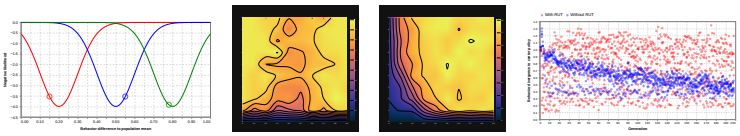
Following eq. (7), the policy network will try to simultaneously follow the policy gradient and stay inside the behavior potential well to roughly keep a Δ_i^{target} divergence to the center policy. α is a hyper-parameter balancing the two objectives. To determine Δ_i^{target}, we simply sample from a uniform distribution as BRP: $\Delta_i^{target} \sim U_{[0,\Delta_{max}^{BTT}]}$. As σ_{BTT} directly controls the steepness of the Gaussian distribution, a larger σ_{BTT} means less restriction over the policy's divergence and vice versa.

5 Experiments

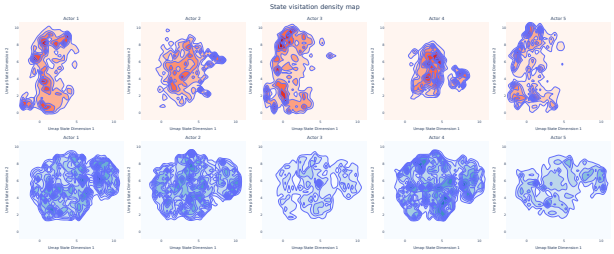
5.1 Exploratory studies

As previous studies have shown [11] [14], perturbing Tanh-activated networks is easier than perturbing ReLU-activated networks. As perturbing networks with BRP is straight forward and network architecture agnostic, we conducted a comparative study to see how those two-types of networks respond to BRP. To be specific, we randomly sampled directions and record behavior divergence changes along those directions. As in fig. 3(a) fig. 3(b), where the x-axis is the percentage of the positive sign in one direction, the y-axis is the magnitude along that direction and the color-scale measures the behavior divergence (the brighter the larger), it is clear that randomly perturbing Tanh-activated networks has a larger chance of inducing significant behavior changes, which explains why Tanh-activated networks are generally favored in perturbation-based methods.

To verify that actors trained by BTT are uniformly distributed as Δ_{BTT}^{target} is sampled from a uniform distribution, we trained two BEL instances with $\alpha = 0.0$ (without BTT) and $\alpha = 1.0$ (with BTT). From 2(d), it is obvious that without BTT, the trained policy are quite concentrated. And when BTT is applied, the behavior divergences of the population constantly follows the uniform distribution. To further verify that BTT can lead to diverse behaviors, we plotted the state visitation map on the DelayedHalfCheetah-v3 environment to visualize how the offspring policies explore different states. As is shown in 2(e), one generation of BEL's BTT-trained population visits very different states while naively-trained population without BTT doesn't show the same level of diversity.

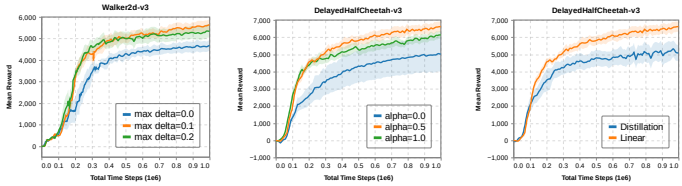


(a) (b) (c) (d)



(e)

Figure 2: (a) BTT illustration. The circles represent actors. Due to the log-likelihood training objective, actors will try to adapt their behaviors around their Gaussian centers. (b) BRP for a Tanh-activated network. (c) BRP for a ReLU-activated network. (d) Trained offspring behavior divergence illustration. (e) State visitation density map of five actors trained with BTT (first row) versus without BTT (second row) after one generation.



(a) (b) (c)

Figure 3: (a) BRP ablation. max delta=0 corresponds to no BRP (b) BTT ablation. $alpha = 0$ corresponds to no BTT, larger $alpha$ means more constrained behavior. (c) Recombination ablation.

5.2 Ablative studies

To verify the effectiveness of BRP, we tested different Δ_{max}^{BRP} settings on the Walker2d-v3 environment. From fig. 3(a), it is noticeable that BRP not only significantly speeds up the learning process, but also helps avoiding local optimums. As a matter of fact, we find that when BRP is applied, critic networks tend to constantly induce larger training loss throughout training. This phenomenon indicates that BRP indeed brings another level of behavior uncertainty, which forces critics to make better prediction.

