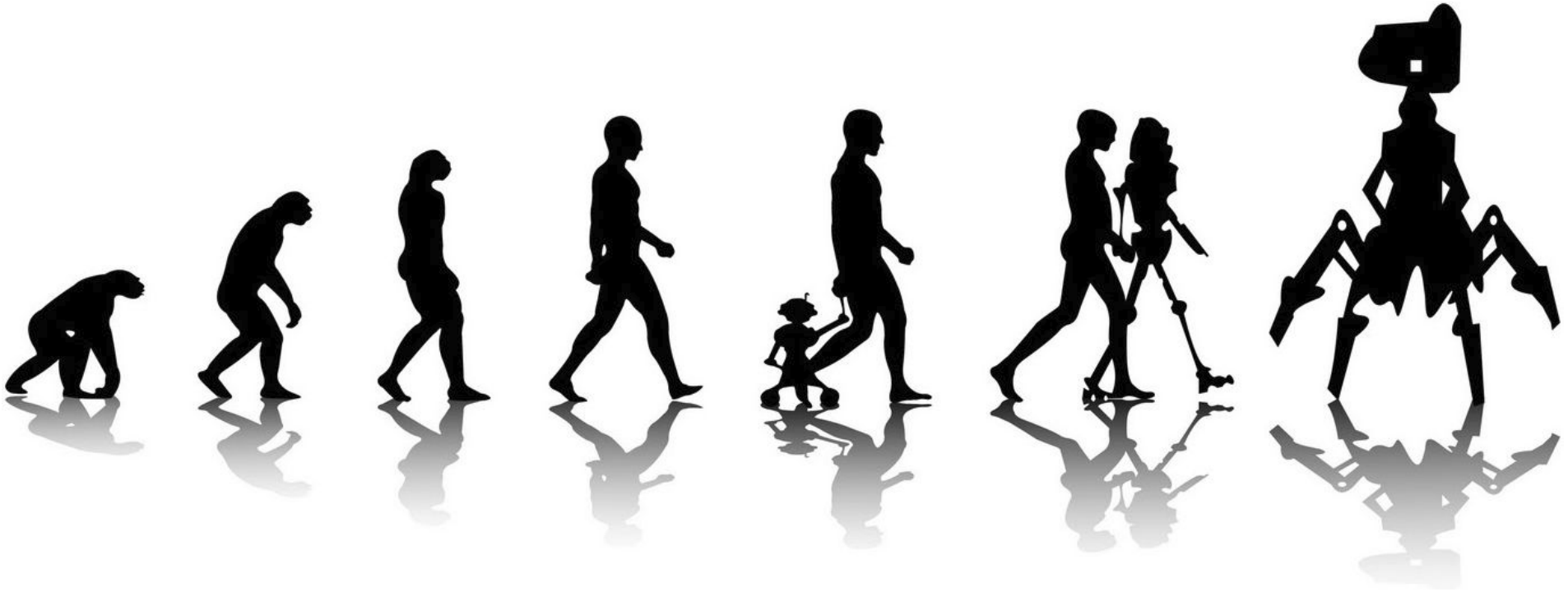


Modern Robotics: Evolutionary Robotics

COSC 4560 / COSC 5560

Professor Cheney

4/25/18

Emergence of Locomotion Behaviours in Rich Environments

Nicolas Heess, Dhruva TB, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, S. M. Ali Eslami, Martin Riedmiller, David Silver
DeepMind

Algorithm 1 Proximal Policy Optimization (adapted from [8])

for $i \in \{1, \cdots, N\}$ **do**
  Run policy π_θ for T timesteps, collecting $\{s_t, a_t, r_t\}$
  Estimate advantages $\hat{A}_t = \sum_{t'>t} \gamma^{t'-t} r_{t'} - V_\phi(s_t)$
  $\pi_{\text{old}} \leftarrow \pi_\theta$
  for $j \in \{1, \cdots, M\}$ **do**
    $J_{PPO}(\theta) = \sum_{t=1}^{T} \frac{\pi_\theta(a_t|s_t)}{\pi_{old}(a_t|s_t)} \hat{A}_t - \lambda \mathrm{KL}[\pi_{old}|\pi_\theta]$
    Update θ by a gradient method w.r.t. $J_{PPO}(\theta)$
  end for
  for $j \in \{1, \cdots, B\}$ **do**
    $L_{BL}(\phi) = -\sum_{t=1}^{T} (\sum_{t'>t} \gamma^{t'-t} r_{t'} - V_\phi(s_t))^2$
    Update ϕ by a gradient method w.r.t. $L_{BL}(\phi)$
  end for
  if $\mathrm{KL}[\pi_{old}|\pi_\theta] > \beta_{high} \mathrm{KL}_{target}$ **then**
    $\lambda \leftarrow \alpha\lambda$
  else if $\mathrm{KL}[\pi_{old}|\pi_\theta] < \beta_{low} \mathrm{KL}_{target}$ **then**
    $\lambda \leftarrow \lambda/\alpha$
  end if
end for

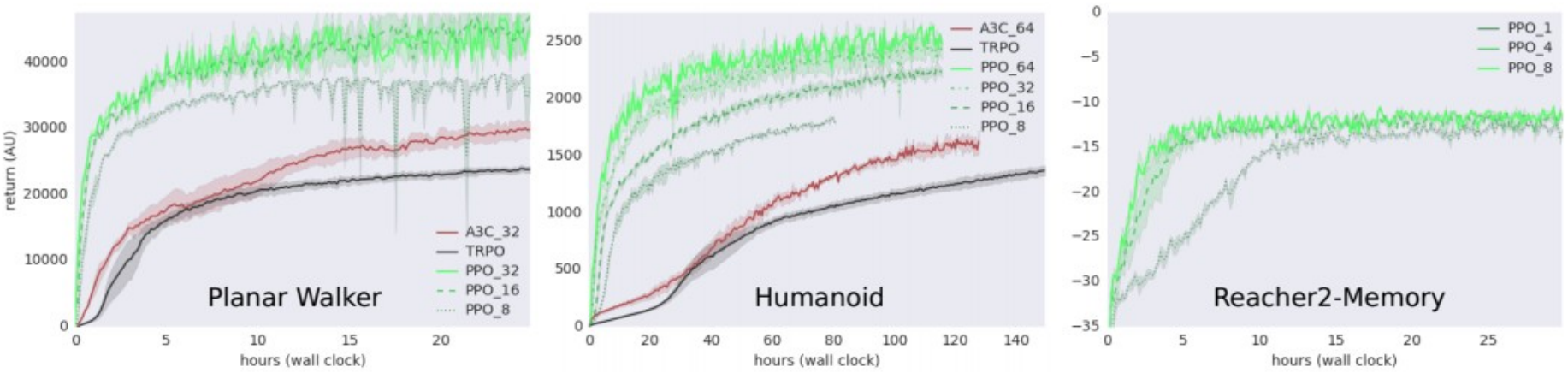


Figure 1: **DPPO benchmark performance** on the *Planar Walker* (*left*), *Humanoid* (*middle*), and *Memory Reacher* (*right*) tasks. In all cases, DPPO achieves performance equivalent to TRPO, and scales well with the number of workers used. The *Memory Reacher* task demonstrates that it can be used with recurrent networks.

