Evolution Strategies for Reinforcement Learning in OpenAI Gym

John Moss (Computer Science, Philosophy, Math; 2nd Year Undergraduate, UW-Madison)

***Abstract*—Recent work has shown Evolution Strategies (ES) to be a viable option for parameter optimization in Deep Reinforcement Learning. We evaluate the effictiveness and computational efficiency of ES in optimizing a simple policy network for the *LunarLanding-v2* OpenAI Gym problem, versus a comparable Actor Critic optimization model.**

I. Introduction

Recent advanced in deep learning have led to breakthroughs in a wide breadth of domains, including computer vision[4][5] and speech recognition [6]. However, these successes have often relied heavily on large quantities of hand-labelled data. For many domains, labelling data is prohibitively expensive—as such, a key goal of artificial intelligence is the development of agents capable of succeeding in domains without labelled data.

Reinforcement Learning (RL) is a formulation of this challenge, tasking agents with learning intelligent behaviour from sensory input and a sparse, noisy, often delayed reward signal. Recent success at this task, including systems capable of superhuman performance at Atari[8] and Go[13] have validated the RL perspective and generated significant excitement in the community.

Evolution Strategies (ES) is an alternative to "standard" RL approaches, recently found by Salimens et al. to be competitive in modern RL domains[14], while offering some advantages over mainstream, gradient-based techniques. This paper investigates the effectiveness of ES in the *LunarLanding-v2* environment from OpenAI Gym[10], comparing it to a comparable Actor-Critic model in terms of data efficiency, computational efficiency, and parallelizability.

We find, in the *LunarLanding-v2* environment:

1) ...the performance (in terms of maximum accumulated reward) of our ES implementation to be comparable to our Actor Critic model,
2) ...the data efficiency of ES to be, on average 3.7% of the Actor Critic model, with significantly better per-episode computational costs nearly outweighing the data inefficiency,
3) ...the training efficiency of ES to be significantly higher when learning a probability distribution of actions, rather than maximizing a singular desired action.

II. Reinforcement Learning

The Reinforcement Learning problem is described as "involving agents that interact with an environment, sense their state and the state of the environment, and choose actions based on these interactions"[2]. Here we consider a task of this nature, formalized as an agent interacting with an environment ε via a sequence of observations, actions, and rewards. At each time-step t, the agent picks an action a_t from the set of legal actions A, to be passed to the environment where it may modify the the environments state. An observation s_{t+1} is then returned to the agent, along with a reward r_{t+1}. The task of the agent is to develop a policy π for computing a at any given time-step, given the previous sequence of observations and rewards. Here we measure the success of an agent by way of cumulative reward $\sum_{t=0}^{\infty} r_t$.

A. Deep Reinforcement Learning

Recent breakthroughs in RL have relied upon deep neural networks to develop an effective policy, due in part to their "rich representation" of complex non-linear behaviour[12][8]. Recent literature has primarily been concerned with using successful techniques from deep supervised learning (backprop, gradient descent, etc.) to find optimal network parameters for RL domains. (See: [12], REINFORCE, [9], [11]). One particularly successful trick,

Written as a final report for CS639: Deep Learning, taught by Prof. Jude Shavlik at UW-Madison, Spring 2017 (taken as independent research, CS699)

"experience replay", allows for batching and/or random sampling of observation-action-reward triples, significantly stabilizing off-policy training[3][12]. Evolution Strategies, as a simple black-box optimization technique, is yet another way to achieve the parameter optimization goal. ES, and a standard modern RL approach, Actor Critic Policy Gradients[12] are described below (and in Section III, evaluated, side-by-side, in the *LunarLauncher-v2* OpenAI Gym environment).

