Deep Neuroevolution: Genetic Algorithms are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning

Felipe Petroski Such Vashisht Madhavan Edoardo Conti Joel Lehman Kenneth O. Stanley Jeff Clune

Uber AI Labs

{felipe.such, jeffclune}@uber.com

Abstract

Deep artificial neural networks (DNNs) are typically trained via gradient-based learning algorithms, namely backpropagation. Evolution strategies (ES) can rival backprop-based algorithms such as Q-learning and policy gradients on challenging deep reinforcement learning (RL) problems. However, ES can be considered a gradient-based algorithm because it performs stochastic gradient descent via an operation similar to a finite-difference approximation of the gradient. That raises the question of whether non-gradient-based evolutionary algorithms can work at DNN scales. Here we demonstrate they can: we evolve the weights of a DNN with a simple, gradient-free, population-based genetic algorithm (GA) and it performs well on hard deep RL problems, including Atari and humanoid locomotion. The Deep GA successfully evolves networks with over four million free parameters, the largest neural networks ever evolved with a traditional evolutionary algorithm. These results (1) expand our sense of the scale at which GAs can operate, (2) suggest intriguingly that in some cases following the gradient is not the best choice for optimizing performance, and (3) make immediately available the multitude of techniques that have been developed in the neuroevolution community to improve performance on RL problems. To demonstrate the latter, we show that combining DNNs with novelty search, which was designed to encourage exploration on tasks with deceptive or sparse reward functions, can solve a high-dimensional problem on which reward-maximizing algorithms (e.g. DQN, A3C, ES, and the GA) fail. Additionally, the Deep GA parallelizes better than ES, A3C, and DQN, and enables a state-of-the-art compact encoding technique that can represent million-parameter DNNs in thousands of bytes.

1. Introduction

A recent trend in machine learning and AI research is that old algorithms work remarkably well when combined with sufficient computing resources and data. That has been the story for (1) backpropagation applied to deep neural networks in supervised learning tasks such as computer vision (Krizhevsky et al., 2012) and voice recognition (Seide et al., 2011), (2) backpropagation for deep neural networks combined with traditional reinforcement learning algorithms, such as Q-learning (Watkins & Dayan, 1992; Mnih et al., 2015) or policy gradient (PG) methods (Sehnke et al., 2010; Mnih et al., 2016), and (3) evolution strategies (ES) applied to reinforcement learning benchmarks (Salimans et al., 2017). One common theme is that all of these methods are gradient-based, including ES, which involves a gradient approximation similar to finite differences (Williams, 1992; Wierstra et al., 2008; Salimans et al., 2017). This historical trend raises the question of whether a similar story will play out for gradient-free methods, such as population-based GAs.

This paper investigates that question by testing the performance of a simple GA on hard deep reinforcement learning (RL) benchmarks, including Atari 2600 (Bellemare et al., 2013; Brockman et al., 2016; Mnih et al., 2015) and Humanoid Locomotion in the MuJoCo simulator (Todorov et al., 2012; Schulman et al., 2015; 2017; Brockman et al., 2016). We compare the performance of the GA with that of contemporary algorithms applied to deep RL (i.e. DQN (Mnih et al., 2015), a Q-learning method, A3C (Mnih et al., 2016), a policy gradient method, and ES). One might expect GAs to perform far worse than other methods because they are so simple and do not follow gradients. Surprisingly, we found that GAs turn out to be a competitive algorithm for RL – performing better on some domains and worse on others – adding a new family of algorithms to the toolbox for deep RL problems. We also validate the effectiveness of learning with GAs by comparing their performance to that of random search. While the GA always outperforms random search, interestingly we discovered that in some Atari games random search outperforms power-

ful deep RL algorithms (DQN on 4/13 games and A3C on 5/13) and ES (3/13), suggesting that these domains may be easier than previously thought, at least for some algorithms, and that local optima, saddle points, noisy gradient estimates, or some other force is impeding progress on these problems for gradient-based methods. Note that although deep neural networks often do not struggle with local optima in supervised learning (Pascanu et al., 2014), local optima remain an issue in RL because the reward signal may deceptively encourage the agent to perform actions that prevent it from discovering the globally optimal behavior.

Like ES and the deep RL algorithms, the GA has unique benefits. One is that, through a new technique we introduce, GAs turn out to be faster than ES due to their being even more amenable to parallelization. The GA and ES are both faster than Q-learning and policy gradient methods when substantial distributed computation is available (here, 720 CPU cores across dozens of machines). Another benefit is that, via this same new technique, even multi-million-parameter networks trained by GAs can be encoded with very few (thousands of) bytes, yielding what we believe to be a state-of-the-art compact encoding method.

In general, the unexpectedly competitive performance of the GA (and even random search) suggests that the structure of the search space in some of these domains is not always amenable to gradient-based search. That realization can open up new research directions, focusing on when and how to exploit the regions where a gradient-free search might be more appropriate. It also expands the toolbox of ideas and methods applicable for RL and may lead to new kinds of hybrid algorithms.

2. Background

At a high level, an RL problem challenges an agent to maximize some notion of cumulative reward (e.g. total, or discounted) for a given problem without supervision as to how to accomplish that goal (Sutton & Barto, 1998). A host of traditional RL algorithms perform well on small, tabular state spaces (Sutton & Barto, 1998). However, scaling to high-dimensional problems (e.g. learning to act directly from pixels) was challenging until RL algorithms harnessed the representational power of deep neural networks (DNNs), thus catalyzing the field of deep reinforcement learning (deep RL) (Mnih et al., 2015). Three broad families of deep learning algorithms have shown promise on RL problems so far: Q-learning methods such as DQN (Mnih et al., 2015), policy gradient methods (Sehnke et al., 2010) (e.g. A3C (Mnih et al., 2016), TRPO (Schulman et al., 2015), PPO (Schulman et al., 2017)), and more recently evolution strategies (ES) (Salimans et al., 2017).

Deep Q-learning algorithms approximate the optimal Q function with DNNs, yielding policies that, for a given state, choose the action that maximizes the Q-value (Watkins & Dayan, 1992; Mnih et al., 2015; Hessel et al., 2017). Policy gradient methods directly learn the parameters of a DNN policy that outputs the probability of taking each action in each state. A team from OpenAI recently experimented with a simplified version of Natural Evolution Strategies (Wierstra et al., 2008), specifically one that learns the mean of a distribution of parameters, but not its variance. They found that this algorithm, which we will refer to simply as evolution strategies (ES), is competitive with DQN and A3C on difficult RL benchmark problems, with much faster training times (i.e. faster wall-clock time when many CPUs are available) due to better parallelization (Salimans et al., 2017).

All of these methods can be considered gradient-based methods, as they all calculate or approximate gradients in a DNN and optimize those parameters via stochastic gradient descent/ascent (although they do not require differentiating through the reward function, such as a simulator). DQN calculates the gradient of the loss of the DNN Q-value function approximator via backpropagation. Policy gradients sample behaviors stochastically from the current policy and then reinforce those that perform well via stochastic gradient ascent. ES does not calculate gradients analytically, but instead approximates the gradient of the reward function in the parameter space (Salimans et al., 2017; Wierstra et al., 2008).

Here we test whether a truly gradient-free method – a simple GA – can perform well on challenging deep RL tasks. We find GAs perform surprisingly well and thus can be considered a new addition to the set of algorithms for deep RL problems. We first describe our genetic algorithm and other contemporary methods, and then report the experimental results that led to our conclusions.

3. Methods

The next sections describe the methods used in this paper's experiments, namely, the GA that is applied across all experiments, and the novelty search algorithm with which the GA is combined in one set of experiments.

3.1. Genetic Algorithm

We purposefully test with an extremely simple GA to set a baseline for how well gradient-free evolutionary algorithms work for RL problems. We expect future work to reveal that adding the legion of enhancements that exist for GAs (Fogel & Stayton, 1994; Haupt & Haupt, 2004; Caponetto et al., 2003; Clune et al., 2011; Mouret & Doncieux, 2009; Lehman & Stanley, 2011b; Pugh et al., 2016; Stanley et al.,

2009; Stanley, 2007; Mouret & Clune, 2015) will improve their performance on deep RL tasks.