Bodies We consider three different torque-controlled bodies, described roughly in terms of increasing complexity. *Planar walker*: a simple walking body with 9 DoF and 6 actuated joints constrained to the plane. *Quadruped*: a simple three-dimensional quadrupedal body with 12 DoF and 8 actuated joints. *Humanoid*: a three-dimensional humanoid with 21 actuated dimensions and 28 DoF. The bodies can be seen in action in figures 4, 5, and 7 respectively. Note that the *Planar walker* and *Humanoid* bodies overlap with those used in the benchmarking tasks described in the previous section, however the benchmark tasks only consisted of simple locomotion in an open plane.

Rewards We keep the reward for all tasks simple and consistent across terrains. The reward consists of a main component proportional to the velocity along the x-axis, encouraging the agent to make forward progress along the track, plus a small term penalizing torques. For the walker the reward also includes the same box constraints on the pose as in section 2. For the quadruped and humanoid we penalize deviations from the center of the track, and the humanoid receives an additional reward per time-step for not falling. Details can be found in the supplemental material. We note that differences in the reward functions across bodies are the consequence of us adapting previously proposed reward functions (cf. e.g. [12, 18]) rather than the result of careful tuning, and while the reward functions vary slightly across bodies we do not change them to elicit different behaviors for a single body.

Terrain and obstacles All of our courses are procedurally generated; in every episode a new course is generated based on pre-defined statistics. We consider several different terrain and obstacle types: (a) *hurdles*: hurdle-like obstacles of variable height and width that the walker needs to jump or climb over; (b) *gaps*: gaps in the ground that must be jumped over; (c) *variable terrain*: a terrain with different features such as ramps, gaps, hills, etc.; (d) *slalom walls*: walls that form obstacles that require walking around, (e) *platforms*: platforms that hover above the ground which can be jumped on or crouched under. Courses consist of a sequence of random instantiations of the above terrain types within user-specified parameter ranges.

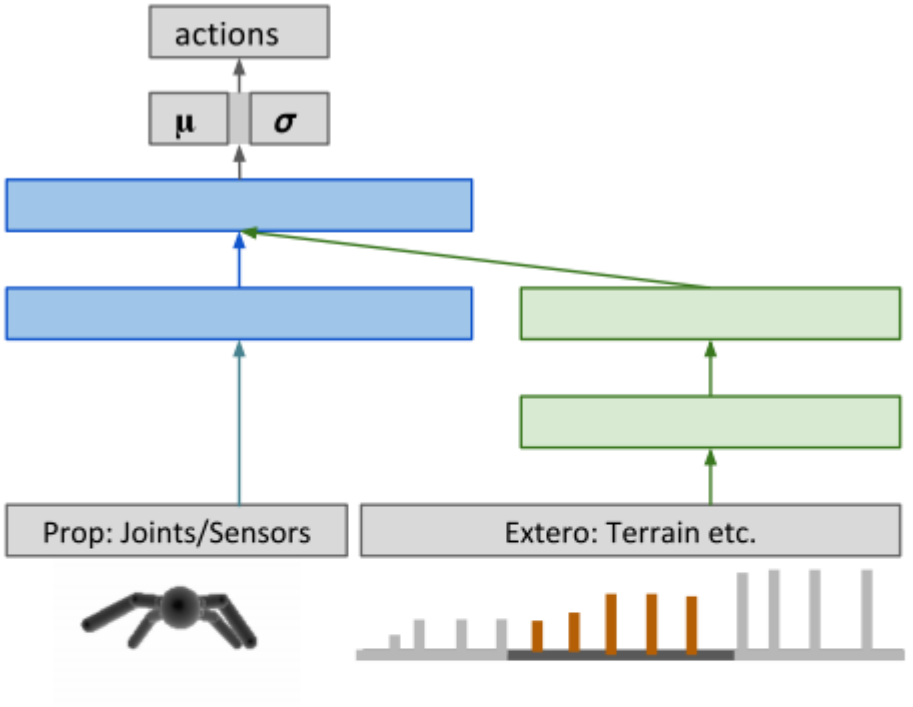


Figure 2: Schematic of the network architecture. We use an architecture similar to [22], consisting of a component processing information local to the controlled body (egocentric information; blue) and a modulatory component that processes environment and task related "exteroceptive" information such as the terrain shape (green).

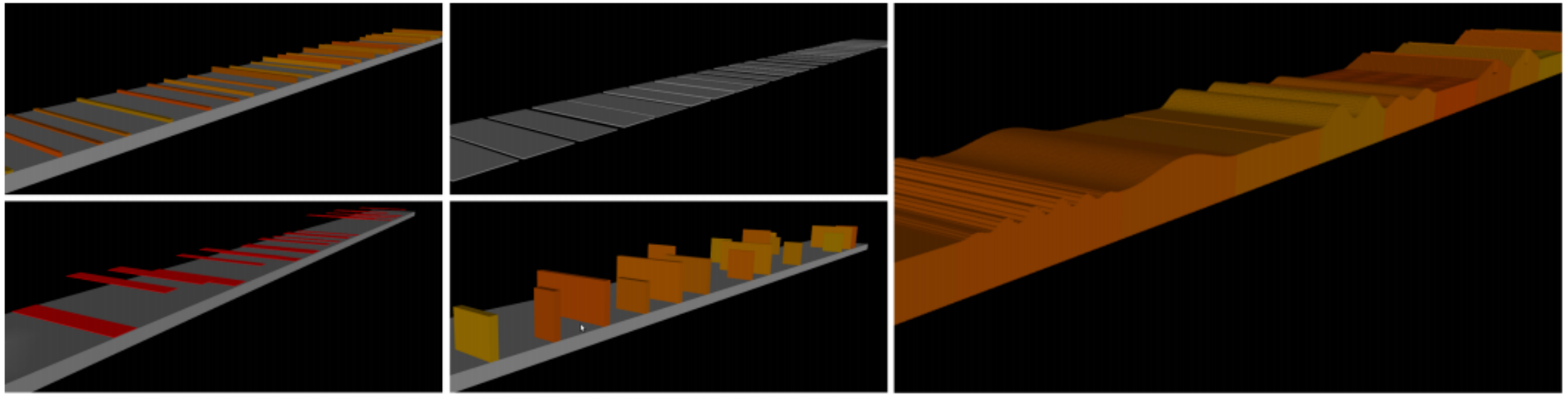


Figure 3: Examples of the terrain types used in the experiments. Left to right and top to bottom: *hurdles*, *platforms*, *gaps*, *slalom walls*, *variable terrain*.

