Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI-18)

Back to Basics: Benchmarking Canonical Evolution Strategies for Playing Atari

Patryk Chrabaszcz, Ilya Loshchilov, Frank Hutter
University of Freiburg, Freiburg, Germany
{chrabasp,ilya,fh}@cs.uni-freiburg.de

Abstract

Evolution Strategies (ES) have recently been demonstrated to be a viable alternative to reinforcement learning (RL) algorithms on a set of challenging deep RL problems, including Atari games and MuJoCo humanoid locomotion benchmarks. While the ES algorithms in that work belonged to the specialized class of natural evolution strategies (which resemble approximate gradient RL algorithms, such as REINFORCE), we demonstrate that even a very basic canonical ES algorithm can achieve the same or even better performance. This success of a basic ES algorithm suggests that the state-of-the-art can be advanced further by integrating the many advances made in the field of ES in the last decades.

We also demonstrate qualitatively that ES algorithms have very different performance characteristics than traditional RL algorithms: on some games, they learn to exploit the environment and perform much better while on others they can get stuck in suboptimal local minima. Combining their strengths with those of traditional RL algorithms is therefore likely to lead to new advances in the state of the art.

1 Introduction

In machine learning, Evolution Strategies (ES) are mainly used for direct policy search in reinforcement learning [Gomez *et al.*, 2008; Heidrich-Meisner and Igel, 2009; Stulp and Sigaud, 2013; Salimans *et al.*, 2017] and hyperparameter tuning in supervised learning, e.g., for Support Vector Machines [Glasmachers and Igel, 2008; Igel, 2011] and Deep Neural Networks [Loshchilov and Hutter, 2016].

Recently it has been shown [Salimans *et al.*, 2017] that ES algorithms can be used for tasks which are dominated by deep reinforcement learning (RL) algorithms. Those tasks include learning a policy with discrete action set to control an agent's behavior in a wide variety of Atari environments, as well as learning a policy with continuous action space for agents operating in MuJoCo [Todorov *et al.*, 2012] environments. ES algorithms offer a set of attractive advantages when compared to deep RL algorithms:

- They are highly parallelizable, since the amount of information that has to be exchanged between workers does not depend on the network size.
- Depending on the problem, they can offer better exploration, and as a result different training runs can converge to qualitatively different solutions.
- They are not sensitive to the distribution of rewards and do not require careful tuning of discount factors while still facilitating long-term foresight more than traditional discount-based RL algorithms.
- They can be used for the optimization of non-differentiable policy functions.

In this work, we go one step further than Salimans *et al.* [2017] and study the applicability of even simpler ES algorithm to the task of learning a policy network for playing Atari games. Salimans *et al.* [2017] used a specialized ES algorithm that belongs to the class of Natural Evolution Strategies (NES) [Wierstra *et al.*, 2014], which computes approximate gradients similar to the REINFORCE algorithm [Williams, 1992]. Here, we demonstrate that comparable results can already be achieved with a simpler very basic Canonical ES algorithm from the 1970s [Rechenberg, 1973; Rudolph, 1997].

Our contributions in this work are as follows:

- We demonstrate that even a very basic Canonical ES algorithm is able to match (and sometimes supersede) the performance of the Natural Evolution Strategy used by Salimans *et al.* [2017] for playing Atari games.
- We demonstrate that after 5 hours of training, Canonical ES is able to find novel solutions that exploit the game design and even find bugs in one game that allow them to achieve unprecedented high scores.
- We experimentally study the performance characteristics of both ES algorithms, demonstrating that (1) individual runs have high variance in performance and that (2) longer runs (5h instead of 1h) lead to significant further performance improvements.
- By demonstrating that Canonical ES is a competitive alternative to traditional RL algorithms and the specialized ES algorithms tested so far on the Atari domain we set a benchmark for future work on modern ES variants that are directly based on the canonical version.

2 Background

In this section, we discuss background on RL for playing Atari and on the previously-introduced NES method.

2.1 Reinforcement Learning for Playing Atari

In the Atari task of the OpenAI gym environment [Brockman *et al.*, 2016], the agent needs to learn to maximize its cumulative reward solely by interacting with the environment (i.e., playing the game). Inputs to the agent include the raw pixels displayed on the Atari console as well as the reward signal; its actions correspond to joystick movements to execute.

Recent developments in the field of deep RL have made it possible to address challenging problems that require processing of high dimensional inputs, such as the raw images in this Atari domain, which can be addressed by deep convolutional neural networks. This approach was popularized by Google DeepMind's Nature paper on the deep Q network (DQN) [Mnih and others, 2015], a Q-learning method that estimates the utility of an action given a current state by means of a deep neural network. Given this network for approximating the Q function, in any state s, DQN's policy then simply selects the action a with the largest predicted Q value $Q(s, a)$.

While DQN requires this maximization over the action space, policy gradient algorithms directly parameterize a policy networks that maps a state to a probability distribution over actions. Policy gradient algorithms, such as the Asynchronous Advantage Actor Critic (A3C) [Mnih *et al.*, 2016], directly optimize this policy network.