A genetic algorithm (Holland, 1992; Eiben et al., 2003) evolves a population $\mathcal{P}$ of N individuals (here, neural network parameter vectors θ, often called *genotypes*). At every *generation*, each θ_i is evaluated, producing a *fitness* score (aka reward) $F(\theta_i)$. Our GA variant performs *truncation selection*, wherein the top T individuals become the parents of the next generation. To produce the next generation, the following process is repeated $N-1$ times: A parent is selected uniformly at random with replacement and is *mutated* by applying additive Gaussian noise to the parameter vector: $\theta' = \theta + \sigma\epsilon$ where $\epsilon \sim \mathcal{N}(0, I)$. The appropriate value of σ was determined empirically for each experiment, as described in Supplementary Information (SI) Table 3. The N^{th} individual is an unmodified copy of the best individual from the previous generation, a technique called *elitism*. Historically, GAs often involve *crossover* (i.e. combining parameters from multiple parents to produce an offspring), but for simplicity we did not include it. The new population is then evaluated and the process repeats for G generations or until some other stopping criterion is met. Algorithm 1 outlines pseudo-code for this approach.

We plan to release open source code and parameter configurations for all of our experiments as soon as possible. Hyperparameters were fixed for all Atari games, chosen once at the outset before any experimentation based on our intuitions for which ones would work well for this architecture size. The results on Atari would thus likely improve with different hyperparameters. We did run an extensive hyperparameter search for the Humanoid Locomotion task. The hyperparameters for all experiments are listed in SI.

Algorithm 1 Simple Genetic Algorithm

Input: mutation power σ, population size N, number of selected individuals T, policy initialization routine ϕ.
for $g = 1, 2...G$ generations **do**
  for $i = 1, ..., N$ in next generation's population **do**
    if $g = 1$ **then**
      $\mathcal{P}_i^g = \phi(\mathcal{N}(0, I))$ {initialize random DNN}
      $F_i^g = F(\mathcal{P}_i^g)$ {assess its fitness}
    else
      if $i = 1$ **then**
        $\mathcal{P}_i^g = \mathcal{P}_i^{g-1}; F_i^g = F_i^{g-1}$ {copy the elite}
      else
        $k =$ uniformRandom$(1, T)$ {select parent}
        Sample $\epsilon \sim \mathcal{N}(0, I)$
        $\mathcal{P}_i^g = \mathcal{P}_k^{g-1} + \sigma\epsilon$ {mutate parent}
        $F_i^g = F(\mathcal{P}_i^g)$ {assess its fitness}
  Sort $\mathcal{P}^g$ and F^g with descending order by F^g
Return: highest performing policy, $\mathcal{P}_1^g$

GA implementations traditionally store each individual as a parameter vector θ, but this approach scales poorly in memory and network transmission costs with large populations and parameter vectors (e.g. deeper and wider neural networks). We propose a novel method to store large parameter vectors compactly by representing each parameter vector as an initialization seed plus the list of random seeds that produce the series of mutations applied to θ. This innovation was essential to enabling GAs to work at the scale of deep neural networks, and we thus call it a Deep GA. This technique also has the benefit of offering a state-of-the-art compression method (Section 5).

One motivation for choosing ES versus Q-learning and policy gradient methods is its faster wall-clock time with distributed computation, owing to better parallelization (Salimans et al., 2017). We found that the Deep GA not only preserves this benefit, but slightly improves upon it. The GA is faster than ES for two main reasons: (1) for every generation, ES must calculate how to update its neural network parameter vector θ. It does so via a weighted average across many (10,000 in Salimans et al. 2017) *pseudo-offspring* (random θ perturbations) weighted by their fitness. This averaging operation is slow for large neural networks and large numbers of pseudo-offspring (the latter is required for healthy optimization), and is not required for the Deep GA. (2) The ES requires require virtual batch normalization to generate diverse policies amongst the pseudo-offspring, which is necessary for accurate finite difference approximation (Salimans et al., 2016). Virtual batch normalization requires additional forward passes for a reference batch–a random set of observations chosen at the start of training–to compute layer normalization statistics that are then used in the same manner as batch normalization (Ioffe & Szegedy, 2015). We found that the random GA parameter perturbations generate sufficiently diverse policies without virtual batch normalization and thus avoid these additional forward passes through the network.

3.2. Novelty Search

One benefit of training deep neural networks with simple GAs is that doing so enables us to immediately take advantage of algorithms previously developed in the neuroevolution community. As a demonstration, we experiment with novelty search (NS) (Lehman & Stanley, 2011a), which was designed for deceptive domains in which reward-based optimization mechanisms converge to local optima. NS avoids these local optima by ignoring the reward function during evolution and instead rewarding agents for performing behaviors that have never been performed before (i.e. that are novel). Surprisingly, it can often outperform algorithms that utilize the reward signal, a result demonstrated on maze navigation and simulated biped locomotion tasks (Lehman & Stanley, 2011a). Here we apply NS to see

how it performs when combined with DNNs on a deceptive image-based RL problem (that we call the *Image Hard Maze*). We refer to the GA that optimizes for novelty as GA-NS.

NS requires a behavior characteristic (BC) that describes the behavior of a policy $BC(\pi)$ and a behavioral distance function between the BCs of any two policies: $\text{dist}(BC(\pi_i), BC(\pi_j))$. After each generation, members of the population have a probability p (here, 0.01) of having their BC stored in an *archive*. The novelty of a policy is defined as the average distance to the k (here, 25) nearest neighbors (sorted by behavioral distance) in the population or archive. Novel individuals are thus determined based on their behavioral distance to current or previously seen individuals. The GA otherwise proceeds as normal, substituting novelty for fitness (reward). For reporting and plotting purposes only, we identify the individual with the highest reward per generation. The algorithm is presented in SI Sec. 8.

4. Experiments

Our experiments focus on the performance of the GA on the same challenging problems that have validated the effectiveness of state-of-the-art deep RL algorithms and ES (Salimans et al., 2017). They include learning to play Atari directly from pixels (Mnih et al., 2015; Schulman et al., 2017; Mnih et al., 2016; Bellemare et al., 2013) and a continuous control problem involving a simulated humanoid robot learning to walk (Brockman et al., 2016; Schulman et al., 2017; Salimans et al., 2017; Todorov et al., 2012). We also tested on an Atari-scale maze domain that has a clear local optimum (Image Hard Maze) to study how well these algorithms avoid deception (Lehman & Stanley, 2011a).

For all our experiments we record the best agent found in each of multiple, independent, randomly initialized GA runs: 5 for the Atari domains, 5 for Humanoid Locomotion, and 10 for the Image Hard Maze. During evolution, each agent is evaluated n times (n=1 for Atari and Image Hard Maze domains and n=5 for the Humanoid Locomotion domain) and fitness is the mean reward. However, because false positives can arise with so few evaluations, for reporting (only) at every generation, the individual with highest mean fitness (the *elite*) is re-evaluated 30 times to better estimate its true mean fitness. The highest 30-evaluation reward across all generations is considered the final reward of an individual run. For each treatment, we then report the median value across runs of those final per-run reward values.

4.1. Atari

Training deep neural networks to play Atari – mapping directly from pixels to actions – was a celebrated feat that arguably launched the deep RL era and expanded our understanding of the difficulty of RL domains that machine learning could tackle (Mnih et al., 2015). Here we test how the performance of DNNs evolved by a simple GA compare to DNNs trained by the major families of deep RL algorithms and ES. Due to limited computational resources, we compare results on 13 Atari games. Some were chosen because they are games on which ES performs well (Frostbite, Gravitar, Kangaroo, Venture, Zaxxon) or poorly (Amidar, Enduro, Skiing, Seaquest) and the remaining games were chosen from the ALE (Bellemare et al., 2013) set in alphabetical order (Assault, Asterix, Asteroids, Atlantis). To facilitate comparisons with results reported in Salimans et al. (2017), we keep the number of game frames agents experience over the course of a GA run constant (at one billion frames). The frame limit results in a differing number of generations per independent GA run (SI Sec. Table 2), as policies of different quality in different runs may see more frames in some games (e.g. if the agent lives longer).