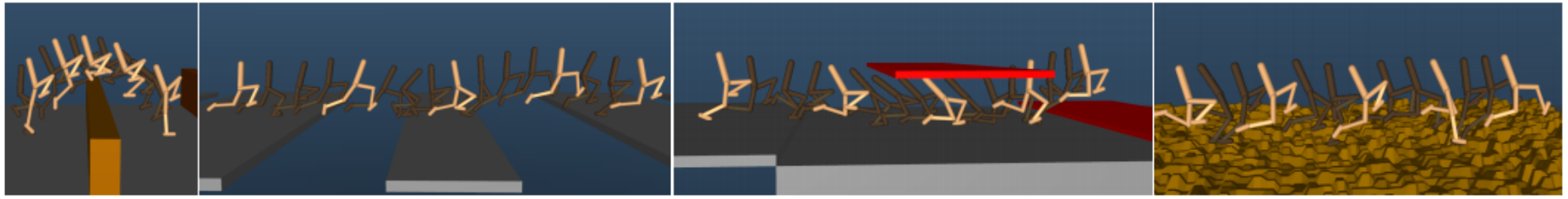


Figure 4: *Walker skills*: Time-lapse images of a representative *Planar Walker* policy traversing rubble; jumping over a hurdle; jumping over gaps and crouching to pass underneath a platform.

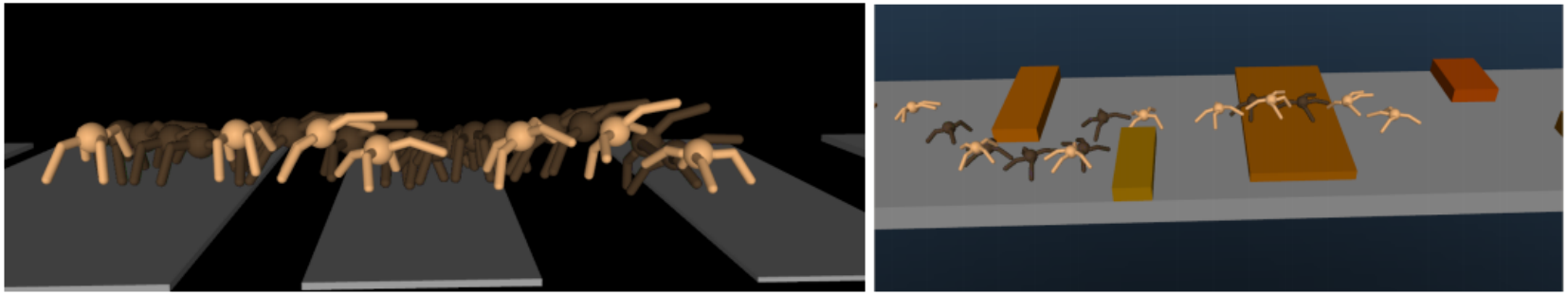


Figure 5: Time-lapse images of a representative *Quadruped* policy traversing gaps (left); and navigating obstacles (right)

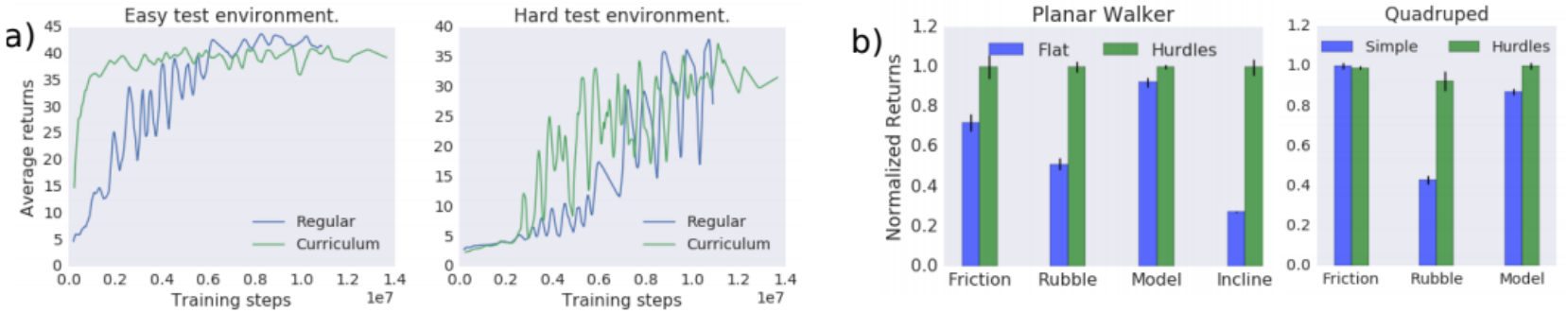


Figure 6: *a) Curriculum training*: Evaluation of policies trained on hurdle courses with different statistics: "regular" courses contain arbitrarily interleaved high and low hurdles (blue); "curriculum" courses gradually increase hurdle height over the course of the track (green). During training we evaluate both policies on validation courses with low/"easy" hurdles (left) and tall/"hard" hurdles (right). The performance of the policy trained on the curriculum courses increases faster. *b) Robustness* of *Planar Walker* policies (left) and *Quadruped* policies (right): We evaluate how training on *hurdles* (green) increases policy robustness relative to training on flat terrain (blue). Policies are assessed on courses with unobserved changes in ground friction, terrain surface (rubble), strength of the body actuators, and incline of the ground plane. There is a notable advantage in some cases for policies trained on the hurdle terrain. All plots show the average returns normalized for each terrain setting.

Emergence of Locomotion Behaviours in Rich Environments



Evolution Strategies as a Scalable Alternative to Reinforcement Learning

Tim Salimans **Jonathan Ho** **Xi Chen** **Szymon Sidor** **Ilya Sutskever**
OpenAI

Algorithm 1 Evolution Strategies

Input: Learning rate α, noise standard deviation σ, initial policy parameters θ_0
for $t = 0, 1, 2, \ldots$ **do**
    Sample $\epsilon_1, \ldots \epsilon_n \sim \mathcal{N}(0, I)$
    Compute returns $F_i = F(\theta_t + \sigma\epsilon_i)$ for $i = 1, \ldots, n$
    Set $\theta_{t+1} \leftarrow \theta_t + \alpha \frac{1}{n\sigma} \sum_{i=1}^{n} F_i \epsilon_i$
end for

Algorithm 1 Evolution Strategies

Input: Learning rate α, noise standard deviation σ, initial policy parameters θ_0
for $t = 0, 1, 2, \ldots$ **do**
  Sample $\epsilon_1, \ldots \epsilon_n \sim \mathcal{N}(0, I)$
  Compute returns $F_i = F(\theta_t + \sigma\epsilon_i)$ for $i = 1, \ldots, n$
  Set $\theta_{t+1} \leftarrow \theta_t + \alpha \frac{1}{n\sigma} \sum_{i=1}^{n} F_i \epsilon_i$
end for