An ablation study on the DelayedHalfCheetah-v3 environment is conducted to show that BTT is indeed helpful for exploration .DelayedHalfCheetah-v3 is a modified HalfCheetah-v3 environment where the reward is manually delayed for 20 time steps, making it a difficult sparse reward environment. The proportion of the log-likelihood objective is tuned with α. We can observe from fig. 3(b) that when α is set to zero, which means no BTT in the training objective, BEL can't effectively explore. However, on the other hand, when α is too large, actors may also lose performance since their behaviors are over constrained.

As [12] [15] pointed out, many operators in EA are designed for black-box optimization, and can be potentially harmful for neural networks. An experiment comparing weighted linear recombination and distillation-based recombination was designed on the DelayedHalfCheetah-V3 environment, where all parts of BEL are kept the same except for the recombination phase. In this phase, offspring actors are ranked according to their latest episodic rewards, and then, λ offspring actors are treated as

Table 1: Numerical results for final best mean reward of different algorithms on selected tasks

Tasks\Algorithms	Statistics	BEL (Ours)	SAC	TD3	CEM-RL	PDERL
HalfCheetah-v3	Mean	**12725.39**	10482.39	10408.62	10636.94	6917.24
	Std	**202.89**	1253.81	1093.64	2131.36	444.95
	Median	**12751.99**	11058.84	10810.48	11323.23	7026.15
	WallClock	4.50h	5.34h	**2.31h**	5.52h	3.20h
Ant-v3	Mean	**6082.41**	5208.65	5090.81	3455.95	1609.40
	Std	**166.28**	282.64	651.13	1359.87	542.42
	Median	**6147.19**	5259.90	5385.89	3487.73	1582.24
	WallClock	5.81h	8.03h	**3.09h**	6.47h	3.61h
Walker2d-v3	Mean	**5723.30**	4637.03	3855.60	4173.30	1588.51
	Std	**498.38**	414.19	760.91	1153.97	641.26
	Median	**6087.36**	4682.27	4138.82	4358.34	1253.77
	WallClock	4.14h	7.54h	**2.54h**	5.91h	3.42h
Hopper-v3	Mean	**3717.14**	3543.35	3426.26	3597.87	1293.66
	Std	**101.08**	103.29	192.31	495.28	356.54
	Median	**3740.41**	3580.16	3333.04	3749.87	1160.93
	WallClock	4.29h	7.94h	**2.30h**	5.82h	3.35h
Humanoid-v3	Mean	5337.20	**5617.94**	5319.09	215.79	815.96
	Std	113.53	133.93	114.38	**0.44**	90.86
	Median	5364.52	**5588.50**	5333.41	215.76	821.11
	WallClock	7.74h	8.48h	**4.51h**	9.25h	4.85h
Delayed-HalfCheetah-v3	Mean	**6777.87**	4763.14	4730.74	6276.42	2865.77
	Std	**596.49**	758.29	806.42	857.72	658.37
	Median	**6857.65**	4734.52	4469.37	6372.91	3095.51
	WallClock	4.69h	5.73h	**2.45h**	6.82h	3.15h

teacher networks. Their actions on N sampled observations are recorded as demonstrations. Then, the center actor is treated as the student network to imitate offspring actors. To give different actors different importance, the following weighted imitation training objective is constructed:

$$\mathcal{L}(\pi_\theta) = \sum_{k=1}^{N} \sum_{i=1}^{\lambda} \omega_i ||\pi_\theta(s_k) - \tilde{\pi}_i(s_k)||^2, \sum_{i=1}^{\lambda} \omega_i = 1 \tag{8}$$

Much to our surprise, though the distillation-based method seem to learn a bit faster in the early phase, it quickly fell into the local optimum and can hardly made its way out. This experiment showed that though naive linear recombination may break the behavior of the output network to some extent, this kind of behavior uncertainty may result in extra exploration which is beneficial.