During training, each agent is evaluated on a full episode (capped at 20k frames), which can include multiple lives, and fitness is the final episode reward. The following are identical to DQN (Mnih et al., 2015): (1) data preprocessing, (2) network architecture, and (3) the stochastic environment that starts each episode with up to 30 random, initial no-op operations. We use the larger DQN architecture from Mnih et al. (2015) consisting of 3 convolutional layers with 32, 64, and 64 channels followed by a hidden layer with 512 units. The convolutional layers use 8×8, 4×4, and 3×3 filters with strides of 4, 2, and 1, respectively. All hidden layers were followed by a rectifier nonlinearity (ReLU). The network contains over 4M parameters.

Comparing our results with those from other algorithms fairly is extremely difficult, as such comparisons are inherently apples and oranges in many different ways. One important consideration is whether agents are evaluated on random starts (a random number of no-op actions), which is the regime they are trained on, or starts randomly sampled from human play, which tests for generalization (Nair et al., 2015). Because we do not have a database of human starts to sample from, our agents are evaluated with random starts. Where possible, we compare our results to those for other algorithms for which such random start results are available. That is true for DQN and ES, but not true for A3C, where we had to include results on human starts.

We also attempt to control for the number of frames seen during training, but because DQN is far slower to run, we present results from the literature that train on fewer frames

(200M, which requires 7-10 days of computation vs. hours of computation needed for ES and the GA to train on 1B frames). There are many variants of DQN that we could compare to, including the Rainbow (Hessel et al., 2017) algorithm that combines many different recent improvements to DQN (Van Hasselt et al., 2016; Wang et al., 2015; Schaul et al., 2015; Sutton & Barto, 1998; Bellemare et al., 2017; Fortunato et al., 2017). However, we choose to compare the GA to the original, vanilla DQN algorithm, partly because we also introduce a vanilla GA, without the many modifications and improvements that have been previously developed (Haupt & Haupt, 2004).

In what will likely be a surprise to many, the simple GA is able to train deep neural networks to play many Atari games roughly as well as DQN, A3C, and ES (Table 1). Among the 13 games we tried, DQN, ES, and the GA each produced the best score on 3 games, while A3C produced the best score on 4. On Skiing, the GA produced a score higher than any other algorithm to date that we are aware of, including all the DQN variants in the Rainbow DQN paper (Hessel et al., 2017). On some games, the GA performance advantage over DQN, A3C, and ES is considerable (e.g. Frostbite, Venture, Skiing). Videos of policies evolved by the GA can be viewed here: `https://goo.gl/QBHDJ9`. In a head-to-head comparisons on these 13 games, the GA performs better on 6 vs. ES, 6 vs. A3C, and 5 vs. DQN.

There are also many games in which the GA performs worse, continuing a theme in deep RL where different families of algorithms perform differently across different domains (Salimans et al., 2017). We note that all such comparisons are preliminary because we did not have the computational resources to gather sufficient sample sizes (and test on enough games) to see if the algorithms are significantly different; instead the key takeaway is that they all tend to perform roughly similarly in that each does well on different games.

Because performance did not plateau in the GA runs, we test whether the GA improves further given additional computation. We thus run the GA four times longer (4B frames) and in all games but one, its score improves (Table 1). With these post-4B-frame scores, the GA outperforms each of the other algorithms (A3C, ES, and DQN) on 7 of the 13 games in head-to-head comparisons. In most games, the GA's performance still has not converged at 4B frames, leaving open the question of to how well the GA will ultimately perform when run even longer. To our knowledge, this 4M+ parameter neural network is the largest neural network ever evolved with a simple GA.

One remarkable fact is how quickly the GA finds high-performing individuals. Because we employ a large population size (5,000), each run lasts relatively few generations (min 72, max 409, SI Sec. Table 2). In fact, in many games, the GA finds a solution better than DQN in only one or tens of generations! Specifically, the median GA performance is higher than the *final* DQN performance in 1, 1, 1, 29, and 58 generations for Frostbite, Venture, Skiing, Gravitar, and Kangaroo, respectively. Similar results hold for ES, where 1, 1, 3, and 14 generations of the GA were needed to obtain higher performance than ES for Frostbite, Skiing, Amidar, and Venture, respectively. The number of generations required to beat A3C were 1, 1, 1, 1, 1, 16, and 52 for Enduro, Frostbite, Kangaroo, Skiing, Venture, Gravitar, and Amidar, respectively.

Each generation, the GA tends to make small-magnitude changes to the parameter vector controlled by σ (see Methods). That the GA outperforms DQN, A3C, and ES in so few generations – especially when it does so in the first generation (which is before a round of selection) – suggests that many high-quality policies exist near the origin (to be precise, in or near the region in which the random initialization function generates policies). That raises the question: is the GA doing anything more than random search?

To answer this question, we evaluate many policies randomly generated by the GA's initialization function ϕ and report the best score. We gave random search approximately the same amount of frames and computation as the GA and compared their performance (Table 1). In every case, the GA significantly outperformed random search (Fig. 1, $p < 0.05$, this and all future p values are via a Wilcoxon rank-sum test). The improved performance suggests the GA is performing healthy optimization over generations.

Surprisingly, given how celebrated and impressive DQN, ES and A3C are, random search actually outperforms DQN on 4 out of 13 games (Asteroids, Frostbite, Skiing, & Venture), ES on 3 (Amidar, Frostbite, & Skiing), and A3C on 5 (Enduro, Frostbite, Kangaroo, Skiing, & Venture). Interestingly, some of these policies produced by random search are not trivial, degenerate policies. Instead, they appear quite sophisticated. Consider the following example from the game Frostbite, which requires an agent to perform a long sequence of jumps up and down rows of icebergs moving in different directions (while avoiding enemies and optionally collecting food) to build an igloo brick by brick (Fig. 2). Only after the igloo is built can the agent enter the igloo to receive a large payoff. Over its first two lives, a policy found by random search completes a series of 17 actions, jumping down 4 rows of icebergs moving in different directions (while avoiding enemies) and back up again three times to construct an igloo. Then, only once the igloo is built, the agent immediately moves towards it and enters it, at which point it gets a large reward. It then repeats the entire process on a harder level, this time

	DQN	Evolution Strategies	Random Search	GA	GA	A3C
Frames, Time	200M, ∼7-10d	1B, ∼ 1h	1B, ∼ 1h	1B, ∼ 1h	4B, ∼ 4h	1.28B, 4d
Forward Passes	450M	250M	250M	250M	1B	960M
Backward Passes	400M	0	0	0	0	640M
Operations	1.25B U	250M U	250M U	250M U	1B U	2.24B U
Amidar	**978**	112	151	216	294	264
Assault	4,280	1,674	642	819	1,006	**5,475**
Asterix	4,359	1,440	1,175	1,885	2,392	**22,140**
Asteroids	1,365	1,562	1,404	2,056	2,056	**4,475**
Atlantis	279,987	**1,267,410**	45,257	79,793	125,883	911,091
Enduro	**729**	95	32	39	50	-82
Frostbite	797	370	1,379	**4,801**	**5,623**	191
Gravitar	473	**805**	290	462	637	304
Kangaroo	7,259	**11,200**	1,773	8,667	10,920	94
Seaquest	**5,861**	1,390	559	807	1,241	2,355
Skiing	-13,062	-15,442	-8,816	**-6,995**	**-6,522**	-10,911
Venture	163	760	547	**810**	**1,093**	23
Zaxxon	5,363	6,380	2,943	5,183	6,827	**24,622**

Table 1. **The Atari results reveal a simple genetic algorithm is competitive with Q-learning (DQN), policy gradients (A3C), and evolution strategies (ES).** Shown are game scores (higher is better). Comparing performance between algorithms is inherently challenging (see main text), but we attempt to facilitate comparisons by showing estimates for the amount of computation (*operations*, the sum of forward and backward neural network passes), data efficiency (the number of game frames from training episodes), and how long in wall-clock time the algorithm takes to run. The GA, DQN, and ES, perform best on 3 games each, while A3C wins on 4 games. Surprisingly, random search often finds policies superior to those of DQN, A3C, and ES (see text for discussion). Note the dramatic differences in the speeds of the algorithm, which are much faster for the GA and ES, and data efficiency, which favors DQN. The scores for DQN are from Hessel et al. (2017) while those for A3C and ES are from Salimans et al. (2017). For A3C, DQN, and ES, we cannot provide error bars because they were not reported in the original literature; GA and random search error bars are visualized in (Fig. 1). The wall-clock times are approximate because they depend on a variety of hard-to-control-for factors. We found the GA runs slightly faster than ES on average.

also gathering food and thus earning bonus points (video: `https://youtu.be/CGHgENV1hII`). That policy resulted in a very high score of 3,620 in less than 1 hour of random search, vs. an average score of 797 produced by DQN after 7-10 days of optimization. One may think that random search found a lucky open loop sequence of actions overfit to that particular stochastic environment. Remarkably, we found that this policy actually generalizes to other initial conditions too, achieving a median score of 3,170 (with 95% bootstrapped median confidence intervals of 2,580 - 3,170) on 200 different test environments (each with up to 30 random initial no-ops, a standard testing procedure (Hessel et al., 2017; Mnih et al., 2015)).