State Representation. In Atari games it is important to model the state to include information from previous frames that will influence an agent's performance. In this work we use the following standard preprocessing pipeline [Mnih and others, 2015] with an implementation provided by OpenAI [Dhariwal *et al.*, 2017]. First, we apply preprocessing to the screen frames to reduce the dimensionality and remove artifacts related to the technical limitations of the Atari game console (flickering). Specifically, we apply a pixel-wise max operation on the current frame and the one preceeding it. Next, we convert the output from this operation into a grayscale image, resize it and crop it to 84x84 pixels. At the end of this pipeline, we stack together the result of the 4 last frames produced this way to construct a 84x84x4 state tensor. Also following common practice, to speed up policy evaluation (and thus reduce training time), instead of collecting every frame and making a decision at every step, we collect every 4th frame (3rd frame for SpaceInvaders to make the laser visible [Mnih *et al.*, 2013]) and apply the same action for all frames in between. Figure 1 visualizes the full preprocessing pipeline. Figure 2 shows an example of the state representation for 2 different games.

2.2 Natural Evolution for Playing Atari

Salimans *et al.* [2017] recently demonstrated that an ES algorithm from the specialized class of Natural Evolution Strategies (NES; [Wierstra *et al.*, 2014]) can be used to successfully train policy networks in a set of RL benchmark environments (Atari, MuJoCo) and compete with state-of-the-art RL algorithms. Algorithm 1 describes their approach on a high level. In a nutshell, it evolves a distribution over policy networks

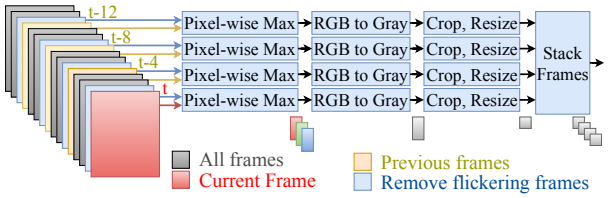

Figure 1: Preprocessing pipeline. Take every 4th frame, apply max operation to remove screen flickering, convert to grayscale, resize/crop, stack 4 last frames.

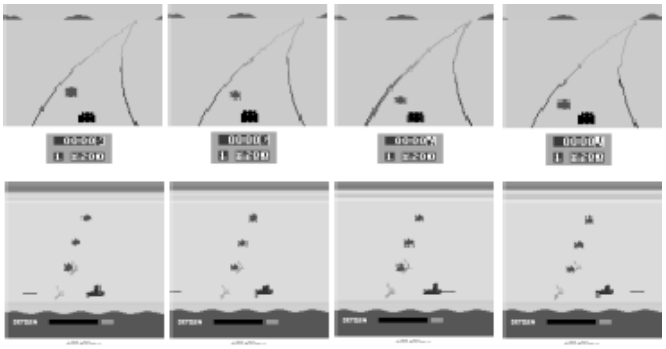

Figure 2: State representation (84x84x4 tensor) for 2 different games: Enduro and Seaquest. Channels are shown next to each other for better visualization.

over time by evaluating a population of λ different networks in each iteration, starting from initial policy parameter vector θ_0. At each iteration t, the algorithm evaluates the game scores $F(\cdot)$ of λ different policy parameter vectors centered around θ_t (lines 3-5) to estimate a gradient signal, using mirrored sampling [Brockhoff *et al.*, 2010] to reduce the variance of this estimate. Since the λ game evaluations are independent of each other, ES can make very efficient use of parallel compute resources. The resulting λ game scores are then ranked (line 6), making the algorithm invariant to their scale; as noted by the authors, this approach (called fitness shaping [Wierstra *et al.*, 2014] but used in all ESs since the 1970s) decreases the probability of falling into local optima early and lowers the influence of outliers. Based on these λ ranks of local steps around θ_t, the algorithm approximates a gradient g (line 7) and uses this with a modern version of gradient descent (Adam [Kingma and Ba, 2014] or SGD with momentum) with weight decay to compute a robust parameter update step (line 8) in order to move θ_t towards the parameter vectors that achieved higher scores.

We note that the computation of the approximate gradient g in line 7 follows the same approach as the well-known policy gradient algorithm REINFORCE. This can be shown as follows. Denoting the distribution from which we draw policy network parameters θ as p_ψ, the gradient of the expected reward $F(\theta)$ with respect to ψ is:

$$\nabla_\psi \mathbb{E}_{\theta\sim p_\psi}\{F(\theta)\} = \mathbb{E}_{\theta\sim p_\psi}\{F(\theta)\nabla_\psi \log p_\psi(\theta)\}. \quad (1)$$

Because p_ψ is chosen as an isotropic Gaussian distribution with mean θ_t and fixed standard deviation (mutation step-size)

Algorithm 1: OpenAI ES

Input:
$optimizer$ - Optimizer function
σ - Mutation step-size
λ - Population size
θ_0 - Initial policy parameters
F - Policy evaluation function

1 **for** *t = 0, 1, ...* **do**
2 **for** *i = 1, 2, ...* $\frac{\lambda}{2}$ **do**
3 Sample noise vector: $\epsilon_i \sim \mathcal{N}(0, I)$
4 Evaluate score in the game: $s_i^+ \leftarrow F(\theta_t + \sigma * \epsilon_i)$
5 Evaluate score in the game: $s_i^- \leftarrow F(\theta_t - \sigma * \epsilon_i)$
6 Compute normalized ranks: $r = ranks(s), r_i \in [0, 1)$
7 Estimate gradient: $g \leftarrow \frac{1}{\sigma * \lambda} \sum_{i=1}^{\lambda} (r_i * \epsilon_i)$
8 Update policy network: $\theta_{t+1} \leftarrow \theta_t + optimizer(g)$