5.3 Comparison to state-of-the-art RL and EA-RL methods

In this section, the performance of the proposed BEL is compared against pure RL methods including TD3 [2], SAC [3] and TD3-Ensemble as well as other EA-RL methods including [11] and an improved version of ERL which is called PDERL [12]. For TD3, CEM-RL, and PDERL, we used the code published by the original authors. For SAC, the stable baselines3 library is used. Every algorithm is run on the same machine, and the results we got were close to what authors had claimed in the original papers. Five tasks from MuJoCo continuous control benchmark are selected. Swimmer-v3 is excluded since it was found that tuning the reward discount factor to 0.9999 could make all algorithms perform more or less the same, reaching approximately 350 reward. Another DelayedHalfCheetah-v3 environment is constructed by delaying the reward signal for 20 time steps, making it a hard exploration task. Following the convention from other literatures, for all algorithms, their learning curves are aggregated over 10 repeated runs across one million time steps. And the evaluated policies are tested for 10 times. For BEL, the population center policy is used for testing. Note that though BEL trains the population in a parallel fashion, for fair comparison, the total time steps are aggregated for every policy interacting with the environment.

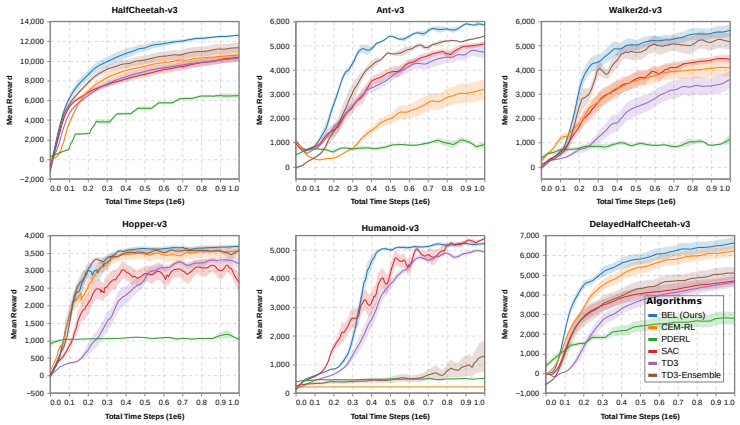


Figure 4: Learning curves on 6 MuJoCo environments in one million time steps.

Sample efficiency It can be seen from fig. 4 that BEL turns out to be very competitive against comparing methods in terms of sample efficiency. On the one hand, it can pick up signals faster than other methods, indicating its high sample efficiency. On the other hand, its final best performance outperforms other methods except on Humanoid-v3.

BEL versus TD3-Ensemble Since we trained multiple actor-critic pairs in BEL, it is natural to question if the good performance of BEL comes from the ensemble nature. To answer this question, we tested the performance of TD3-ensemble where equal numbers of actor-critic pairs are trained, and all hyper parameters are kept as close as possible. From table 1, it is clear that BEL outperforms TD3-ensemble on all tasks.

Stability As can be seen from fig. 4, BEL also generally has smaller standard deviations across runs, even compared to other population based evolutionary methods whose population sizes are larger, this means BEL is very stable. Another phenomenon that suggests BEL's robustness is in the Humanoid-v3 environment, where the naive TD3-Ensemble share the same learning rate (which is larger than single instance TD3) as BEl, but failed to stably learn.

Computation efficiency Since all experiments are conducted on the same machine and all on CPUs, we also recorded the median wall-clock running time of all algorithms. TD3 is the fastest algorithm as it is also the most light-weight one. PDERL ranks the second because not all policies in its population are trained, a great portion of its population are directly evaluated after perturbation. BEL ranks the third among all algorithms, and is generally faster than SAC and CEM-RL. We think BEL reaches a good balance between sample-efficiency and computation overhead.

Limitations Though generally good performance can be expected from BEL, it still has the following limitations. First, as multiple networks are trained in parallel, a computation node with a multi-core CPU and relatively large RAM is required. Second, as can be seen from the Humanoid-v3 where BEL does not outperform SAC, it may indicate BRP and BTT do not scale very well as action space dimension grows. Further studies regarding the scalability of BEL need to be conducted.

6 Conclusion

In this work, a novel population-based evolutionary training framework for off-policy RL algorithms called BEL is proposed. Exploratory and ablative studies show the effectiveness of BRP and BTT. Benchmark comparisons against other methods show BEL outperforms state-of-the-art RL and EA-RL methods in terms of sample efficiency. The training pipeline is conceptually simple and we offer efficient parallel implementation. Along with the improved stability and exploration ability, we believe BEL can serve as a competitive training method for real-world robot learning with off-policy RL algorithms.

References

[1] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. URL http://arxiv.org/abs/1509.02971.

[2] S. Fujimoto, H. van Hoof, and D. Meger. Addressing Function Approximation Error in Actor-Critic Methods. URL http://arxiv.org/abs/1802.09477.

[3] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. URL http://arxiv.org/abs/1801.01290.