These examples and the success of RS versus DQN, A3C, and ES suggest that many Atari games that seem hard based on the low performance of leading deep RL algorithms may not be as hard as we think, and instead that these algorithms for some reason are performing extremely poorly on tasks that are actually quite easy. They further suggest that sometimes the best search strategy is not to follow the gradient, but instead to conduct a dense search in a local neighborhood and select the best point found, a subject we return to in the discussion (Sec. 6).

4.2. Humanoid Locomotion

We next test the GA on a challenging continuous control problem, specifically humanoid locomotion (Fig. 3a). We test with the MuJoCo Humanoid-v1 environment in OpenAI Gym (Todorov et al., 2012; Brockman et al., 2016), which involves a simulated humanoid robot learning to walk. Solving this problem has validated modern, powerful algorithms such as A3C (Mnih et al., 2016), TRPO (Schulman et al., 2015), and ES (Salimans et al., 2017).

This problem involves mapping a vector of 376 scalars that describe the state of the humanoid (e.g. its position, velocity, angle) to 17 joint torques. The robot receives a scalar reward that is a combination of four components each timestep. It gets positive reward for standing and its velocity in the positive x direction, and negative reward the more energy it expends and for how hard it impacts the ground. These four terms are summed over every timestep in an episode to calculate the total reward.

To stabilize training, we normalize each dimension of the

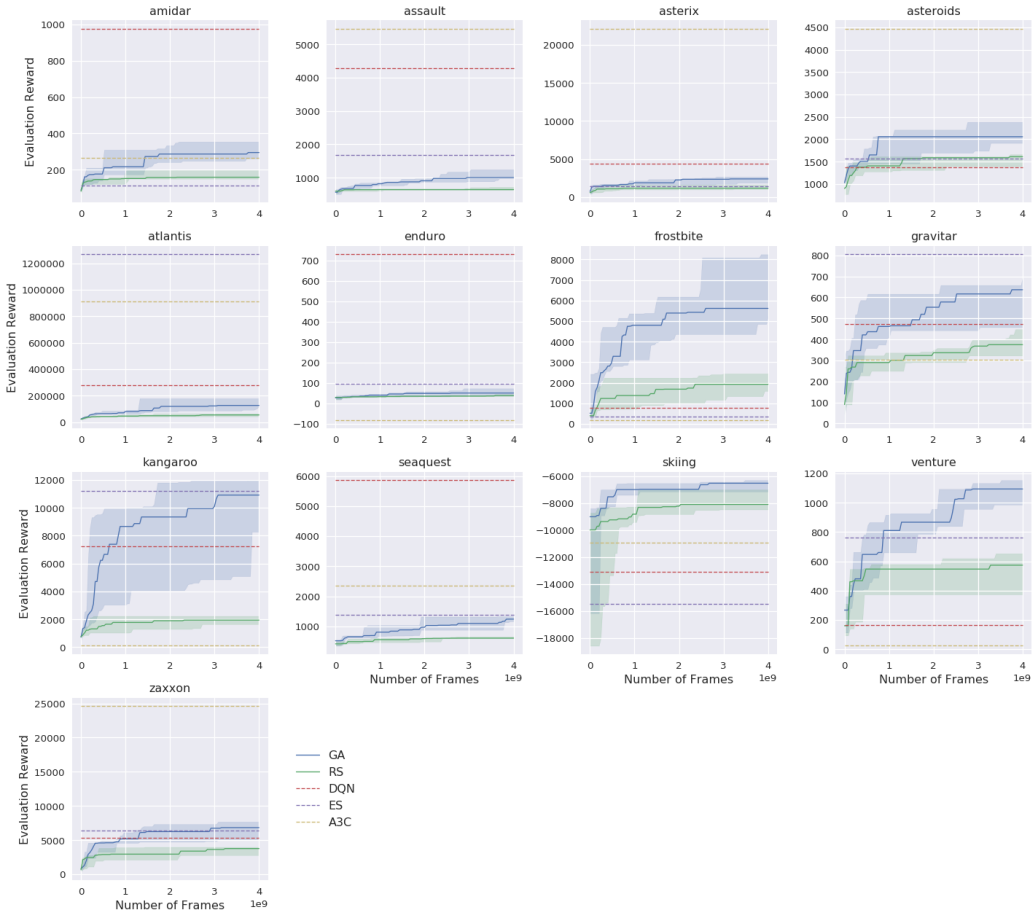


Figure 1. **GA and random search performance across generations on Atari 2600 games.** The GA significantly outperforms random search in every game ($p < 0.05$). The performance of the GA and random search to DQN, A3C, and ES depends on the game. We plot final scores (as dashed lines) for DQN, A3C, and ES because we do not have their performance values across training and because they trained on different numbers of game frames (see Table 1). For GA and RS, we report the median and 95% bootstrapped confidence intervals of the median across 5 experiments of the best mean evaluation score (over 30 stochastic rollouts) seen up to that point in training.

input space separately by subtracting the mean and dividing by the variance of data for that input gathered in the domain by 10,000 random policies. We also applied annealing to the mutation power σ, decreasing it to 0.001 after 1,000 generations, which resulted in a small performance boost at the end of training.

The architecture has two 256-unit hidden layers with tanh activation functions. This architecture is the one in the configuration file included in the source code released by Salimans et al. (2017). The architecture described in their paper is similar, but smaller, having 64 neurons per layer (Salimans et al., 2017). Although relatively shallow by deep learning standards, and much smaller than the Atari DNNs, this architecture still contains $\sim$167k parameters, which is orders of magnitude greater than the largest neural networks evolved for robotics tasks that we are aware of, which contained 1,560 (Huizinga et al., 2016) and before that 800 parameters (Clune et al., 2011). Many assumed evolution would fail at larger scales (e.g. networks with hundreds of thousands or millions of weights, as in this paper). Previous work has called the problem solved with a score around 6,000 (Salimans et al., 2017). The GA achieves a median above that level after $\sim$1,500 generations. However, it requires far more computation than ES to do so (ES requires $\sim$100 generations for median performance to surpass the 6,000 threshold). It is not clear why the GA requires so much more computation and hyperparameter tuning on this problem, especially given how quickly the GA found high-performing policies in the Atari domain. While the GA needs far more computation in this domain, it is interesting nevertheless that it does eventually solve it by producing an agent that can walk and score over 6,000. Considering its very fast discovery of high-

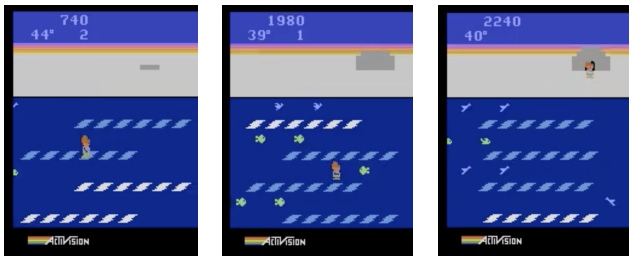
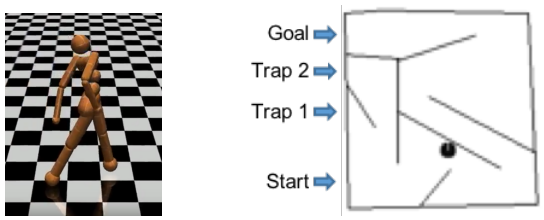


Figure 2. **Example of high-performing individual on Frostbite found through random search**. See text for a description of the behavior of this policy. Its final score is 3,620 in this episode, which is higher than the scores produced by DQN, A3C and ES, although not as high as the score found by the GA (Table 1).