Algorithm 2 Parallelized Evolution Strategies

Input: Learning rate α, noise standard deviation σ, initial policy parameters θ_0
Initialize: n workers with known random seeds, and initial parameters θ_0
for $t = 0, 1, 2, \ldots$ **do**
  for each worker $i = 1, \ldots, n$ **do**
    Sample $\epsilon_i \sim \mathcal{N}(0, I)$
    Compute returns $F_i = F(\theta_t + \sigma\epsilon_i)$
  end for
  Send all scalar returns F_i from each worker to every other worker
  for each worker $i = 1, \ldots, n$ **do**
    Reconstruct all perturbations ϵ_j for $j = 1, \ldots, n$ using known random seeds
    Set $\theta_{t+1} \leftarrow \theta_t + \alpha \frac{1}{n\sigma} \sum_{j=1}^{n} F_j \epsilon_j$
  end for
end for

4 Experiments

4.1 MuJoCo

We found that ES was able to solve these tasks up to TRPO's final performance after 5 million timesteps of environment interaction. To obtain this result, we ran ES over 6 random seeds and compared the mean learning curves to similarly computed curves for TRPO. The exact sample complexity tradeoffs over the course of learning are listed in Table 1, and detailed results are listed in Table 3 of the supplement. Generally, we were able to solve the environments in less than 10x penalty in sample complexity on the hard environments (Hopper and Walker2d) compared to TRPO. On simple environments, we achieved up to 3x better sample complexity than TRPO.

Table 1: MuJoCo tasks: Ratio of ES timesteps to TRPO timesteps needed to reach various percentages of TRPO's learning progress at 5 million timesteps.

Environment	25%	50%	75%	100%
HalfCheetah	0.15	0.49	0.42	0.58
Hopper	0.53	3.64	6.05	6.94
InvertedDoublePendulum	0.46	0.48	0.49	1.23
InvertedPendulum	0.28	0.52	0.78	0.88
Swimmer	0.56	0.47	0.53	0.30
Walker2d	0.41	5.69	8.02	7.88

Game	DQN	A3C FF, 1 day	HyperNEAT	ES FF, 1 hour	A2C FF
Amidar	133.4	283.9	184.4	112.0	**548.2**
Assault	3332.3	**3746.1**	912.6	1673.9	2026.6
Asterix	124.5	**6723.0**	2340.0	1440.0	3779.7
Asteroids	697.1	**3009.4**	1694.0	1562.0	1733.4
Atlantis	76108.0	772392.0	61260.0	1267410.0	**2872644.8**
Bank Heist	176.3	**946.0**	214.0	225.0	724.1
Battle Zone	17560.0	11340.0	**36200.0**	16600.0	8406.2
Beam Rider	8672.4	**13235.9**	1412.8	744.0	4438.9
Berzerk		**1433.4**	1394.0	686.0	720.6
Bowling	41.2	36.2	**135.8**	30.0	28.9
Boxing	25.8	33.7	16.4	49.8	**95.8**
Breakout	303.9	**551.6**	2.8	9.5	368.5
Centipede	3773.1	3306.5	**25275.2**	7783.9	2773.3
Chopper Command	3046.0	**4669.0**	3960.0	3710.0	1700.0
Crazy Climber	50992.0	**101624.0**	0.0	26430.0	100034.4
Demon Attack	12835.2	**84997.5**	14620.0	1166.5	23657.7
Double Dunk	**21.6**	0.1	2.0	0.2	3.2
Enduro	**475.6**	82.2	93.6	95.0	0.0
Fishing Derby	2.3	13.6	**49.8**	49.0	33.9
Freeway	25.8	0.1	29.0	**31.0**	0.0
Frostbite	157.4	180.1	**2260.0**	370.0	266.6
Gopher	2731.8	**8442.8**	364.0	582.0	6266.2
Gravitar	216.5	269.5	370.0	**805.0**	256.2
Ice Hockey	3.8	4.7	**10.6**	4.1	4.9
Kangaroo	2696.0	106.0	800.0	**11200.0**	1357.6
Krull	3864.0	8066.6	**12601.4**	8647.2	6411.5
Montezuma's Revenge	50.0	**53.0**	0.0	0.0	0.0
Name This Game	5439.9	5614.0	**6742.0**	4503.0	5532.8
Phoenix		**28181.8**	1762.0	4041.0	14104.7
Pit Fall		**123.0**	0.0	0.0	8.2
Pong	16.2	11.4	17.4	**21.0**	20.8
Private Eye	298.2	194.4	**10747.4**	100.0	100.0
Q*Bert	4589.8	13752.3	695.0	147.5	**15758.6**
River Raid	4065.3	**10001.2**	2616.0	5009.0	9856.9
Road Runner	9264.0	31769.0	3220.0	16590.0	**33846.9**
Robotank	**58.5**	2.3	43.8	11.9	2.2
Seaquest	**2793.9**	2300.2	716.0	1390.0	1763.7
Skiing		13700.0	7983.6	**15442.5**	15245.8
Solaris		1884.8	160.0	2090.0	**2265.0**
Space Invaders	1449.7	**2214.7**	1251.0	678.5	951.9
Star Gunner	34081.0	**64393.0**	2720.0	1470.0	40065.6
Tennis	2.3	10.2	0.0	4.5	**11.2**
Time Pilot	5640.0	5825.0	**7340.0**	4970.0	4637.5
Tutankham	32.4	26.1	23.6	130.3	**194.3**
Up and Down	3311.3	54525.4	43734.0	67974.0	**75785.9**
Venture	54.0	19.0	0.0	**760.0**	0.0
Video Pinball	20228.1	**185852.6**	0.0	22834.8	46470.1
Wizard of Wor	246.0	**5278.0**	3360.0	3480.0	1587.5
Yars Revenge		7270.8	**24096.4**	16401.7	8963.5
Zaxxon	831.0	2659.0	3000.0	**6380.0**	5.6

3.2 Problem dimensionality

The gradient estimate of ES can be interpreted as a method for randomized finite differences in high-dimensional space. Indeed, using the fact that $\mathbb{E}_{\epsilon \sim N(0,I)} \{F(\theta)\, \epsilon/\sigma\} = 0$, we get

$$\nabla_\theta \eta(\theta) = \mathbb{E}_{\epsilon \sim N(0,I)} \{F(\theta + \sigma\epsilon)\, \epsilon/\sigma\} \; = \mathbb{E}_{\epsilon \sim N(0,I)} \{(F(\theta + \sigma\epsilon) - F(\theta))\, \epsilon/\sigma\}$$

It is now apparent that ES can be seen as computing a finite difference derivative estimate in a randomly chosen direction, especially as σ becomes small. The resemblance of ES to finite differences suggests the method will scale poorly with the dimension of the parameters θ. Theoretical analysis indeed shows that for general non-smooth optimization problems, the required number of optimization steps scales linearly with the dimension [Nesterov and Spokoiny, 2011]. However, it is important to note that this does not mean that larger neural networks will perform worse than smaller networks when optimized using ES: what matters is the difficulty, or intrinsic dimension, of the optimization problem. To see that the dimensionality of our model can be completely separate from the effective dimension of the optimization problem, consider a regression problem where we approximate a univariate variable y with a linear model $\hat{y} = \mathbf{x} \cdot \mathbf{w}$: if we double the number of features and parameters in this model by concatenating $\mathbf{x}$ with itself (i.e. using features $\mathbf{x}' = (\mathbf{x}, \mathbf{x})$), the problem does not become more difficult. The ES algorithm will do exactly the same thing when applied to this higher dimensional problem, as long as we divide the standard deviation of the noise by two, as well as the learning rate.