Algorithm 1 ES, Salimans (2017)

procedure ES(learning rate: α, noise standard deviation: σ, initial parameters: θ)
  for $t = 0, 1, 2, ...$ **do**
    Sample $\epsilon_0, ...\epsilon_n \sim N(0, 1)$
    Compute $F_i = F(\theta + \sigma\theta)$
    $\theta_{t+1} = \theta_t + \alpha \frac{1}{n\sigma} \sum_{i=1}^{n} F_i \epsilon_i$
  end for
end procedure

B. Evolution Strategies

Evolution Strategies is a class of optimization algorithms first proposed by Rechenberg & Eigen (1973), further developed by Schwefel (1977), and investigated for use in RL by Salimans, et al. (2017). ES algorithms, inspired by natural selection, consist of a parameter vector perturbed to create a number of candidate vectors, each evaluated against an objective function. Each iteration ("generation") of this process results in a new "parent" parameter vector, computed by combination of the most successful (highest scoring) candidates ("children").

In the context of RL, ES can be thought of as optimizing the policy function $\pi(a|s;\theta)$ by gradient ascent (more intuitively, finite difference hill climbing) on $\mathbb{E}(R_t)$.

The particular ES algorithm described by Salimans et al. (2017) is as follows. Let F denote the objective function acting on parameters θ. Then θ can be optimized over (by way of stochastic gradient descent) with the following score estimator:

$$\Delta_\theta(\theta) = \frac{1}{\sigma} \mathbb{E}_{\epsilon \sim N(0,1)} \{F(\theta + \sigma\epsilon)\epsilon\} \qquad (1)$$

This equation translates to a simple algorithm wherein (1) candidate vectors are generated by application of a gaussian to the parameter vector, (2) the objective score of each calculated, and (3) a new parameter vector is computed by the linear combination of candidate vectors and their objective function scores (Algorithm 1). ES differs from traditional approaches to weight optimization, most significantly in that it is gradient-free. Salimans et al. find several advantages to not computing a gradient:

1) Obvious savings in memory and computation (estimated around two thirds under standard network architectures)
2) Ease of distributed computation: gradient-based approaches, such as policy gradients and Q-learning, require the communication of gradients in order for workers to update parameters. ES requires only that two scalars be communicated—reward accumulated over the episode (from worker to master), and gaussian noise seed (from master to worker). (Assuming a shared noise table is distributed in the initial step, which has been shown to be both bandwidth and computationally cheap.)
3) Lack of dependence on high-precision calculations (e.g. gradients) allow for weight optimization on specialized hardware typically used only for "feed-forward" prediction (such as TPUs).
4) Interestingly, ES (and other black-box optimizers) are invariant to the frequency of interaction the agent has with the environment. Standard RL approaches depend heavily on an appropriate "frameskip" hyperparameter for successful optimization[12]. By perturbing the parameter (i.e. policy) space rather than the action space, ES avoids this complication.

C. Actor Critic Policy Gradients

Like ES, the Actor Critic approach directly parameterizes the policy $\pi(a|s;\theta)$ and aims to optimize the parameters θ by performing gradient ascent on $\mathbb{E}[R_t]$[12]. This is achieved by way of two complementary learned functions: an actor (here, π) and a critic (here, V). Intuitively, the critic learns to represent the value of some given state, and the actor learns a policy based on this representation.

REINFORCE[1], a classic policy-based model, was an early take on actor/critic interaction. In this model, parameters θ are updated in the direction $\Delta_\theta \mathbb{E}[R_t]$, shown to be estimated without bias by $\Delta_\theta log\pi(a_t|s_t;\theta)R_t$. The variance of this estimate

is found to be reduced by subtracting a learned function of the state $b_t(s_t)$, known as the baseline[1], from the discounted reward R_t. Recently, learned value functions have been used in place of the baseline $b_t(s_t) \approx V^\pi(s_t)$[2]. When this is the case, the policy gradient can be understood as being scaled by $R_t - V(s_t)$, or the *advantage* (A) of action a_t at state s_t. (Note R_t is itself an approximation of $Q(a_t, s_t)$, yielding the intuitive definition of advantage as $Q(a_t, s_t) - V(s_t)$)[12].

Both the actor π and the critic V can be learned by standard gradient descent on the loss functions $L_V = (R_t - V(s_t))^2$ and $L_\pi = log(\pi(s_t)) * A(s)$, respectively, and can be trained continuously and simultaneously by an RL agent[1].

III. Methodology

We compare two methods of parameter optimization, ES and an implementation of AC, on the *LunarLander-v2* environment of OpenAI Gym.