σ, the only parameter of p_ψ is θ_t and we have:

$$\nabla_\psi \log p_\psi(\theta) = \nabla_{\theta_t} \log \frac{1}{\sigma\sqrt{2\pi}} e^{-(\theta-\theta_t)^2/2\sigma^2} = \frac{\theta - \theta_t}{\sigma^2} \tag{2}$$

and therefore the following identity holds:

$$\nabla_\psi \mathbb{E}_{\theta\sim p_\psi}\{F(\theta)\} = \mathbb{E}_{\theta\sim p_\psi}\{F(\theta) * \frac{\theta - \theta_t}{\sigma^2}\} \tag{3}$$

$$\approx \frac{1}{\sigma * \lambda} \sum_{i=1}^{\lambda} F(\theta^{(i)}) * \frac{(\theta^{(i)} - \theta_t)}{\sigma}, \tag{4}$$

where the last step is simply an approximation by λ samples $\theta^{(i)} \sim p_\psi$. Equation 4 is exactly as in line 7 of the algorithm except that the raw game scores $F(\theta^{(i)})$ are replaced with their ranks r_i due to fitness shaping.

Salimans *et al.* [2017] also made two further contributions to stabilize the training and improve performance. Firstly, they introduced a novel parallelization technique (which uses a noise table to reduce communication cost in order to scale to a large number of λ parallel workers) and used virtual batch normalization [Salimans *et al.*, 2016] to make the network output more sensitive to the noise in the parameter space.

3 Canonical ES

While the specialized ES algorithm proposed by OpenAI is equivalent to a policy gradient algorithm, in this work we consider a very basic canonical ES algorithm that belongs to the prominent family of $(\mu, \lambda) - ES$ optimization algorithms. Algorithm 2 illustrates this simple approach. Starting from a random parameter vector θ_0, in each iteration t we generate an offspring population of size λ. For each element of the population, we add sampled mutation noise $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$ to the current parameter vector θ_t (line 3) and evaluate the game score of the resulting vector by one episode rollout (line 4). We then pick the top μ parameter vectors according to the collected score and form a new parameter vector θ_{t+1} as their weighted mean (lines 5-6).

This algorithm is very basic in its setting and conceptually simpler than the NES-inspired algorithm by Salimans *et al.* [2017]. While our presentation of the latter (see Algorithm 1) hides complexity in the final call to an advanced gradient-based optimizer (Salimans *et al.* [2017] used Adam [Kingma and Ba, 2014] for MujoCo in their publicly available implementation), in canonical ES we do not use any advanced optimizer. We also do not decay the parameters. We use standard weights to compute the weighted mean of the top μ solutions instead of uniform weights used in OpenAI ES. The new elements introduced by Salimans *et al.* [2017] that we *do* use are virtual batch normalization (which is a component of the game evaluations F and not really of the ES algorithm itself) and the efficient parallelization of ES using a random noise table.

We initially implemented the Cumulative Step-size σ Adaptation (CSA) procedure [Hansen and Ostermeier, 1996], which is standard in canonical ES algorithms. However, due to the high time cost of game evaluations, during our time-limited training, we are only able to perform up to thousands update iterations. Since the dimensionality of the parameter vector is relatively large (1.7M), this results in only a negligible change of σ during the training. Therefore, effectively, our algorithm used a fixed step-size and thus we removed step-size adaptation from the description from Algorithm 2 making it even somewhat simpler than typical ES. We employed weighted recombination [Rudolph, 1997] and weights w as in CSA-ES.

Algorithm 2: Canonical ES Algorithm

Input:
σ - Mutation step-size
θ_0 - Initial policy parameters
F - Policy evaluation function
λ - Offspring population size
μ - Parent population size
Initialize :
$w_i = \frac{\log(\mu+0.5)-\log(i)}{\sum_{j=1}^{\mu} \log(\mu+0.5)-\log(j)}$

1 **for** $t = 0, 1, ...$ **do**
2 **for** $i = 1 ... \lambda$ **do**
3 Sample noise: $\epsilon_i \sim \mathcal{N}(0, I)$
4 Evaluate score in the game: $s_i \leftarrow F(\theta_t + \sigma * \epsilon_i)$
5 Sort $(\epsilon_1, \ldots, \epsilon_\lambda)$ according to s (ϵ_i with best s_i first)
6 Update policy: $\theta_{t+1} \leftarrow \theta_t + \sigma * \sum_{j=1}^{\mu} w_j * \epsilon_j$
7 Optionally, update step size σ (see text)

4 Experiments

In our experiments, we evaluate the performance of the Canonical ES on a subset of 8 Atari games available in OpenAI Gym [Brockman *et al.*, 2016]. We selected these games to represent different levels of difficulty, ranging from simple ones like Pong and Breakout to complex games like Qbert and Alien. We make our implementation of the Canonical ES algorithm available online at https://github.com/PatrykChrabaszcz/Canonical_ES_Atari.

We compare our results against those obtained with the ES algorithm proposed by OpenAI [Salimans *et al.*, 2017]. Since no implementation of that algorithm is publicly available for Atari games, we re-implemented it with help from OpenAI[1],

[1]We thank Tim Salimans for his helpful email support.