In practice, we observe slightly better results when using larger networks with ES. For example, we tried both the larger network and smaller network used in A3C [Mnih et al., 2016] for learning Atari 2600 games, and on average obtained better results using the larger network. We hypothesize that this is due to the same effect that makes standard gradient-based optimization of large neural networks easier than for small ones: large networks have fewer local minima [Kawaguchi, 2016].

3.1 When is ES better than policy gradients?

Given these two methods of smoothing the decision problem, which should we use? The answer depends strongly on the structure of the decision problem and on which type of Monte Carlo estimator is used to estimate the gradients $\nabla_\theta F_{PG}(\theta)$ and $\nabla_\theta F_{ES}(\theta)$. Suppose the correlation between the return and the individual actions is low (as is true for any hard RL problem). Assuming we approximate these gradients using simple Monte Carlo (REINFORCE) with a good baseline on the return, we have

$$\mathrm{Var}[\nabla_\theta F_{PG}(\theta)] \approx \mathrm{Var}[R(\mathbf{a})]\,\mathrm{Var}[\nabla_\theta \log p(\mathbf{a};\theta)],$$
$$\mathrm{Var}[\nabla_\theta F_{ES}(\theta)] \approx \mathrm{Var}[R(\mathbf{a})]\,\mathrm{Var}[\nabla_\theta \log p(\tilde{\theta};\theta)].$$

If both methods perform a similar amount of exploration, $\mathrm{Var}[R(\mathbf{a})]$ will be similar for both expressions. The difference will thus be in the second term. Here we have that $\nabla_\theta \log p(\mathbf{a};\theta) = \sum_{t=1}^{T} \nabla_\theta \log p(a_t;\theta)$ is a sum of T uncorrelated terms, so that the variance of the policy gradient estimator will grow nearly linearly with T. The corresponding term for evolution strategies, $\nabla_\theta \log p(\tilde{\theta};\theta)$, is independent of T. Evolution strategies will thus have an advantage compared to policy gradients for long episodes with very many time steps. In practice, the effective number of steps T is often reduced in policy gradient methods by discounting rewards. If the effects of actions are short-lasting, this allows us to dramatically reduce the variance in our gradient estimate, and this has been critical to the success of applications such as Atari games. However, this discounting will bias our gradient estimate if actions have long lasting effects. Another strategy for reducing the effective value of T is to use value function approximation. This has also been effective, but once again runs the risk of biasing our gradient estimates. Evolution strategies is thus an attractive choice if the effective number of time steps T is long, actions have long-lasting effects, and if no good value function estimates are available.

3.3 Advantages of not calculating gradients

In addition to being easy to parallelize, and to having an advantage in cases with long action sequences and delayed rewards, black box optimization algorithms like ES have other advantages over RL techniques that calculate gradients. The communication overhead of implementing ES in a distributed setting is lower than for competing RL methods such as policy gradients and Q-learning, as the only information that needs to be communicated across processes are the scalar return and the random seed that was used to generate the perturbations ϵ, rather than a full gradient. Also, ES can deal with maximally sparse and delayed rewards; only the total return of an episode is used, whereas other methods use individual rewards and their exact timing.

By not requiring backpropagation, black box optimizers reduce the amount of computation per episode by about two thirds, and memory by potentially much more. In addition, not explicitly calculating an analytical gradient protects against problems with exploding gradients that are common when working with recurrent neural networks. By smoothing the cost function in parameter space, we reduce the pathological curvature that causes these problems: bounded cost functions that are smooth enough can't have exploding gradients. At the extreme, ES allows us to incorporate non-differentiable elements into our architecture, such as modules that use *hard attention* [Xu et al., 2015].

Black box optimization methods are uniquely suited to low precision hardware for deep learning. Low precision arithmetic, such as in binary neural networks, can be performed much cheaper than at high precision. When optimizing such low precision architectures, biased low precision gradient estimates can be a problem when using gradient-based methods. Similarly, specialized hardware for neural network inference, such as TPUs [Jouppi et al., 2017], can be used directly when performing optimization using ES, while their limited memory usually makes backpropagation impossible.

By perturbing in parameter space instead of action space, black box optimizers are naturally invariant to the frequency at which our agent acts in the environment. For MDP-based reinforcement learning algorithms, on the other hand, it is well known that *frameskip* is a crucial parameter to get right for the optimization to succeed [Braylan et al., 2005]. While this is usually a solvable problem for games that only require short-term planning and action, it is a problem for learning longer term strategic behavior. For these problems, RL needs hierarchy to succeed [Parr and Russell, 1998], which is not as necessary when using black box optimization.

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

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
        $k = \text{uniformRandom}(1, T)$ {select parent}
        Sample $\epsilon \sim \mathcal{N}(0, I)$
        $\mathcal{P}_i^g = \mathcal{P}_k^{g-1} + \sigma\epsilon$ {mutate parent}
        $F_i^g = F(\mathcal{P}_i^g)$ {assess its fitness}
  Sort $\mathcal{P}^g$ and F^g with descending order by F^g
Return: highest performing policy, $\mathcal{P}_1^g$

	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.