[4] R. S. Sutton and A. Barto. *Reinforcement Learning: An Introduction*. Adaptive Computation and Machine Learning. The MIT Press, second edition edition. ISBN 978-0-262-03924-6.

[5] H. van Hasselt, Y. Doron, F. Strub, M. Hessel, N. Sonnerat, and J. Modayil. Deep Reinforcement Learning and the Deadly Triad. URL http://arxiv.org/abs/1812.02648.

[6] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever. Evolution Strategies as a Scalable Alternative to Reinforcement Learning. URL https://arxiv.org/abs/1703.03864v2.

[7] F. P. Such, V. Madhavan, E. Conti, J. Lehman, K. O. Stanley, and J. Clune. Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning. URL http://arxiv.org/abs/1712.06567.

[8] H. Mania, A. Guy, and B. Recht. Simple random search provides a competitive approach to reinforcement learning. URL http://arxiv.org/abs/1803.07055.

[9] S. Khadka and K. Tumer. Evolution-Guided Policy Gradient in Reinforcement Learning. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc. URL https://proceedings.neurips.cc/paper/2018/hash/85fc37b18c57097425b52fc7afbb6969-Abstract.html.

[10] S. Khadka, S. Majumdar, T. Nassar, Z. Dwiel, E. Tumer, S. Miret, Y. Liu, and K. Tumer. Collaborative Evolutionary Reinforcement Learning. URL http://arxiv.org/abs/1905.00976.

[11] A. Pourchot, N. Perrin, and O. Sigaud. Importance mixing: Improving sample reuse in evolutionary policy search methods. URL http://arxiv.org/abs/1808.05832.

[12] C. Bodnar, B. Day, and P. Lió. Proximal Distilled Evolutionary Reinforcement Learning. 34 (04):3283–3290. ISSN 2374-3468, 2159-5399. doi:10.1609/aaai.v34i04.5728.

[13] K. Lee, B.-U. Lee, U. Shin, and I. S. Kweon. An Efficient Asynchronous Method for Integrating Evolutionary and Gradient-based Policy Search. 33:10124–10135. URL https://papers.nips.cc/paper/2020/hash/731309c4bb223491a9f67eac5214fb2e-Abstract.html.

[14] M. Plappert, R. Houthooft, P. Dhariwal, S. Sidor, R. Y. Chen, X. Chen, T. Asfour, P. Abbeel, and M. Andrychowicz. Parameter Space Noise for Exploration. URL http://arxiv.org/abs/1706.01905.

[15] J. Lehman, J. Chen, J. Clune, and K. O. Stanley. Safe Mutations for Deep and Recurrent Neural Networks through Output Gradients. URL http://arxiv.org/abs/1712.06563.

[16] N. Hansen. The CMA Evolution Strategy: A Tutorial. URL http://arxiv.org/abs/1604.00772.

[17] B. Recht. A Tour of Reinforcement Learning: The View from Continuous Control. 2(1): 253–279. doi:10.1146/annurev-control-053018-023825.

[18] D. Wierstra, T. Schaul, T. Glasmachers, Y. Sun, and J. Schmidhuber. Natural Evolution Strategies. URL http://arxiv.org/abs/1106.4487.

[19] I. Rechenberg. Evolutionsstrategie optimierung technischer systeme nach prinzipien der biologischen evolution. URL https://scholar.google.com/scholar_lookup?title=Evolutionsstrategie+Optimierung+technischer+Systeme+nach+Prinzipien+der+biologischen+Evolution&author=Rechenberg%2C+Ingo.&publication_year=1973.

[20] H.-P. Schwefel. Evolutionsstrategien für die numerische optimierung. In H.-P. Schwefel, editor, *Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie: Mit einer vergleichenden Einführung in die Hill-Climbing- und Zufallsstrategie*, Interdisciplinary Systems Research / Interdisziplinäre Systemforschung, pages 123–176. Birkhäuser. ISBN 978-3-0348-5927-1. doi:10.1007/978-3-0348-5927-1_5.

[21] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller. Deterministic Policy Gradient Algorithms. In *International Conference on Machine Learning*, pages 387–395. PMLR. URL http://proceedings.mlr.press/v32/silver14.html.

[22] J. Lehman, J. Chen, J. Clune, and K. O. Stanley. ES is more than just a traditional finite-difference approximator. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '18, pages 450–457. Association for Computing Machinery. ISBN 978-1-4503-5618-3. doi:10.1145/3205455.3205474.