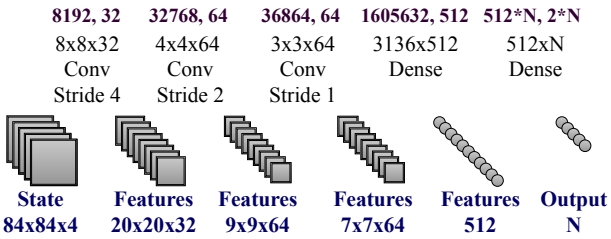

Figure 3: Neural network architecture. Numbers on top show number of parameters in each layer (kernel parameters and batch norm parameters). Each batch norm layer has a trainable shift parameter β; the last batch norm has an additional trainable scale parameter α.

and the results of our implementation, which we refer to as "OpenAI ES (our)", roughly match those reported by Salimans *et al.* [2017], which we refer to as "OpenAI ES" (see Table 1). We believe that differences between the original OpenAI results and the results from our reimplementation are mainly due to the fact that our approach does not adjust the max episode length during training. Thus, from the beginning, we shift the search direction towards solutions that perform well in long runs. This seems to have varying effects that strongly depends on the game.

Network Architecture. We use the same network structure as the original DQN work [Mnih and others, 2015], only changing the activation function from ReLU to ELU [Clevert *et al.*, 2015] and adding batch normalization layers [Ioffe and Szegedy, 2015]. The network as presented in Figure 3 has approximately 1.7M parameters. We initialize network weights using samples from a normal distribution $\mathcal{N}(\mu = 0, \sigma = 0.05)$.

Virtual Batch Normalization. Following Salimans *et al.* [2017], we use virtual batch normalization [Salimans *et al.*, 2016]. In order to collect the reference batch, at the beginning of the training we play the game using random actions. In each step, we save the corresponding state with the probability $p(save) = 1\%$ and stop when 128 samples have been collected.

Training. For each game and each ES variant we tested, we performed 3 training runs, each on 400 CPUs with a time budget of 10 hours. Every worker (CPU) evaluates 2 offspring solutions, meaning that our setting is roughly the same as training for 5 hours with full parallelization (800 CPUs); therefore, we label this setting as "5 hours". In addition, we save the solution proposed after 2 hours of training (equivalent to 1 hour with full parallelization) or after 1 billion training frames (whatever comes first) to allow for a fair comparison with results reported by Salimans *et al.* [2017]; we label this setting as "1 hour". During training, one CPU is reserved to evaluate the performance of the solution proposed in the current iteration; hence, the offspring population size in our experiments is $\lambda = 798$. In each decision step, the agent passes its current environment state through the network and performs an action that corresponds to the output with the highest value. We limit episodes to have a maximum length of 25k steps; we do not adjust this value during training.

An episode includes multiple lives, and we do not terminate an episode after the agent dies the first time in order to allow the learning of strategies that span across multiple lives. We start each episode with up to 30 initial random no-op actions.

Results. First, we studied the importance of the parent population size μ. This hyperparameter is known to often be important for ES algorithms and we found this to also hold here. We measured performance for $\mu \in \{10, 20, 50, 100, 200, 400\}$ and observed different optimal values of μ for different games (Figure 4). For the subsequent analyses we fixed $\mu = 50$ for all games.

In Table 1, for each game and for both Canonical ES and OpenAI ES, we report the results of our 3 training runs; for each of these runs, we evaluated the final policy found using 30 rollouts. We ordered the 3 runs according to their mean score and compared the results row-wise. We ran a Mann-Whitney U test to check if the distributions of the 30 rewards significantly differed between the two ES variants. After 5 hours of training, Canonical ES performed significantly better than OpenAI ES in 9 out of 24 different runs ($p < 0.05$) and worse in 7 (with 8 ties); this shows that our simple Canonical ES is competitive with the specialized OpenAI ES algorithm on complex benchmark problems from the Atari domain. Additionally, we tested whether the algorithms still made significant progress between 1 and 5 hours of training; indeed, the performance of our Canonical ES algorithm improved significantly in 16 of 24 cases, demonstrating that it often could make effective use of additional training time. However, qualitatively, we observed that the performance of both algorithms tends to plateau in locally optimal solutions for extended periods of time (see Figure 5) and that they often find solutions that are not robust to the noise in the domain; i.e., there is a high variance in the points scored with the same trained policy network across initial environment conditions.

5 Qualitative Analysis

Visual inspection and comparison between solutions found by reinforcement learning algorithms and solutions found by ES algorithms shows some significant differences. In this section we describe interesting agent behaviors on different games; a video with evaluation runs on all games is available on-line at goo.gl/Mz2ZUs.

First, we study two games in which most of the ES runs converged to similar sub-optimal solutions: Seaquest and Enduro. In Seaquest, the agent dives to the bottom of the sea and starts to shoot left and right, occasionally hitting an enemy and scoring points (Figure 6). However, it is not able to detect the lack of oxygen and quickly dies. In Enduro, the agent steers the car to keep it in the middle of the road and accelerate, but from time to time it bounces back after hitting a rival car in front of it. Both solutions are easy-to-reach local optima and do not require developing any complex behavior; since the corresponding scores achieved with these policies are much higher than those of random policies we believe that it is hard for ES to escape from these local attractor states in policy space.