A. Implementation

All experiments were implemented in Python, with the ES model implemented in Numpy, and the Actor Critic model implemented in TensorFlow[7]. Training and evaluation took place on an AWS EC2 g2.2xlarge instance, with the TensorFlow models utilizing a NVIDIA K20 GPU for training acceleration via CuDNN.

B. Policy Network

In each instance, the policy network consists of an artificial neural network with one hidden layer (consisting of 4 ReLU units) and a softmax output layer. Weights are normally distribution with mean 0 and standard deviation 1, bias' are initialized to 0.

C. ES

"Generations" are generated with population size 100, and drawn from the initial parameter vector normally with mean 0 and standard deviation 0.1. The learning rate used was $\alpha = 0.00025$. Rewards were *not* normalized, in an effort to retain similarity with the actor critic model. Further, to help offset a correlation between environment initial state and accumulated reward, each candidate performs 3 episodes, with the average accumulated reward recorded. Two variations are tested: one in which actions are sampled from the output layer as a probability distribution (i.e. the network learns $P(a)$), the other in which the action is taken as the max node in the output layer.

(Note to Prof.: I wrote the ES implementation myself, with some light guidance from Andrej Karpathy's blog post on the subject.)

D. Actor Critic

Our implementation of the policy network is as above, with a similar value network consisting of one hidden layer (with 5 ReLU units) and a single linear output node. Rewards are estimated with discount factor 0.8 over a maximum of 350 time steps. As with the ES implementation, 3 episodes of observation-acion-reward triples are collected before updating the critic network, in order to combat high correlation between non-determinism in the environment and cumulative reward. Both policy and value networks are implemented in TensorFlow, and optimized by the TensorFlow AdamOptimizer implementation, with the default learning rate of $\alpha = 0.001$, beta 0.9, beta2 0.99, epsilon $1e-08$.

(Note to Prof.: The Actor Critic model was heavily modified from the implementation written by Yuke Zhu, hosted on Github at yukezhu/tensorflow-reinforce.)

E. OpenAI Gym

OpenAI Gym is a collection of RL benchmark environments with a simple Python interface for developing agents and comparing results in a standardized fashion across learning algorithms[10].

F. LunarLander-v2

LunarLander-v2 is a particular environment within OpenAI Gym in which the goal is to land a lunar pod gently within designated markers. At each time-step the agent is given an 8-dimensional observation vector, with components corresponding to coordinate position, velocities, leg-touch-down-detection, etc. The agent is offered a discrete action space in which it may fire the main (downward) jet,

[1]Some implementations, not including our own, add the entropy of the policy, $H(\pi)$, as a secondary component to L_π. This has been shown to help prevent premature convergence to local optima in some cases.