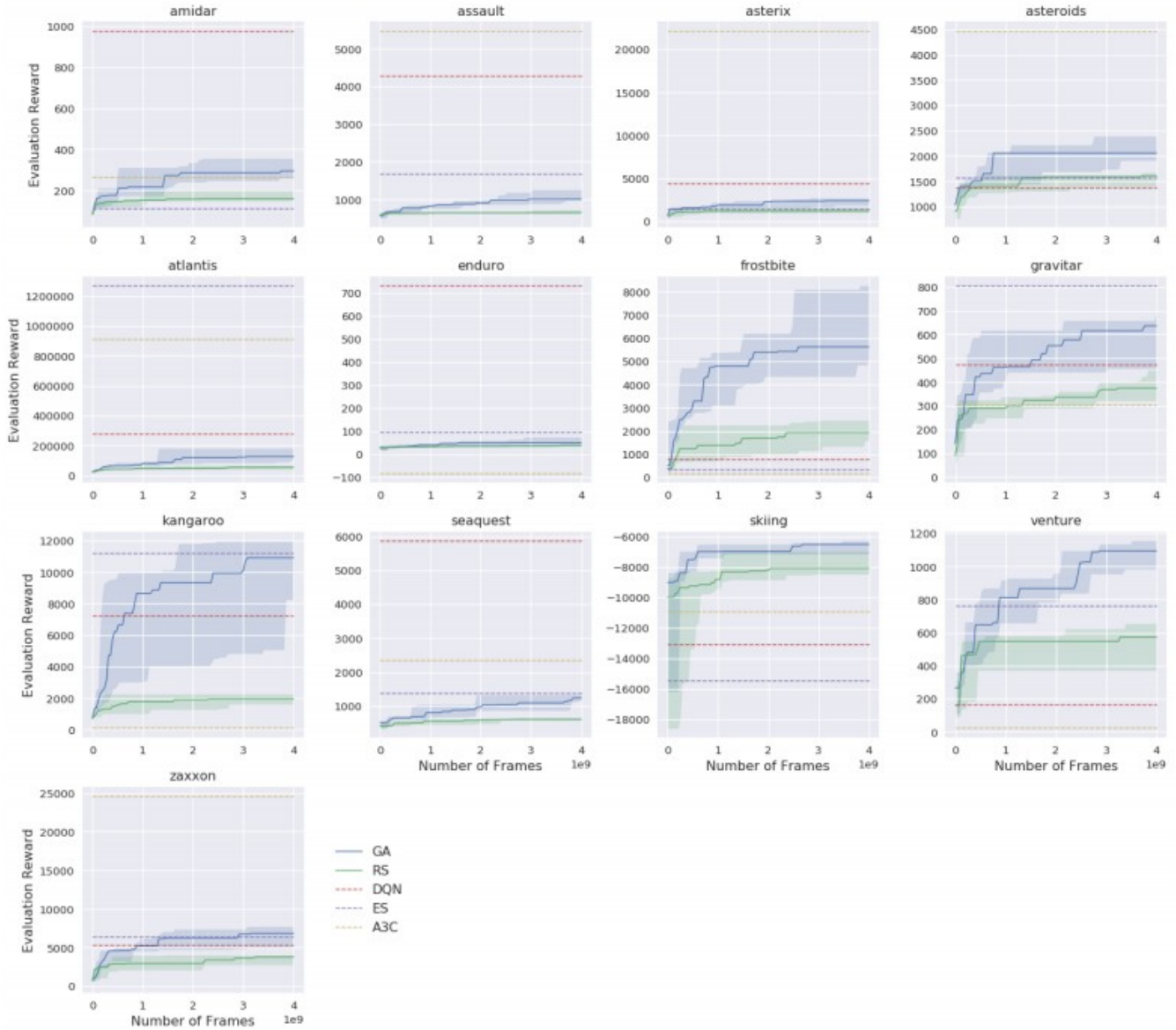
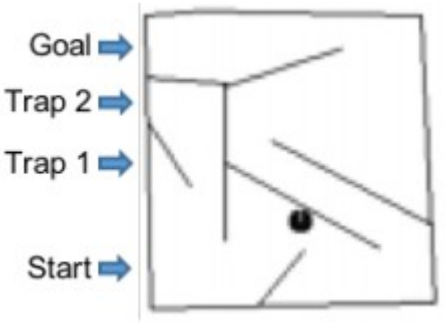


Figure 1. **GA and random search performance across generations on Atari 2600 games.** The GA significantly outperforms random search in every game ($p < 0.05$). The performance of the GA and random search to DQN, A3C, and ES depends on the game. We plot final scores (as dashed lines) for DQN, A3C, and 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.

3.2. Novelty Search

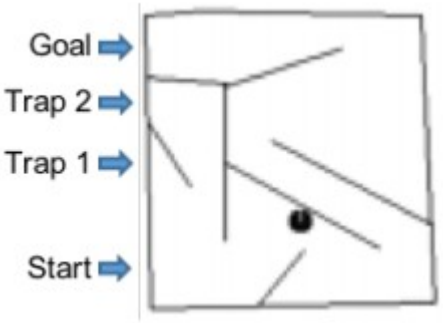
4.3. Image Hard Maze



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.

3.2. Novelty Search

4.3. Image Hard Maze



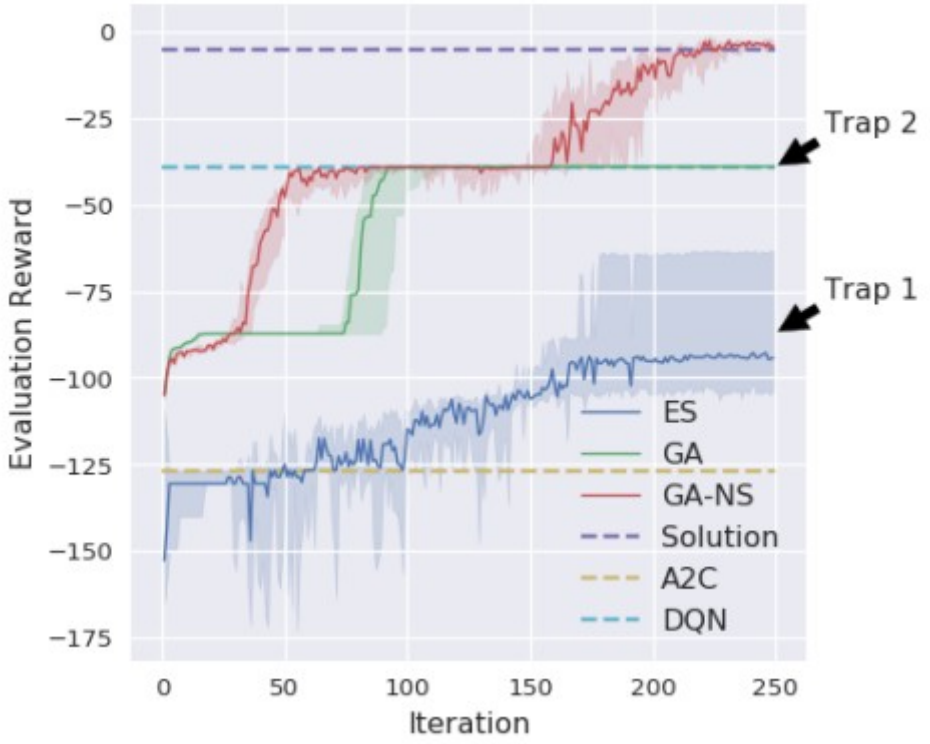


Figure 4. **Image Hard Maze results reveal that novelty search can train deep neural networks to avoid local optima that stymie other algorithms.** The GA, which solely optimizes for reward and has no incentive to explore, gets stuck on the local optimum of Trap 2 (the goal and traps are visualized in 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.