We next study the game Qbert, in which Canonical ES found two particularly interesting solutions that exploit bugs

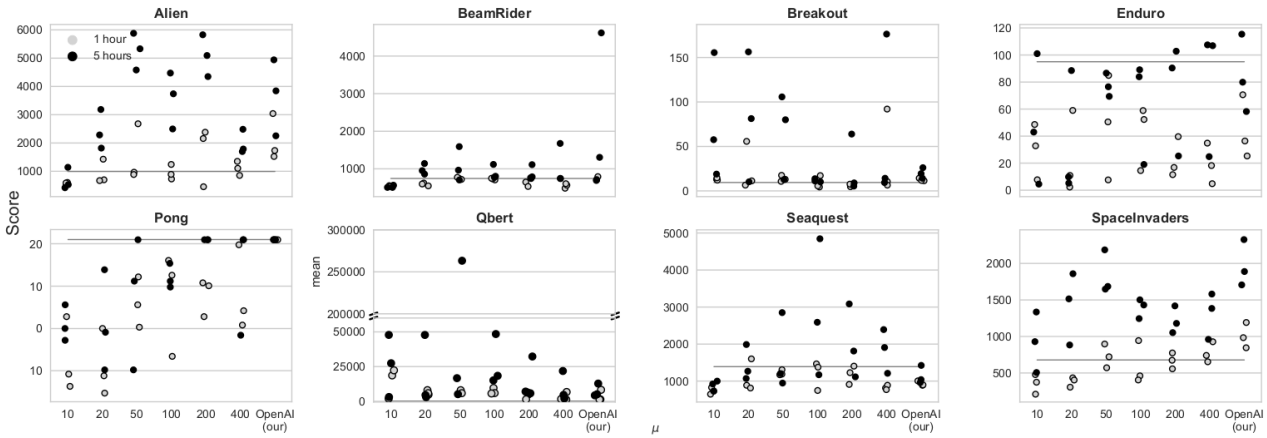

Figure 4: Final evaluation scores (mean across 30 evaluation rollouts with random initial no-ops) for Canonical ES ($\mu \in 10, 20, 50, 100, 200, 400$) and for our implementation of OpenAI ES. For each setting we use population size $\lambda = 798$ and report the performance from 3 separate training runs. We report evaluation scores after 1 hour and after 5 hours of training. The horizontal line indicates the results reported by Salimans *et al.* [2017].

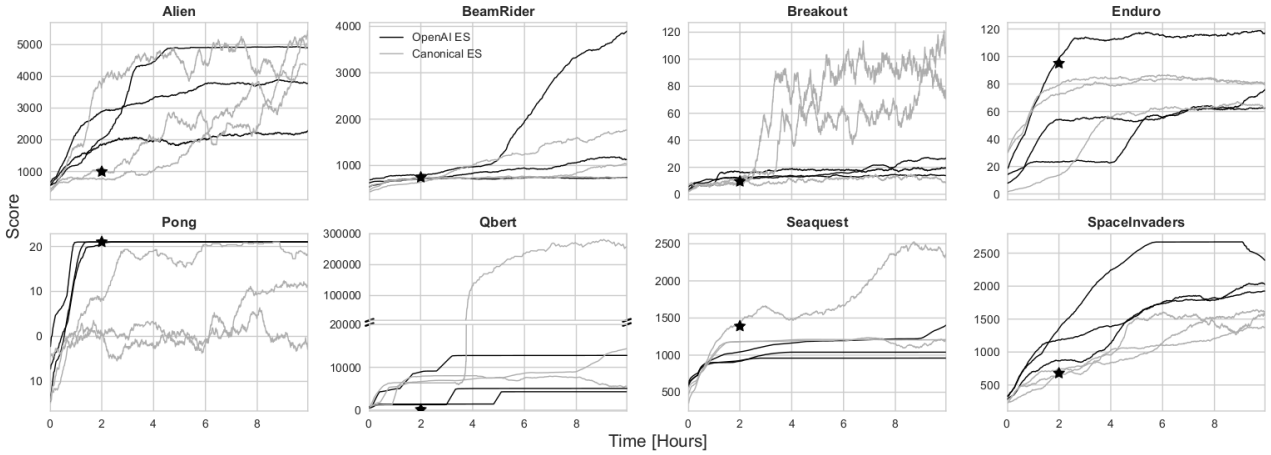

Figure 5: Training curves for CanonicalES ($\mu = 50$) and OpenAI ES (our). At each iteration t we evaluate currently proposed solution θt two times using one CPU, because of that values reported in the Table1 (mean over 30 evaluation runs) might differ. For better plot quality we filter out the noise. We observe that both algorithms often get stuck in local optima.

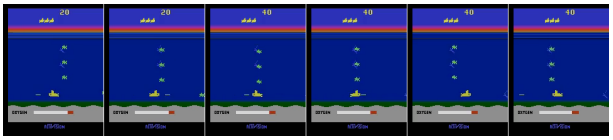

Figure 6: The agent learns to dive to the bottom of the sea and constantly shoot left and right, occasionally scoring points.

in the game. In the first case, which turned out to be a known bug, instead of completing the first level, the agent loops a sequence of actions, in which it baits an enemy to jump off the game platform together and scores the points for killing the enemy; for some unknown reason, the game engine does not count this suicide as a loss of life, and the agent is able to repeat this process indefinitely. This pattern is shown in Figure 7 (top) and in a video at goo.gl/ig8E3t. In the second, more interesting, and previously unknown bug, the agent finds a sequence of actions that completes the first level, but, for an unknown reason, does not lead to the game advancing to the next level; instead, all platforms start to blink and the agent quickly gains a huge amount of points. This pattern is shown in Figure 7 (bottom) and in a video at goo.gl/1K9A17; it has since been reproduced on the original Atari console (goo.gl/jkt6jS).