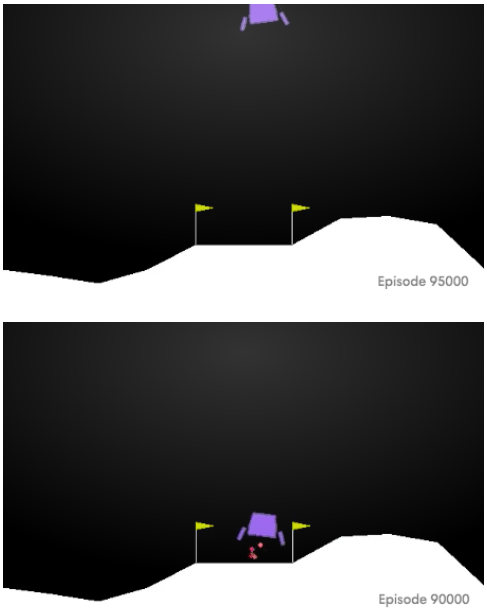

Fig. 1. *LunarLander-v2* environment from OpenAI Gym

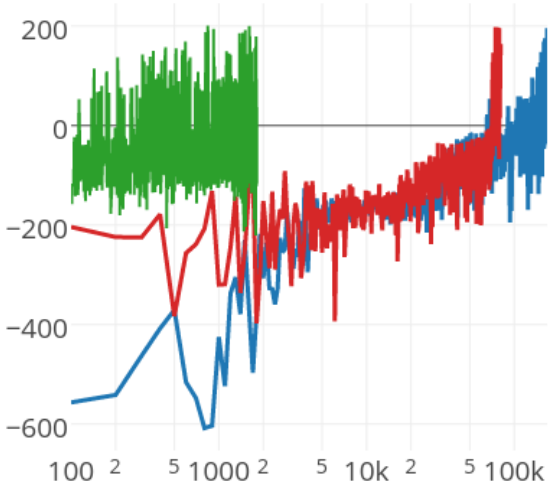

Fig. 2. Cumulative reward, per episode. x-axis (log scale): episode, y-axis: cumulative reward. ES (argmax) – Blue, ES (P) – Red, Actor Critic – Green

and/or the left/right jets. Landing upright provides a reward of 100, while crashing yields −100. Each leg-ground contact provides a reward of 10, and each frame spent firing the main jet gives reward −0.3. "Solving" the environment is considered accumulating a score of 200.

IV. Data

We compare ES and Actor Critic on the criteria of overall performance and training efficiency in the *LunarLander-v2* environment, measuring specifically: maximum cumulative reward, and cumulative reward per episode (shape of the training curve).

A. ES vs. Actor Critic

The Actor Critic model reached the maximum rewards achieved by ES in less than 4% the episodes, making it at least 25x as data efficient, with comparable success (converging to comparable rewards). Despite known difficulties in quantifying computational efficiency in "wall clock" time, it may be worth noting that each of the first 10 episodes took the Actor Critic model, on average, 2.28s to compute. The ES model, by comparison took 10.4s (on average) to compute each of the first 10 episodes, implying a mean rate of 0.035s per episode—21.9x faster than the Actor Critic model, despite the Actor Critic model being trained on a dedicated GPU.

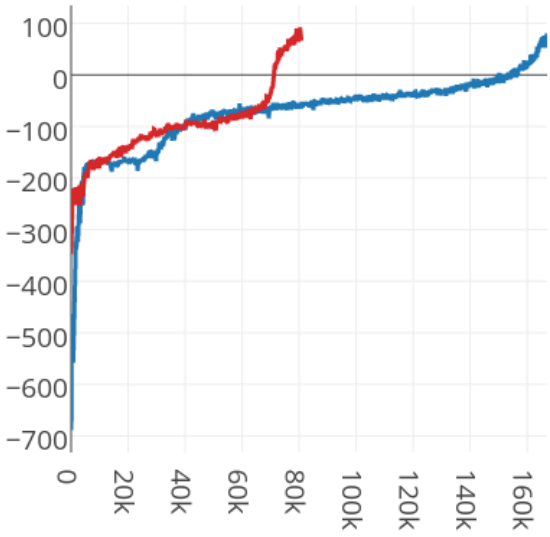

Fig. 3. Average reward across population, per episode. x-axis: episode, y-axis: average episodic reward. ES (argmax) – Blue, ES (P) – Red

This disparity is likely due to ES's freedom from computing gradients, and the multiple off-policy gradient updates required per policy change in the Actor Critic model. It seems probable that further speedups could result from exploiting the natural parallelizability of ES[14], making ES an even more attractive choice for some domains.

It is also interesting to note the relative stability of each model, in terms of cumulative reward over

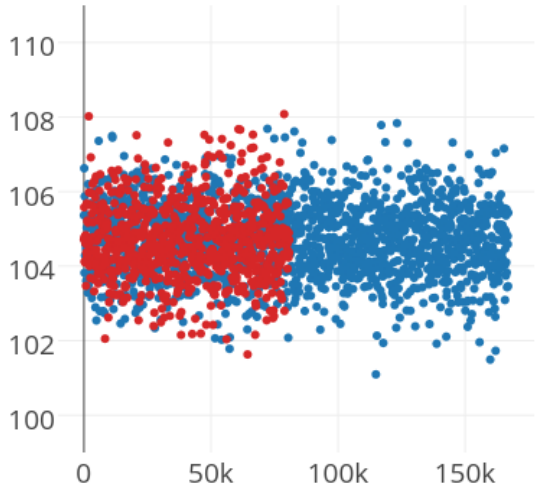

Fig. 4. Magnitude of parameter update, by episode. x-axis: episode, y-axis: parameter update magnitude. ES (argmax) – Blue, ES (P) – Red

training episode. The Actor Critic model, while ultimately training faster, has a significantly higher reward variance, as compared to the ES model (a fact easily explained by analysis of the two methods, but interesting to observe nonetheless).