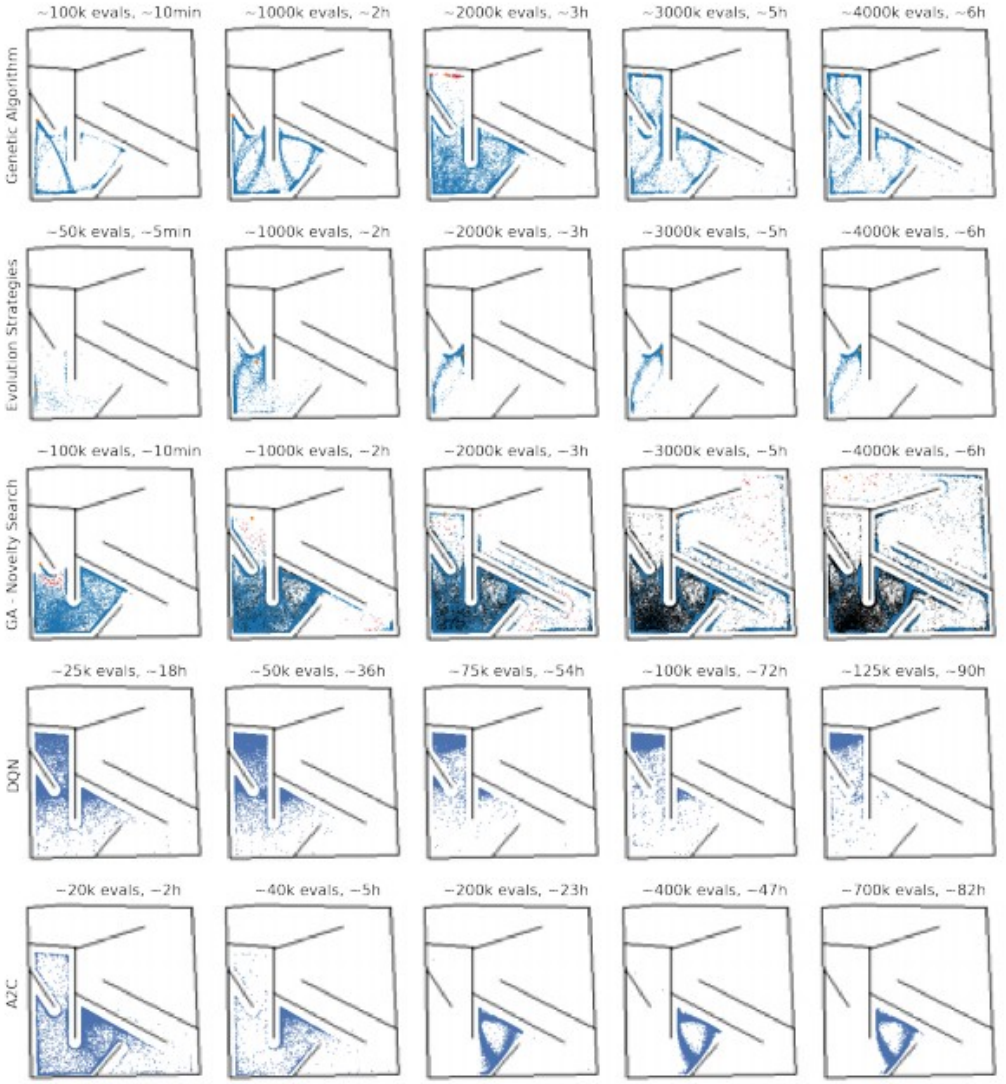
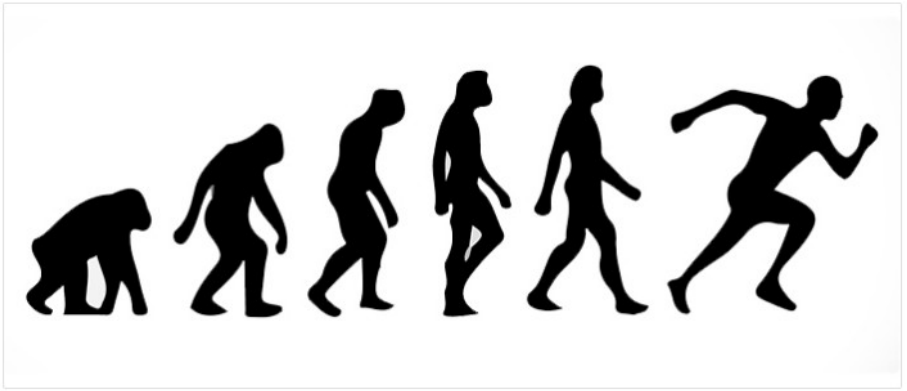


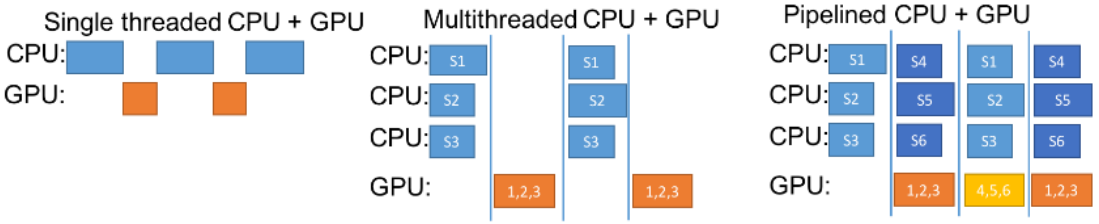
Figure 5. **How different algorithms explore the deceptive Image Hard Maze over time.** Traditional reward-maximization algorithms do not exhibit sufficient exploration to avoid the local optimum (going up). In contrast, a GA optimizing for novelty only (GA-NS) explores the entire environment and ultimately 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.

Accelerating Deep Neuroevolution: Train Atari in Hours on a Single Personal Computer