Breakout seems to be a challenging environment for ES algorithms. Canonical ES only found reasonable solutions for a few settings. The best solution shows a strategy that looks similar to the best strategies obtained by using reinforcement learning algorithms, in which the agent creates a hole on one side of the board and shoots the ball through it to gain many points in a short amount of time (Figure 8). However, even this best solution found by ES is not stable: for different initial environment conditions the agent with the same policy network quickly loses the game with only few points.

	OpenAI ES 1 hour	OpenAI ES (our) 1 hour	Canonical ES 1 hour	OpenAI ES (our) 5 hours	Canonical ES 5 hours
Alien		**3040 ± 276.8**	2679.3 ± 1477.3	4940 ± 0	**5878.7 ± 1724.7**
Alien	994	**1733.7 ± 493.2**	965.3 ± 229.8	3843.3 ± 228.7	**5331.3 ± 990.1**
Alien		**1522.3 ± 790.3**	885 ± 469.1	2253 ± 769.4	**4581.3 ± 299.1**
BeamRider		**792.3 ± 146.6**	774.5 ± 202.7	**4617.1 ± 1173.3**	1591.3 ± 575.5
BeamRider	744	708.3 ± 194.7	**746.9 ± 197.8**	**1305.9 ± 450.4**	965.3 ± 441.4
BeamRider		690.7 ± 87.7	**719.6 ± 197.4**	**714.3 ± 189.9**	703.5 ± 159.8
Breakout		14.3 ± 6.5	**17.5 ± 19.4**	26.1 ± 5.8	**105.7 ± 158**
Breakout	9.5	11.8 ± 3.3	**13 ± 17.1**	19.4 ± 6.6	**80 ± 143.4**
Breakout		**11.4 ± 3.6**	10.7 ± 15.1	**14.2 ± 2.7**	12.7 ± 17.7
Enduro		70.6 ± 17.2	**84.9 ± 22.3**	**115.4 ± 16.6**	86.6 ± 19.1
Enduro	95	36.4 ± 12.4	**50.5 ± 15.3**	**79.9 ± 18**	76.5 ± 17.7
Enduro		**25.3 ± 9.6**	7.6 ± 5.1	58.2 ± 10.5	**69.4 ± 32.8**
Pong		**21.0 ± 0.0**	12.2 ± 16.6	**21.0 ± 0.0**	**21.0 ± 0.0**
Pong	21	**21.0 ± 0.0**	5.6 ± 20.2	**21 ± 0**	11.2 ± 17.8
Pong		**21.0 ± 0.0**	0.3 ± 20.7	**21 ± 0**	−9.8 ± 18.6
Qbert		**8275 ± 0**	8000 ± 0	12775 ± 0	**263242 ± 433050**
Qbert	147.5	1400 ± 0	**6625 ± 0**	5075 ± 0	**16673.3 ± 6.2**
Qbert		1250 ± 0	**5850 ± 0**	4300 ± 0	**5136.7 ± 4093.9**
Seaquest		1006 ± 20.1	**1306.7 ± 262.7**	1424 ± 26.5	**2849.7 ± 599.4**
Seaquest	1390	898 ± 31.6	**1188 ± 24**	1040 ± 0	**1202.7 ± 27.2**
Seaquest		887.3 ± 20.3	**1170.7 ± 23.5**	**960 ± 0**	946.7 ± 275.1
SpaceInvaders		**1191.3 ± 84.6**	896.7 ± 123	**2326.5 ± 547.6**	2186 ± 1278.8
SpaceInvaders	678.5	**983.7 ± 158.5**	721.5 ± 115	**1889.3 ± 294.3**	1685 ± 648.6
SpaceInvaders		**845.3 ± 69.7**	571.3 ± 98.8	**1706.5 ± 118.3**	1648.3 ± 294.5

Table 1: Evaluation scores (mean over 30 evaluation runs with up to 30 initial no-ops) for different training times and algorithms. Second column contains results as reported in the previous OpenAI work. For each training time limit we compare results from OpenAI ES(our) and Canonical ES $\mu = 50$. For each setting we performed 3 training runs, ordered the results for each game and compared them row by row, boldfacing the better score. Results for which the difference is significant across the 30 evaluation runs based on a Mann-Whitney U test ($p < 0.05$) are marked in blue.

In the games SpaceInvaders and Alien we also observed interesting strategies. We do not clip rewards during the training as is sometimes done for reinforcement learning algorithms. Because of that the agent puts more attention to behaviors that result in a higher reward, sometimes even at the cost of the main game objective. In SpaceInvaders, we observe that in the best solution the agent hits the mother-ship that appears periodically on the top of the screen with 100% accuracy. In Alien, the agent focuses on capturing an item that makes it invincible and then goes to the enemy spawn point to collect rewards for eliminating newly appearing enemies. However, the agent is not able to detect when the invincibility period ends.

6 Recent Related Work

Evolution strategies, such as the Covariance Matrix Adaptation Evolution Strategy (CMA-ES [Hansen *et al.*, 2003]), are commonly used as a baseline approach in reinforcement learning tasks [Heidrich-Meisner and Igel, 2009; Stulp and Sigaud, 2013; Duan *et al.*, 2016; Li, 2017]. Here, we only discuss the most recent related works, which also followed up on the work by Salimans *et al.* [2017]; in particular, we discuss three related arXiv preprints that scientists at Uber released in the last two months about work concurrent to ours.