(a) (b)

Figure 3. **Two different test domains.** Left: The Human Locomotion domain. The humanoid robot has to learn to walk efficiently. Shown is an example policy evolved with a GA. Right: The Image Hard Maze domain. A small wheeled robot must navigate to the goal with this bird's-eye view as pixel inputs. Shown is an example image frame as seen by the robot's neural network. The text annotations and arrows are not visible to the robot. The robot starts in the bottom left corner facing right.

performing solutions in Atari, clearly the GA's advantage versus other methods depends on the domain, and understanding this dependence is an important target for future research.

4.3. Image Hard Maze

The Hard Maze domain is a staple in the neuroevolution community, where it demonstrates the problem of local optima (aka deception) in reinforcement learning (Lehman & Stanley, 2011a). In it, a robot receives more reward the closer it gets to the goal. Specifically, a single reward is provided at the end of an episode consisting of the negative of the straight-line distance between the final position of the agent and the goal. The problem is deceptive because greedily getting closer to the goal leads an agent to permanently get stuck in one of the deceptive traps in the maze (Fig. 3b). Optimization algorithms that do not conduct sufficient exploration suffer this fate. Novelty search (NS) solves this problem easily because it ignores the reward entirely and encourages agents to visit new places, which ultimately leads to some reaching the goal (Lehman & Stanley, 2011a).

The original version of this problem involves only a few inputs (radar sensors to sense walls) and two continuous outputs, one that controls speed (forward or backward) and another that controls rotation, making it solvable by small neural networks (on the order of tens of connections). Because here we want to demonstrate the benefits of NS at the scale of deep neural networks, we introduce a new version of the domain called Image Hard Maze. Like many Atari games, it shows a bird's-eye view of the world to the agent in the form of an 84×84 pixel image. This change makes the problem easier in some ways (e.g. now it is fully observable), but harder because it is much higher-dimensional: the neural network must learn to process this pixel input and take actions. For temporal context, the current frame and previous three frames are all input at each timestep, following Mnih et al. (2015). An example frame is shown in Fig. 3b. The outputs remain the same as in the original problem formulation.

Following previous work in the original Hard Maze (Lehman & Stanley, 2011a), the BC is the (x, y) position of the robot at the end of the episode (400 timesteps), and the behavioral distance function is the squared Euclidean distance between these final (x, y) positions. Both the environment and the agent are deterministic. The simple simulator rejects forward or backward motion that results in the robot penetrating walls (i.e. the position of the robot remains unchanged from the previous timestep in such cases); these dynamics prohibit a robot from sliding along a wall, although rotational motor commands still have their usual effect in such situations.

We confirm that the results that held for small neural networks on the original, radar-based version of this task also hold for the high-dimensional, visual version of this task with deep neural networks. With a 4M+ parameter network processing pixels, the GA-based novelty search (GA-NS) is able to solve the task by finding the goal (Fig. 4). The GA optimizes for reward only and, as expected, gets stuck in the local optima of Trap 2 (Fig. 5) and thus fails to solve the problem (Fig. 4), significantly underperforming GA-NS ($p < 0.001$). These results thus confirm that intuitions gained in small neural networks about local optima in reward functions hold for much larger, deep neural networks. While local optima may not be a problem with deep neural networks in supervised learning where the correct answer is always given (Pascanu et al., 2014), the same is not true in reinforcement learning problems with sparse or deceptive rewards. Our results confirm that we are able to use exploration methods such as novelty search to solve this sort of deception, even in high-dimensional problems such as those involving learning directly from pixels.

This is the largest neural network optimized by novelty

search to date by three orders of magnitude. In an companion paper (Conti et al., 2017), we also demonstrate a similar finding, by hybridizing novelty search with ES to create NS-ES, and show that it too can help deep neural networks avoid deception in challenging RL benchmark domains. We hope these results encourage more investigation into combining deep neural networks with novelty search and similar methods, such as quality diversity algorithms, which seek to collect a set of high-performing, yet interestingly different policies (Lehman & Stanley, 2011b; Cully et al., 2015; Pugh et al., 2016).

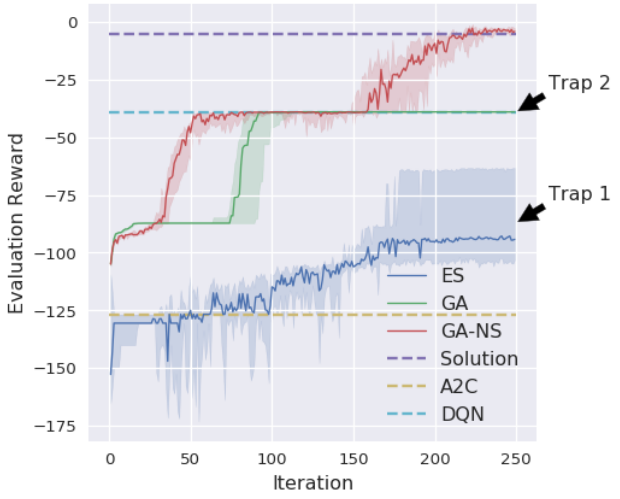


Figure 4. **Image Hard Maze results reveal that novelty search can train deep neural networks to avoid local optima that stymie other algorithms.** The GA, which solely optimizes for reward and has no incentive to explore, gets stuck on the local optimum of Trap 2 (the goal and traps are visualized in Fig. 3b). The GA optimizing for novelty (GA-NS) is encouraged to ignore reward and explore the whole map, enabling it to eventually find the goal. ES performs even worse than the GA, as discussed in the main text. DQN and A2C also fail to solve this task. For ES, the performance of the mean θ policy each iteration is plotted. For GA and GA-NS, the performance of the highest-scoring individual per generation is plotted. Because DQN and A2C do not have the same number of evaluations per iteration as the evolutionary algorithms, we plot their final median reward as dashed lines. Fig. 5 shows the behavior of these algorithms during training.

As expected, ES also fails to solve the task because it focuses solely on maximizing reward (Fig. 5). It is surprising, however, that it significantly underperforms the GA ($p < 0.001$). In 8 of 10 runs it gets stuck near Trap 1, not because of deception, but instead seemingly because it cannot reliably learn to pass through a small bottleneck corridor. This phenomenon has never been observed with population-based GAs, suggesting the ES (at least with these hyperparameters) is qualitatively different than GAs in this regard (Lehman et al., 2017). We believe this difference occurs because ES optimizes for the average reward of the population sampled from a probability distribution. Even if the maximum fitness of agents sampled from that distribution is higher further along a corridor, ES will not move in that direction if the average population is lower (e.g. if other policies sampled from the distribution crash into the walls, or experience other low-reward fates). In a companion paper, we investigate this interesting difference between ES and GAs (Lehman et al., 2017). Note, however, that even when ES moved through this bottleneck (2 out of 10 runs), because it is solely reward-driven, it got stuck in Trap 2.

We also test Q-learning (DQN) and policy gradients on this problem. We did not have source code for A3C, but were able to obtain source code for A2C, which has similar performance (Wu et al., 2017): the only difference (explaining why it has one fewer 'A') is that it is synchronous instead of asynchronous. For these experiments we modified the rewards of the domain to step-by-step rewards (the negative change in distance to goal since the last time-step), but for plotting purposes, we record the final distance to the goal. Having per-step rewards is more standard for these algorithms and gives them more information, but does not remove the deception. Because DQN requires discrete outputs, for it we discretize each of the two continuous outputs into to five equally sized bins. Because it needs to be able to specify all possible output combinations, it thus learns $5^2 = 25$ Q-values.