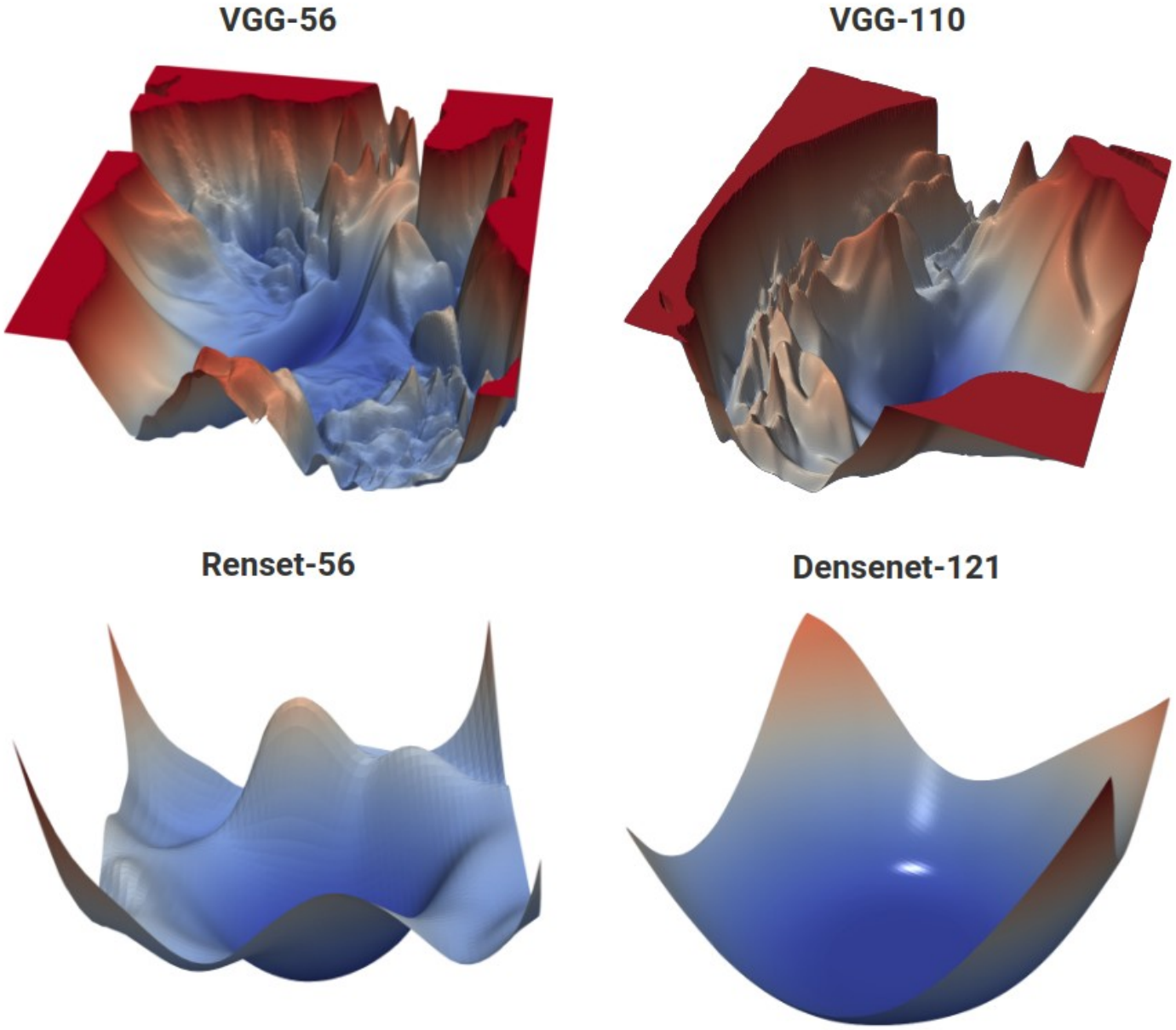
By Felipe Petroski Such, Kenneth O. Stanley and Jeff Clune

April 23, 2018





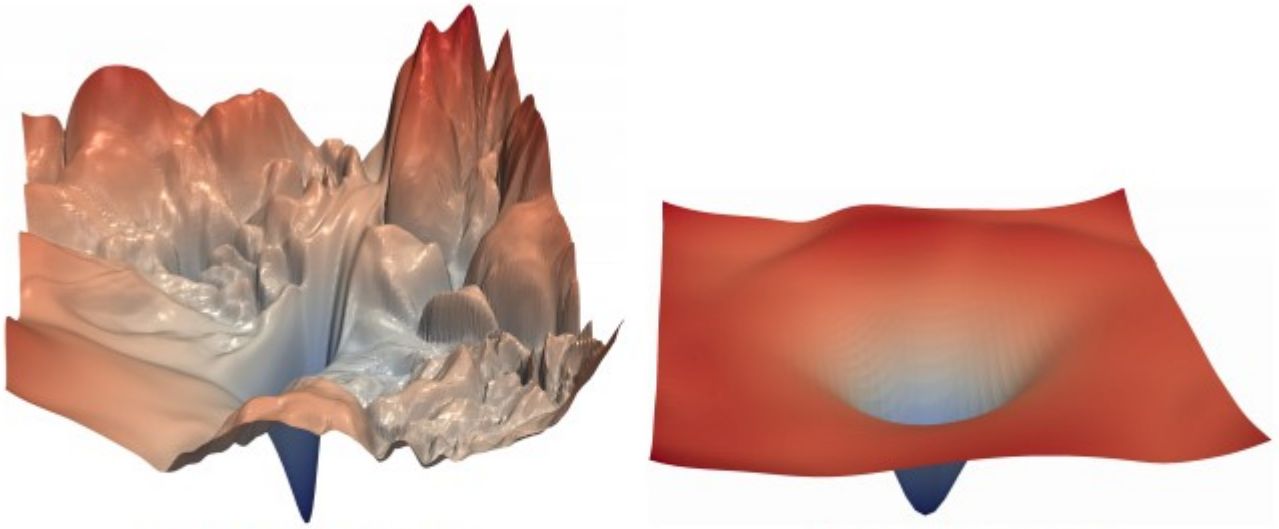
Optimizing the scheduling of populations of heterogeneous networks in RL. The blue boxes are domain simulators, such as the Atari game emulator or physics engines like MuJoCo, which can have episodes of different lengths. A naive way to use a GPU (left) would result in low performance for two reasons: 1) a batch size of one for the GPU, which fails to take advantage of its parallel computation abilities, and 2) idle time while the GPU waits for the CPU and vice versa. A multithreaded approach (center) allows for a more efficient use of the GPU by having multiple CPUs step the simulators in parallel, but causes the GPU to be idle while the CPUs are working and vice-versa. Our pipelined implementation (right) allows the GPU and CPU to operate efficiently. This approach also works with multiple GPUs and CPUs operating simultaneously, which is what we did in practice.



Neural loss functions with and without skip connections. The top row depicts the loss function of a 56-layer and 110-layer net using the CIFAR-10 dataset, without residual connections. The bottom row depicts two skip connection architectures. We have Resnet-56 (identical to VGG-56, except with residual connections), and Densenet (which has a very elaborate set of skip connections). Skip connections cause a dramatic "convexification" of the loss landscape.

VISUALIZING THE LOSS LANDSCAPE OF NEURAL NETS

Hao Li[1], **Zheng Xu**[1], **Gavin Taylor**[2], **Christoph Studer**[3], **Tom Goldstein**[1]
[1]University of Maryland, College Park, [2]United States Naval Academy, [3]Cornell University
{haoli,xuzh,tomg}@cs.umd.edu, taylor@usna.edu, studer@cornell.edu



(a) without skip connections (b) with skip connections

Figure 1: The loss surfaces of ResNet-56 with/without skip connections. The vertical axis is logarithmic to show dynamic range. The proposed filter normalization scheme is used to enable comparisons of sharpness/flatness between the two figures.