Similarly to our work, Such *et al.* [2017] studied the performance of simpler algorithms than OpenAI's specialized ES variant. They show that genetic algorithms (another broad class of black-box optimization algorithms) can also reach results competitive to OpenAI's ES variant and other RL algorithms. Additionally, interestingly, the authors show that for some of the games even simple random search can outperform carefully designed RL and ES algorithms.

Lehman *et al.* [2017] argue that comparing ES to finite-difference-based approximation is too simplistic. The main difference comes from the fact that ES tries to optimize the performance of the distribution of solutions rather than a single solution, thus finding solutions that are more robust to the noise in the parameter space. The authors leave open the question whether this robustness in the parameter space also affects the robustness to the noise in the domain. In our experiments we observe that even for the best solutions on some of the games, the learned policy network is not robust against environment noise.

Conti *et al.* [2017] try to address the problems of local minima (which we also observed in games like Seaquest and Enduro) by augmenting ES algorithms with a novelty search and quality diversity. Their proposed algorithms add an additional criterion to the optimization procedure that encourages exploring qualitatively different solutions during training, thus reducing the risk of getting stuck in a local optimum. The authors also propose to manage a meta-population of diverse solutions allocating resources to train more promising ones. In our experiments we observe that training runs with the same hyperparameters and different initializations often converge

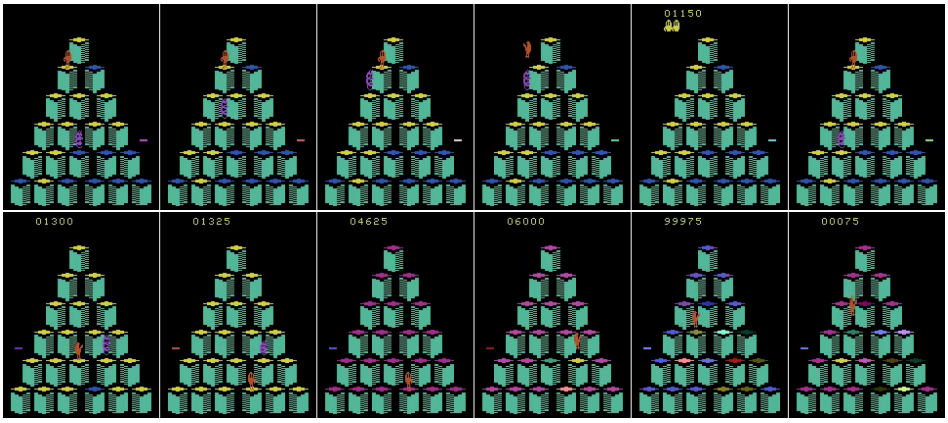

Figure 7: Top: the agent (orange blob in the upper left part of the screen) learns to commit suicide to kill its enemy (purple spring); because of the bug, the game does not count this as a loss of life. Bottom: the agent uses a bug in the game: after solving the first level in a specific sequence, the game does not advance to the next level, but instead the platforms start to blink and the agent gains huge amount of points.

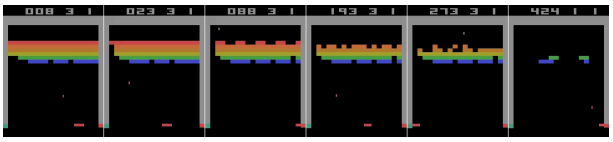

Figure 8: The agent learns to make a hole in the brick wall to collect many points with one ball bounce.

to achieve very different scores; managing a meta-population could therefore be an easy way to improve the results and reduce the variance between the runs. Overall, the success of Conti *et al.* [2017] in improving performance with some of these newer ES methods strengthens our expectation that a wide range of improvements to the state of the art are possible by integrating the multitude of techniques developed in ES research over the last decades into our canonical ES.

7 Conclusion and Future Work

The recent results provided by Open AI [Salimans *et al.*, 2017] suggest that natural evolution strategies represent a viable alternative to more common approaches used for deep reinforcement learning. In this work, we analyzed whether the results demonstrated in that work are due to the special type of ES used there. Our results suggest that even a very basic decades-old ES provides comparable results; thus, more modern ES methods should be considered as a potentially competitive approach to modern deep reinforcement learning algorithms.

In future work, we plan on improving upon the current canonical ES algorithm in several ways. We plan to study CMA-ES for problems where small networks can be applied. Both ES algorithm variants studied in this paper are characterized by a high variance, both between runs with the same hyperparameters and between evaluation runs of the same already trained policy. We believe that the high variance between the runs could be addressed by managing a meta population of solutions and allocating resources to the more promising ones. We also plan to explore possible ways to reduce currently high computational requirements of our approach; preliminary experiments suggest that lower population sizes still obtain good results. One of the drawbacks of ES algorithms we discussed is that they tend to get stuck in locally optimal solutions. We believe that this can be addressed using quality diversity and novelty search algorithms, thus making ES more robust and potentially improving the score in longer training runs. It would be promising to study lower-memory ES methods for larger networks, such as LM-MA-ES [Loshchilov *et al.*, 2017], which we expect to yield further improvements over canonical ES.

We also believe that it is promising to combine ES and RL algorithms. For example, Plappert *et al.* [2017] propose to augment RL with noise injection in the parameter space (as in an ES algorithm) to improve the exploration, and demonstrates the benefits of this approach. We expect that there are further opportunities along these lines to gain improvements by more explicitly combining RL and ES.