Also as expected, DQN and A2C fail to solve this problem (Fig. 4, Fig. 5). Their default exploration mechanisms are not enough to find the global optimum given the deceptive reward function in this domain. DQN is drawn into the expected Trap 2. For reasons that are not clear to us, even though A2C visits Trap 2 frequently early in training, it converges on getting stuck in a different part of the maze.

Overall the results for the Image Hard Maze domain confirm that the Deep GA allows algorithms developed for small-scale neural networks to operate at DNN scale, and can thus be harnessed on hard, high-dimensional problems that require DNNs. In future work, it will be interesting to test the benefits that novelty search provides when combined with a GA on more domains, including Atari and robotics domains. More importantly, our demonstration suggests that other algorithms that enhance GAs can now be combined with DNNs. Perhaps most promising are those that combine a notion of diversity (e.g. novelty) and quality (i.e. being high performing) (Mouret & Clune, 2015; Mouret & Doncieux, 2009; Lehman & Stanley, 2011b; Cully et al., 2015; Pugh et al., 2016).

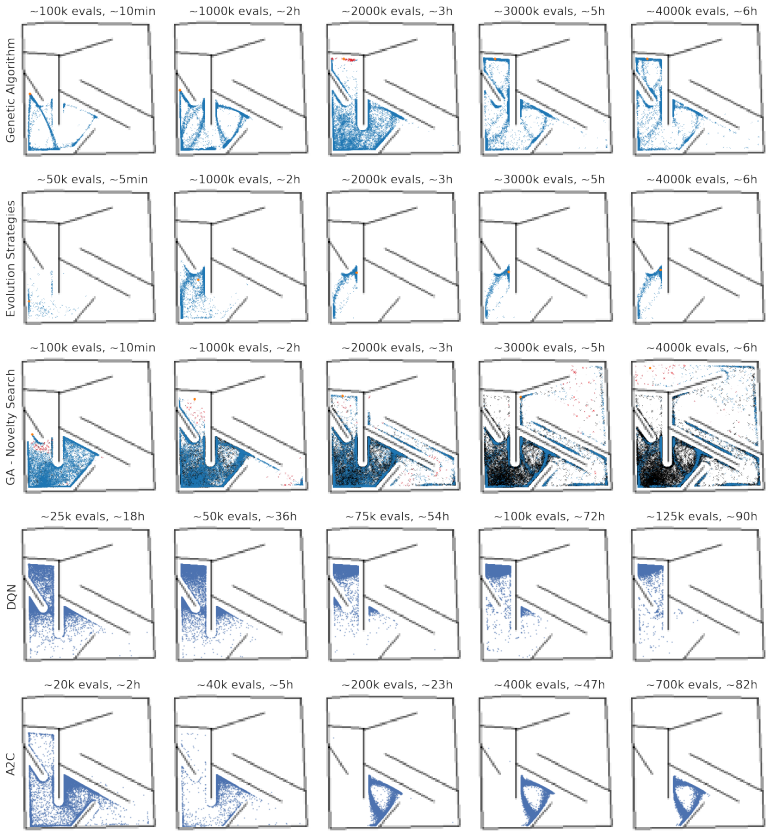


Figure 5. **How different algorithms explore the deceptive Image Hard Maze over time.** Traditional reward-maximization algorithms do not exhibit sufficient exploration to avoid the local optimum (going up). In contrast, a GA optimizing for novelty only (GA-NS) explores the entire environment and ultimately finds the goal. For the evolutionary algorithms (GA-NS, GA, ES), blue crosses represent the population (pseudo-offspring for ES), red crosses represent the top T GA offspring, orange dots represent the final positions of GA elites and the current mean ES policy, and the black crosses are entries in the GA-NS archive. All 3 evolutionary algorithms had the same number of evaluations, but ES and the GA have many overlapping points because they revisit locations due to poor exploration, giving the illusion of fewer evaluations. For DQN and A2C, we plot the end-of-episode position of the agent for each of the 20K episodes prior to the checkpoint listed above the plot.

5. Compact network encoding

As mentioned before, the Deep GA method enables compactly storing DNNs by representing them as the series of mutations required to reconstruct their parameters. This technique is advantageous because the size of the compressed parameters increases with the number of generations instead of with the size of the network (the latter is often much larger than the former). For example, as previously discussed, we were able to evolve competitive Atari-playing agents in some games in as little as tens of generations, enabling us to compress the representation of a 4M+ parameter neural network to just thousands of bytes (a factor of 10,000-fold smaller). As far as we know, this represents the state of the art in encoding large networks compactly. However, it does not count as a general network compression technique because it cannot take an arbitrary network and compress it, and instead only works for networks evolved with a GA. Of course, one could harness this approach to create a general compression technique by evolving a network to match a target network, which is an interesting area for future research.

The compressibility of a network is entirely dependent on the number of mutations needed to achieve a certain performance. For Humanoid Locomotion, this translates to encoding 160,000 parameters in just 6kB (27-fold compression). This amount was the largest number of mutations needed for any of the networks we evolved. All of the solutions on the Image Hard Maze domain could be represented with 1kB or less (16,000 times smaller). The Atari compression benefits change depending on the game due variance in generations between experiments (Table 2), but are always substantial: all Atari final networks were compressible to 300-2,000 bytes (8,000-50,000-fold compression). It does, of course, require computation to reconstruct

the DNN weight vector from this compact encoding.

6. Discussion

The surprising success of the GA in domains thought to require at least some degree of gradient estimation suggests some heretofore under-appreciated aspects of high-dimensional search spaces. The random search results suggest that densely sampling in a region around the origin is sufficient in some cases to find far better solutions than those found by state-of-the-art, gradient-based methods even with far more computation or wall-clock time, suggesting that gradients do not point to these solutions, or that other optimization issues interfere with finding them, such as saddle points or noisy gradient estimates. The GA results further suggest that sampling in the region around good solutions is often sufficient to find even better solutions, and that a sequence of such discoveries is possible in many challenging domains. That result in turn implies that the distribution of solutions of increasing quality is unexpectedly dense, and that you do not need to follow a gradient to find them.

Another, non-mutually exclusive hypothesis, is that GAs have improved performance due to *temporally extended exploration* (Osband et al., 2016). That means they explore consistently since all actions in an episode are a function of the same set of mutated parameters, which has been shown to improve exploration (Plappert et al., 2017). Such consistency helps with exploration for two reasons, (1) an agent takes the same action (or has the same distribution over actions) each time it visits the same state, which makes it easier to learn whether the policy in that state is advantageous, and (2) the agent is also more likely to have correlated actions across states (e.g. always go up) because mutations to its internal representations can affect the actions taken in many states similarly.

Perhaps more interesting is the result that sometimes it is actually *worse* to follow the gradient than sample locally in the parameter space for better solutions. This scenario probably does not hold in all domains, or even in all the regions of a domain where it sometimes holds, but that it holds at all expands our conceptual understanding of the viability of different kinds of search operators. A reason GA might outperform gradient-based methods is if local optima are present, as it can jump over them in the parameter space, whereas a gradient method cannot (without additional optimization tricks such as momentum, although we note that ES utilized the modern ADAM optimizer in these experiments (Kingma & Ba, 2014), which includes momentum). One unknown question is whether GA-style local, gradient-free search is better early on in the search process, but switching to a gradient-based search later allows further progress that would be impossible, or prohibitively computationally expensive, for a GA to make. Another unknown question is the promise of simultaneously hybridizing GA methods with modern algorithms for deep RL, such as Q-learning, policy gradients, or evolution strategies.

