GrES: Group Evolutionary Strategies for Sample Efficient Direct Policy Learning

Coen Armstrong
Stanford University
crrarmstrong@stanford.edu

Diego Jasson
Stanford University
djasson@stanford.edu

Mario Srouji*
Apple, Inc.
msrouji@apple.com

Abstract

Evolutionary Strategies (ES) have recently proven competitive with Reinforcement Learning (RL) in some tasks where their decreased computational cost offsets their lower sample efficiency. We demonstrate ES can be further improved in both sample efficiency and wall-clock time, overcoming traditional RL methods in several benchmarks. We introduce Group Evolutionary Strategies (GrES), an ES algorithm where the population is split into groups of individuals, each taking turns in the same group environment, and collecting individual episodic rewards through a credit assignment mechanism. This differs from standard ES, where each individual acts through an entire episode, and we show experimentally that the sample efficiency of GrES scales with the group size. In our layered implementation, each group member differs from the parent network only in one layer, so we separate the evolution along each layer. To explain the performance of GrES, we isolate the effects of the group size, the credit assignment mechanism, the length of individual turns, and the layering element on the robustness and efficacy of the learned policy. In particular layering brings a big memory improvement, because instead of having to store one model for each member of the next generation, we need only store one layer. We highlight current limitations of existing DNN libraries by implementing multi-model-parallelism (MMP) on a single GPU, which provides an additional acceleration factor on the wall-clock convergence time of ES in general, and of GrES in particular. Our combined GrES and MMP implementation achieves an order of magnitude speed up factor with respect to vanilla ES, and is therefore competitive or better than RL in several benchmarks.

1 Introduction

Simple Evolutionary Strategies (ES) have been recently proven to be competitive, although not yet at par, with RL [3, 9, 11, 14, 19]. When compared to RL, ES can be interpreted as derivative-free direct-policy optimization methods, which either do not make use of the gradient at all (like genetic algorithms [19]) or approximate the gradient with finite differences [14]. ES have several potential advantages: since the fitness evaluation is independent from action frequency and delayed rewards, ES are also tolerant to extremely long horizons; furthermore, their computational cost is not affected by the computation backward passes, temporal discounting, and value function evaluation [14]. ES also scale naturally to distributed systems.The drawback of ES is that fitness evaluation and consequently policy update requires the entire population to complete a full episode. Hence ES are generally less sample efficient than RL, where policy update can occur at high frequency while training, e.g.

*We found the project idea on the Fall Quarter 2018 list from a previous CS229 CA (Mario Srouji, msrouji@apple.com), who is now working at Apple as a Machine Learning Research Engineer. He will be additionally supervising this work as an open collaboration between Stanford and Apple, in hopes of future publication.

34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

by bootstrapping from the current estimate of the value function in model-free temporal difference RL [12]. Overall, despite the smaller computational footprint partially offsetting the worse sample efficiency, ES still have worse wall-clock time than RL [14, 11].

We show that the computational efficiency of ES can be significantly improved through Group Evolutionary Strategy (GrES), where the population is split into groups of individuals. This parallels biological evolution which includes some selection on the level of aggregates like organism or groups rather than just as cells. Within each group, each individual takes turns to act in the same environment, while rewards are dispatched to each individual with a credit assignment mechanism. The credit assignment mechanism varies how much of an organism's fitness score comes from other group members. We normally have half of the score from other groups, but other assignments are possible and may be desirable given the parameters of the problem. We generate offspring by perturbing one layer of the parent network, producing groups where each member is a perturbation of the parent in one specific layer. In this setting, group size is the same as number of layers, though we can also apply the grouping strategy without layering (in which case offspring are generated as in standard ES). This allows us to store only the changed layers in the group, rather than n extra models as usual, which provides a memory improvement from $O(n^2)$ in the number of layers (which equals the group size by definition) to $O(n)$.

Our experiments demonstrate an improvement in sample efficiency proportional to the group size, and provide a proportional speed-up of up to $4\times$ in wall-clock convergence time.