B. Learning argmax vs. probability distribution

Standard RL techniques rely heavily on action sampling as a core mechanism for policy exploration. Given that ES injects noise at the policy level directly, it seems plausible that ES may not need such a mechanism. This, in conjunction with the observation that a significant proportion of ES implementations learn the argmax function, rather than the probability distribution, motivated us investigate the relative impact of learning one versus the other.

As it turns out, this decision has a vast impact on training efficiency—the probability distribution requiring *less than half* the episodes to reach comparable rewards. (See Fig. 2, 3).

V. Related Work

Related work includes the general (and most recent) work on ES for RL by Salimans et al. (2017), and previous work (noted by them) by Koutnk et al. (2013; 2010) and Srivastava et al. (2012) in applying ES to RL more narrowly. They also note work by Wierstra et al. (2008;2014) on training recurrent weights in RNNs with black box optimization, and Hyper-Neat (Stanley et al., 2009) as a similar approach to evolving ANN weights.

This work was similar in experimental design to [12], where various Deep Reinforcement Learning methods are compared against new, asynchronous methods.

VI. Future Work

Future work could involve any the following:

1) Further investigation into the causes and consequences of the vast disparity in training efficiency seen in Fig. 2.
2) The evaluation of ES against other RL approaches (than Actor Critic), and in other RL environments.
3) Investigation of the applicability of ES in training GANs and other notoriously difficult architectures

References

[1] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, no. 3, pp. 229–256, 1992, ISSN: 1573-0565. DOI: 10.1007/BF00992696. [Online]. Available: http://dx.doi.org/10.1007/BF00992696.

[2] R. S. Sutton and A. G. Barto, *Introduction to reinforcement learning*, 1st. Cambridge, MA, USA: MIT Press, 1998, ISBN: 0262193981.

[3] S. Lange, T. Gabel, and M. Riedmiller, "Batch reinforcement learning," 2011.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., Curran Associates, Inc., 2012, pp. 1097–1105.

[5] P. Sermanet, K. Kavukcuoglu, S. Chintala, and Y. LeCun, "Pedestrian detection with unsupervised multi-stage feature learning," *CoRR*, vol. abs/1212.0142, 2012. [Online]. Available: http://arxiv.org/abs/1212.0142.

[6] A. Graves, A. Mohamed, and G. E. Hinton, "Speech recognition with deep recurrent neural networks," *CoRR*, vol. abs/1303.5778, 2013. [Online]. Available: http://arxiv.org/abs/1303.5778.

[7] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Y. Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng, *TensorFlow: Large-scale machine learning on heterogeneous systems*, Software available from tensorflow.org, 2015. [Online]. Available: http://tensorflow.org/.

[8] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015, ISSN: 0028-0836.

[9] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust region policy optimization," *CoRR*, vol. abs/1502.05477, 2015. [Online]. Available: http://arxiv.org/abs/1502.05477.

[10] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *CoRR*, vol. abs/1606.01540, 2016. [Online]. Available: http://arxiv.org/abs/1606.01540.

[11] Y. Duan, J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel, "Rl2: Fast reinforcement learning via slow reinforcement learning," *CoRR*, vol. abs/1611.02779, 2016. [Online]. Available: http://arxiv.org/abs/1611.02779.

[12] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," *CoRR*, vol. abs/1602.01783, 2016. [Online]. Available: http://arxiv.org/abs/1602.01783.

[13] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, Jan. 2016, ISSN: 0028-0836. [Online]. Available: http://dx.doi.org/10.1038/nature16961%20http://10.1038/nature16961%20http://www.nature.com/nature/journal/v529/n7587/abs/nature16961.html%7B%5C#%7Dsupplementary-information.

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