We emphasize that we still know very little about the ultimate promise of GAs versus competing algorithms for training deep neural networks on reinforcement learning problems. On the Atari domain, we did not perform hyperparameter search, so the best GA results could be much higher as it is well known that hyperparameters can have massive effects on the performance of optimization algorithms, including GAs (Haupt & Haupt, 2004; Clune et al., 2008). We also have not yet seen the algorithm converge in most of these domains, so its ceiling is unknown. Additionally, here we used an extremely simple GA, but many techniques have been invented to improve GA performance (Eiben et al., 2003; Haupt & Haupt, 2004), including crossover (Holland, 1992; Deb & Myburgh, 2016), indirect encoding (Stanley, 2007; Stanley et al., 2009; Clune et al., 2011), and encouraging diversity (Lehman & Stanley, 2011a; Mouret & Clune, 2015; Pugh et al., 2016) (a subject we do take an initial look at here with our novelty search experiments), just to name a few. Moreover, many techniques have been invented that dramatically improve the training of DNNs with backpropagation, such as residual networks (He et al., 2015), virtual batch normalization (Salimans et al., 2016), SELU or RELU activation functions (Krizhevsky et al., 2012; Klambauer et al., 2017), LSTMs or GRUs (Hochreiter & Schmidhuber, 1997; Cho et al., 2014), regularization (Hoerl & Kennard, 1970), dropout (Srivastava et al., 2014), and annealing learning rate schedules (Robbins & Monro, 1951). We hypothesize that many of these techniques will also improve neuroevolution for large DNNs, a subject we are currently investigating.

It is also possible that some of these enhancements may remedy the poor data efficiency the GA showed on the Humanoid Locomotion problem. For example, indirect encoding, which allows genomic parameters to affect multiple weights in the final neural network (in a way similar to convolution's tied weights, but with far more flexibility), has been shown to dramatically improve performance and data efficiency when evolving robot gaits (Clune et al., 2011). Those results were found with the HyperNEAT algorithm (Stanley et al., 2009), which has an indirect encoding that abstracts the power of developmental biology (Stanley, 2007), and is a particularly promising direction for Humanoid Locomotion and Atari that we will investigate in future work. More generally, it will be interesting to learn on which domains Deep GA tends to perform well or poorly and understand why. For example, GAs could perform well in other non-differentiable domains, such as architecture search (Liu et al., 2017; Miikkulainen et al.,

2017) and for training limited precision (including binary) neural networks.

Finally, it is worth noting that the GA (like ES before it) benefits greatly from large-scale parallel computation in these studies. In our experiments, each GA run was distributed across hundreds or thousands of CPUs, depending on available computation. That the availability of such resources so significantly changes what is possible with such a simple algorithm motivates further investment in large-scale parallel computing infrastructure. While the dependence of the results on hundreds or thousands of CPUs in parallel could be viewed as an obstacle for some, it could also be interpreted as an exciting opportunity: as prices continue to decline and the availability of such resources becomes more mainstream, more and more researchers will have the opportunity to investigate an entirely new paradigm of opportunities, not unlike the transformation GPUs have enabled in deep learning.

7. Conclusion

Our work introduces a Deep GA, which involves a simple parallelization trick that allows us to train deep neural networks with GAs. We then document that GAs are surprisingly competitive with popular algorithms for deep reinforcement learning problems, such as DQN, A3C, and ES, especially in the challenging Atari domain. We also showed that interesting algorithms developed in the neuroevolution community can now immediately be tested with deep neural networks, by showing that a Deep GA-powered novelty search can solve a deceptive Atari-scale game. It will be interesting to see future research investigate the potential and limits of GAs, especially when combined with other techniques known to improve GA performance. More generally, our results continue the story – started by backprop and extended with ES – that old, simple algorithms plus modern amounts of computation can perform amazingly well. That raises the question of what other old algorithms should be revisited.

Acknowledgements

We thank all of the members of Uber AI Labs for helpful suggestions throughout the course of this work, in particular Zoubin Ghahramani, Peter Dayan, Noah Goodman, Thomas Miconi, and Theofanis Karaletsos. We also thank Justin Pinkul, Mike Deats, Cody Yancey, and the entire OpusStack Team at Uber for providing resources and technical support.

References

Bellemare, Marc G, Naddaf, Yavar, Veness, Joel, and Bowling, Michael. The arcade learning environment: An evaluation platform for general agents. *J. Artif. Intell. Res.(JAIR)*, 47:253–279, 2013.

Bellemare, Marc G, Dabney, Will, and Munos, Rémi. A distributional perspective on reinforcement learning. *arXiv preprint arXiv:1707.06887*, 2017.

Brockman, Greg, Cheung, Vicki, Pettersson, Ludwig, Schneider, Jonas, Schulman, John, Tang, Jie, and Zaremba, Wojciech. OpenAI gym, 2016.

Caponetto, Riccardo, Fortuna, Luigi, Fazzino, Stefano, and Xibilia, Maria Gabriella. Chaotic sequences to improve the performance of evolutionary algorithms. *IEEE transactions on evolutionary computation*, 7(3): 289–304, 2003.

Cho, Kyunghyun, Van Merriënboer, Bart, Bahdanau, Dzmitry, and Bengio, Yoshua. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

Clune, Jeff, Misevic, Dusan, Ofria, Charles, Lenski, Richard E, Elena, Santiago F, and Sanjuán, Rafael. Natural selection fails to optimize mutation rates for long-term adaptation on rugged fitness landscapes. *PLoS Computational Biology*, 4(9):e1000187, 2008.

Clune, Jeff, Stanley, Kenneth O., Pennock, Robert T., and Ofria, Charles. On the performance of indirect encoding across the continuum of regularity. *IEEE Transactions on Evolutionary Computation*, 2011.

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

Cully, A., Clune, J., Tarapore, D., and Mouret, J.-B. Robots that can adapt like animals. *Nature*, 521:503–507, 2015. doi: 10.1038/nature14422.

Deb, Kalyanmoy and Myburgh, Christie. Breaking the billion-variable barrier in real-world optimization using a customized evolutionary algorithm. In *GECCO '16*, pp. 653–660. ACM, 2016.

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

Eiben, Agoston E, Smith, James E, et al. *Introduction to evolutionary computing*, volume 53. Springer, 2003.

Fogel, David B and Stayton, Lauren C. On the effectiveness of crossover in simulated evolutionary optimization. *BioSystems*, 32(3):171–182, 1994.

Fortunato, Meire, Azar, Mohammad Gheshlaghi, Piot, Bilal, Menick, Jacob, Osband, Ian, Graves, Alex, Mnih, Vlad, Munos, Remi, Hassabis, Demis, Pietquin, Olivier, et al. Noisy networks for exploration. *arXiv preprint arXiv:1706.10295*, 2017.

Glorot, Xavier and Bengio, Yoshua. Understanding the difficulty of training deep feedforward neural networks. In *ICAI*, pp. 249–256, 2010.

Haupt, Randy L and Haupt, Sue Ellen. *Practical genetic algorithms*. John Wiley & Sons, 2004.

He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

Hessel, Matteo, Modayil, Joseph, Van Hasselt, Hado, Schaul, Tom, Ostrovski, Georg, Dabney, Will, Horgan, Dan, Piot, Bilal, Azar, Mohammad, and Silver, David. Rainbow: Combining improvements in deep reinforcement learning. *arXiv preprint arXiv:1710.02298*, 2017.

Hochreiter, Sepp and Schmidhuber, Jürgen. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Hoerl, Arthur E and Kennard, Robert W. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.

Holland, John H. Genetic algorithms. *Scientific american*, 267(1):66–73, 1992.

Huizinga, Joost, Mouret, Jean-Baptiste, and Clune, Jeff. Does aligning phenotypic and genotypic modularity improve the evolution of neural networks? In *GECCO '16*, pp. 125–132. ACM, 2016.

Ioffe, Sergey and Szegedy, Christian. Batch normalization: Accelerating deep network training by reducing internal covariate shift. ICML'15, pp. 448–456. JMLR.org, 2015.

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

Klambauer, Günter, Unterthiner, Thomas, Mayr, Andreas, and Hochreiter, Sepp. Self-normalizing neural networks. *arXiv preprint arXiv:1706.02515*, 2017.

Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. Imagenet classification with deep convolutional neural networks. In *NIPS*, pp. 1097–1105, 2012.

Lehman, Joel and Stanley, Kenneth O. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary Computation*, 19(2):189–223, 2011a.

Lehman, Joel and Stanley, Kenneth O. Evolving a diversity of virtual creatures through novelty search and local competition. In *GECCO '11: Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pp. 211–218, Dublin, Ireland, 12-16 July 2011b. ACM. ISBN 978-1-4503-0557-0. doi: doi:10.1145/2001576.2001606.