2 Related Work

The problem of learning a policy to execute a given task is accomplished in RL by a maximizing proper cumulative reward function. Early successes on small-tabular state spaces [20] were extended to high dimensional problems by the representational power of deep neural networks (DNN), thus inaugurating Deep RL (DRL). This allows effective policy gradient [17], value iterations [13], or actor-critic [12] algorithms, together with their improved versions accelerated by higher sample efficiency [15, 16] or efficient implementations [1, 4, 18]. These are all gradient-based methods: DQN, for instance, updates the weights of the DNN Q-value function approximator to decrease the loss on Q-value prediction, whereas policy gradient methods sample behaviors stochastically from the current policy and then reinforce those that perform well via stochastic gradient ascent.

It's recently become clear that ES can be competitive with DRL on both simulated and real environments, while it was formerly believed that the policy search space was too large for simple and derivative-free optimization methods[6, 21, 14, 19]. A simplified version of Natural Evolution Strategies that learns the mean of a distribution of parameters, but not its variance, is competitive with DQN [13] and A3C [12] on difficult RL problems [21], with shorter wall-clock training time when trained on a large cluster of CPUs due to effective scaling with parallelization [14].

Following the gradient to maximize the expected discounted reward, like in DRL, is not always the best strategy for policy learning [19], where a genetic algorithm based on mutation and elitism is also forced to explore new policies by a proper term in the fitness function to prevent early convergence to a sub-optimal policy.

The idea of using teams of agents acting in the same environment is not novel in DRL, but to the best of our knowledge, it has never been used to improve sample efficiency. Instead, Multi-Agent RL (MARL) has been used where multiple actors have to learn to collaborate, and potentially specialize for different sub-tasks [5], or when different agents optimize different rewards function in the case of team-average reward [22], when a multi-objective cost function has to be optimized and the diversification of the reward function may lead to better exploration and diversification of the agents. However, both GrES and MARL require assigning rewards from the environment to the proper agent [5].

In common with other ES implementations, random noise seeds are used to recompute on the fly the DNN weights of different individuals by limiting at the same time the memory occupancy and communication overhead.

Researchers also noticed that ES and even trivial strategies like random search can be competitive with DRL initially, possibly because there are multiple solutions easily discovered by random sampling when the initial population is large enough [11, 19]. However, this advantage is lost when fine tuning

the policy. This observation justifies adoption of hybrid ES/DRL methods that can benefit from the advantages of both strategies [3, 9], and from GrES and MMP too. This hybridisation may be even more beneficial because of the emergence of the Baldwin effect, the possibility that individuals who evolve to be effective learners may have a reproductive advantage [3]. Other forms of hybridization between global and local search algorithms include meta-optimization [7]; in this context, one of the main issue is balancing between the depth and width of the search [10], which is in common with the variance-bias trade-off in temporal difference RL [20**?** , 1, 2] and the problem of setting the length of the turn for each individual in GrES.

3 Method

We will first describe the standard Evolutionary Strategies (ES) method, as our Group Evolutionary Strategies (GrES) method is only different in 4 ways, and actually becomes the standard ES algorithm when the group size is set to 1 and layering is turned off. ES is a black-box optimization algorithm inspired by natural evolution, which aims to maximize a "fitness" score with respect to "parameters" for a given task (or environment in the sense of RL). The ES algorithm has three main steps:

1. At every generation, create a population of parameter vectors through perturbations ('mutation').
2. Evaluate the objective function ('fitness') for all the population by observing their score on a full task episode.
3. Recombine the parameter vectors according to their achieved fitness, with higher-scoring members having more influence, to form the population for the next generation.

This procedure is iterated until the objective function is fully optimized (in terms of the fitness). There are many ways to represent the fitness for a given environment, the population, and how they perform mutations, and re-combinations. In this work, we focus on Reinforcement Learning (RL) problems, and hence our fitness function will be the stochastic episodic-return provided by running an environment. Our population of parameters will be the parameters of a deterministic or stochastic policy describing an agent acting in that environment, which in our implementation are the weights of a Deep Neural Network (DNN). The policy is controlled by either discrete or continuous actions, and we learn a direct-mapped policy - e.g a function mapping environment state (input) directly to environment action (output). The mutations are performed by perturbing a parameter vector at each generation, which is done by injecting (adding) random Gaussian noise to the vector, enough times to produce the desired population size. Once all of the members in a given population run an entire episode (or multiple), the re-combination step does a weighted addition of all of the mutation noise vectors used and their respective fitness scores to make an update to the current generation of parameters. We generally use elitism (picking the top performers) rather than a more sophisticated recombination step, to focus on the layering and grouping strategy.