Acknowledgments

This work has partly been supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant no. 716721. The authors acknowledge support by the state of Baden-Württemberg through bwHPC and the German Research Foundation (DFG) through grant no. INST 39/963-1 FUGG.

References

[Brockhoff *et al.*, 2010] Dimo Brockhoff, Anne Auger, Nikolaus Hansen, Dirk V Arnold, and Tim Hohm. Mirrored sampling and sequential selection for evolution strategies. In *PPSN*, pages 11–21. Springer, 2010.

[Brockman *et al.*, 2016] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[Clevert *et al.*, 2015] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep

network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

[Conti *et al.*, 2017] Edoardo Conti, Vashisht Madhavan, Felipe Petroski Such, Joel Lehman, Kenneth O Stanley, and Jeff Clune. Improving exploration in evolution strategies for deep reinforcement learning via a population of novelty-seeking agents. *arXiv preprint arXiv:1712.06560*, 2017.

[Dhariwal *et al.*, 2017] Prafulla Dhariwal, Christopher Hesse, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. Openai baselines. https://github.com/openai/baselines, 2017.

[Duan *et al.*, 2016] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. In *ICML*, pages 1329–1338, 2016.

[Glasmachers and Igel, 2008] Tobias Glasmachers and Christian Igel. Uncertainty handling in model selection for support vector machines. In *PPSN*, pages 185–194, 2008.

[Gomez *et al.*, 2008] Faustino Gomez, Jürgen Schmidhuber, and Risto Miikkulainen. Accelerated neural evolution through cooperatively coevolved synapses. *Journal of Machine Learning Research*, 9(May):937–965, 2008.

[Hansen and Ostermeier, 1996] Nikolaus Hansen and Andreas Ostermeier. Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation. In *CEC 1996*, pages 312–317. IEEE, 1996.

[Hansen *et al.*, 2003] Nikolaus Hansen, Sibylle D Müller, and Petros Koumoutsakos. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). *Evolutionary computation*, 11(1):1–18, 2003.

[Heidrich-Meisner and Igel, 2009] Verena Heidrich-Meisner and Christian Igel. Hoeffding and Bernstein races for selecting policies in evolutionary direct policy search. In *ICML*, pages 401–408. ACM, 2009.

[Igel, 2011] Christian Igel. Evolutionary kernel learning. In *Encyclopedia of Machine Learning*, pages 369–373. Springer, 2011.

[Ioffe and Szegedy, 2015] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, pages 448–456, 2015.

[Kingma and Ba, 2014] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[Lehman *et al.*, 2017] Joel Lehman, Jay Chen, Jeff Clune, and Kenneth O Stanley. ES is more than just a traditional finite-difference approximator. *arXiv preprint arXiv:1712.06568*, 2017.

[Li, 2017] Yuxi Li. Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274*, 2017.

[Loshchilov and Hutter, 2016] Ilya Loshchilov and Frank Hutter. CMA-ES for Hyperparameter Optimization of Deep Neural Networks. *arXiv preprint arXiv:1604.07269*, 2016.

[Loshchilov *et al.*, 2017] Ilya Loshchilov, Tobias Glasmachers, and Hans-Georg Beyer. Limited-memory matrix adaptation for large scale black-box optimization. *arXiv preprint arXiv:1705.06693*, 2017.

[Mnih and others, 2015] Volodymyr Mnih et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[Mnih *et al.*, 2013] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[Mnih *et al.*, 2016] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *ICML*, pages 1928–1937, 2016.

[Plappert *et al.*, 2017] Matthias Plappert, Rein Houthooft, Prafulla Dhariwal, Szymon Sidor, Richard Y Chen, Xi Chen, Tamim Asfour, Pieter Abbeel, and Marcin Andrychowicz. Parameter space noise for exploration. *arXiv preprint arXiv:1706.01905*, 2017.

[Rechenberg, 1973] Ingo Rechenberg. Evolutionsstrategie–optimierung technisher systeme nach prinzipien der biologischen evolution. 1973.

[Rudolph, 1997] Günter Rudolph. *Convergence properties of evolutionary algorithms*. Kovac, 1997.

[Salimans *et al.*, 2016] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *NIPS*, pages 2234–2242, 2016.

[Salimans *et al.*, 2017] Tim Salimans, Jonathan Ho, Xi Chen, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*, 2017.

[Stulp and Sigaud, 2013] Freek Stulp and Olivier Sigaud. Robot skill learning: From reinforcement learning to evolution strategies. *Paladyn, Journal of Behavioral Robotics*, 4(1):49–61, 2013.

[Such *et al.*, 2017] Felipe Petroski Such, Vashisht Madhavan, Edoardo Conti, Joel Lehman, Kenneth O Stanley, and Jeff Clune. Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. *arXiv preprint arXiv:1712.06567*, 2017.

[Todorov *et al.*, 2012] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *IROS*, pages 5026–5033. IEEE, 2012.

[Wierstra *et al.*, 2014] Daan Wierstra, Tom Schaul, Tobias Glasmachers, Yi Sun, Jan Peters, and Jürgen Schmidhuber. Natural evolution strategies. *Journal of Machine Learning Research*, 15(1):949–980, 2014.

[Williams, 1992] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Reinforcement Learning*, pages 5–32. Springer, 1992.