Lehman, Joel, Chen, Jay, Clune, Jeff, and Stanley, Kenneth O. ES is more than just a traditional finite-difference approximator. *arXiv preprint to appear*, 2017.

Liu, Hanxiao, Simonyan, Karen, Vinyals, Oriol, Fernando, Chrisantha, and Kavukcuoglu, Koray. Hierarchical representations for efficient architecture search. *arXiv preprint arXiv:1711.00436*, 2017.

Miikkulainen, Risto, Liang, Jason, Meyerson, Elliot, Rawal, Aditya, Fink, Dan, Francon, Olivier, Raju, Bala, Navruzyan, Arshak, Duffy, Nigel, and Hodjat, Babak. Evolving deep neural networks. *arXiv preprint arXiv:1703.00548*, 2017.

Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Rusu, Andrei A, Veness, Joel, Bellemare, Marc G, Graves, Alex, Riedmiller, Martin, Fidjeland, Andreas K, Ostrovski, Georg, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

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

Mouret, Jean-Baptiste and Clune, Jeff. Illuminating search spaces by mapping elites. *ArXiv e-prints*, abs/1504.04909, 2015.

Mouret, Jean-Baptiste and Doncieux, Stephane. Overcoming the bootstrap problem in evolutionary robotics using behavioral diversity. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC-2009)*, pp. 1161–1168. IEEE, 2009.

Nair, Arun, Srinivasan, Praveen, Blackwell, Sam, Alcicek, Cagdas, Fearon, Rory, De Maria, Alessandro, Panneershelvam, Vedavyas, Suleyman, Mustafa, Beattie, Charles, Petersen, Stig, et al. Massively parallel methods for deep reinforcement learning. *arXiv preprint arXiv:1507.04296*, 2015.

Osband, Ian, Blundell, Charles, Pritzel, Alexander, and Van Roy, Benjamin. Deep exploration via bootstrapped dqn. In *NIPS*, pp. 4026–4034, 2016.

Pascanu, Razvan, Dauphin, Yann N, Ganguli, Surya, and Bengio, Yoshua. On the saddle point problem for non-convex optimization. *arXiv preprint arXiv:1405.4604*, 2014.

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

Pugh, Justin K, Soros, Lisa B., and Stanley, Kenneth O. Quality diversity: A new frontier for evolutionary computation. 3(40), 2016. ISSN 2296-9144.

Robbins, Herbert and Monro, Sutton. A stochastic approximation method. *The annals of mathematical statistics*, pp. 400–407, 1951.

Salimans, T., Ho, J., Chen, X., Sidor, S., and Sutskever, I. Evolution Strategies as a Scalable Alternative to Reinforcement Learning. *ArXiv e-prints*, 1703.03864, March 2017.

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

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

Schaul, Tom, Quan, John, Antonoglou, Ioannis, and Silver, David. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

Schulman, John, Levine, Sergey, Abbeel, Pieter, Jordan, Michael, and Moritz, Philipp. Trust region policy optimization. In *ICML '15*, pp. 1889–1897, 2015.

Schulman, John, Wolski, Filip, Dhariwal, Prafulla, Radford, Alec, and Klimov, Oleg. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Sehnke, Frank, Osendorfer, Christian, Rückstieß, Thomas, Graves, Alex, Peters, Jan, and Schmidhuber, Jürgen. Parameter-exploring policy gradients. *Neural Networks*, 23(4):551–559, 2010.

Seide, Frank, Li, Gang, and Yu, Dong. Conversational speech transcription using context-dependent deep neural networks. In *Interspeech 2011*. International Speech Communication Association, August 2011.

Srivastava, Nitish, Hinton, Geoffrey, Krizhevsky, Alex, Sutskever, Ilya, and Salakhutdinov, Ruslan. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1): 1929–1958, 2014.

Stanley, Kenneth O. Compositional pattern producing networks: A novel abstraction of development. *Genetic Programming and Evolvable Machines Special Issue on Developmental Systems*, 8(2):131–162, 2007.

Stanley, Kenneth O., D'Ambrosio, David B., and Gauci, Jason. A hypercube-based indirect encoding for evolving large-scale neural networks. *Artificial Life*, 15(2):185–212, 2009.

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

Todorov, Emanuel, Erez, Tom, and Tassa, Yuval. 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.

Van Hasselt, Hado, Guez, Arthur, and Silver, David. Deep reinforcement learning with double q-learning. In *AAAI*, pp. 2094–2100, 2016.

Wang, Ziyu, Schaul, Tom, Hessel, Matteo, Van Hasselt, Hado, Lanctot, Marc, and De Freitas, Nando. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*, 2015.

Watkins, Christopher JCH and Dayan, Peter. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

Wierstra, Daan, Schaul, Tom, Peters, Jan, and Schmidhuber, Juergen. Natural evolution strategies. In *Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence). IEEE Congress on*, pp. 3381–3387. IEEE, 2008.

Williams, Ronald J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

Wu, Yuhuai, Mansimov, Elman, Grosse, Roger B, Liao, Shun, and Ba, Jimmy. Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. In *NIPS*, pp. 5285–5294, 2017.

8. Supplementary Information

8.1. Hyperparameters

The policy initialization function ϕ generates an initial parameter vector. Bias weights for each neuron are set to

zero, and connection weights are drawn from a standard normal distribution; these weights are then rescaled so that the vector of incoming weights for each neuron has unit magnitude. This procedure is known as normalized column weight initialization and the implementation here is taken from the OpenAI baselines package (Dhariwal et al., 2017). Without this procedure, the variance of the distribution of a neuron's activation depends on the number of its inputs, which can complicate optimization of DNNs (Glorot & Bengio, 2010). Other principled initialization rules could likely be substituted in its place.

Game	Minimum Generations	Median Generations	Maximum Generations
Amidar	226	258	269
Enduro	121	121	121
Frostbite	348	409	494
Gravitar	283	292	304
Kangaroo	242	253	322
Seaquest	145	167	171
Skiing	81	86	88
Venture	71	72	75
Zaxxon	335	342	349

Table 2. **The number of generations the GA needed to reach 4B frames.**

Hyperparameter	Humanoid Locomotion	Image Hard Maze	Atari
Population Size (N)	12,500+1	20,000+1	5,000+1
Mutation power (σ)	0.00224	0.005	0.005
Truncation Size (T)	625	61	10
Number of Trials	5	1	1
Archive Probability		0.01	

Table 3. **Hyperparameters.** Population sizes are incremented to account for elites (+1). Many of the unusual numbers were found via preliminary hyperparameter searches in other domains.

Algorithm 2 Novelty Search (GA-NS)

Input: mutation power σ, population size N, number of individuals selected to reproduce per generation T, policy initialization routine ϕ, empty archive $\mathcal{A}$, archive insertion probability p
for $g = 1, 2...G$ generations **do**
  for $i = 1, ..., N$ in next generation's population **do**
    if $g = 1$ **then**
      $\mathcal{P}_i^g = \phi(\mathcal{N}(0, I))$ {Initialize random DNN}
      $BC_i^g = BC(\mathcal{P}_i^g)$
      $F_i^g = F(\mathcal{P}_i^g)$
    else
      if $i = 1$ **then**
        $\mathcal{P}_i^g = \mathcal{P}_i^{g-1}; F_i^g = F_i^{g-1}$ {Copy most novel}
        $BC_i^g = BC_i^{g-1}$
      else
        $k = \text{uniformRandom}(1, T)$ {Select parent}
        Sample $\epsilon \sim \mathcal{N}(0, I)$
        $\mathcal{P}_i^g = \mathcal{P}_k^{g-1} + \sigma\epsilon$ {mutate parent}
        $BC_i^g = BC(\mathcal{P}_i^g)$
        $F_i^g = F(\mathcal{P}_i^g)$
  for $i = 1, ..., N$ in next generation population **do**
    $\mathcal{N}_i^g = dist(BC_i^g, \mathcal{A} \cup BC^g)$
    Insert BC_i^g into $\mathcal{A}$ with probability p
  Sort ($\mathcal{P}^g$, BC^g, F^g) in descending order by $\mathcal{N}^g$
Return: highest performing policy