We make the following modifications to standard ES to achieve superior sample efficiency:

1. For a population size n and groupings G of size g, we create $\frac{n}{g}$ groups. Each group collectively is tested on one episode We run $c * \frac{n}{g}$ episodes to evaluate the fitness a given generation of individuals, instead of $c * n$ as in standard ES (where c is a constant number of episodes used to evaluate the fitness of an individual or group).
2. The members inside of each group take turns in a given environment through a random pick-without-replacement scheme. We randomly select an individual i from the group to play t time-steps in the environment, then select the next random individual (without considering i), and so on. If the episode is not finished and all individuals have played, we re-initialize the selection pool to the whole given group.
3. After running all c episodes to evaluate the fitness of the group, the fitness of each individual i in the group is assigned based on (1) the cumulative score that individual collected s_i and (2) the sum of the scores of the other individuals in the group $\sum_{j \in G \setminus \{i\}} s_j$. We call the final assigned fitness for each individual in the group $f_i = \frac{g}{2g-1} * s_i + \frac{1}{2g-1} * \sum_{j \in G \setminus \{i\}} s_j$. In this method, we are able to assign individual fitness scores that more heavily weight an individual's score in the group, while also taking into account the scores

of the other individuals (since the performance of any given individual is affected by the group's performance). We could equally try different credit assignment mechanisms.

4. Each group member in the next generation is seeded by changing only one layer of the parent network: we add random noise to the ith layer of the ith group member. We then store only the changed layers in the group, so 1 extra model of n layers for group size n, rather than n extra models as usual.

The rest of the ES algorithm is left untouched from [14], and in fact our GrES algorithm simplifies to standard ES exactly when the group size $g = 1$ and layering is turned off.

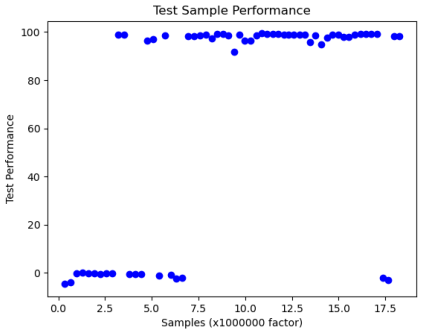
Our layered implementation achieves a significant memory improvement over both standard evolution and group evolution without layering. In standard ES, the next generation of models must be saved one for each member of the generation, so if we have n offspring and k layers the memory requirement is $O(nk)$. In layered evolution, we only change the parent model at one layer for every individual offspring, so we only need to store one layer per offspring, so it scales as $O(n)$. Note that in layered evolution $n = k$, so the memory requirement scaling is square rooted. In our implementation, we store the n layers associated to a group as a new mode, so we are effectively storing only 2 models. Hence layered and regular ES have equivalent memory consumption for $n = k = 2$ but not otherwise.

4 Results

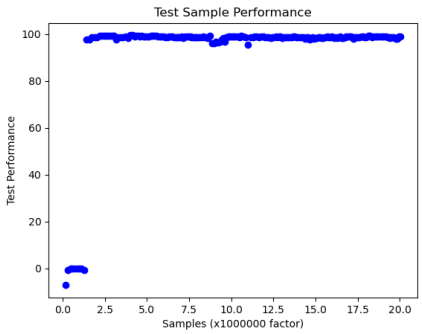
To demonstrate the general applicability and effectiveness of our approach, we experiment across a diverse set of environments, including locomotive and control tasks with both discrete and continuous actions spaces. We avoid any environment specific hyper-parameter tuning, and fixed all hyper-parameters throughout the training session and across experiments. We vary only the group size, population size, or both (depending on the experiment), in order to properly isolate the effects of grouping in GrES. Since favoring random seeds can introduce large bias on the model performance [8], we tested on 3 different random seeds, but the results were fairly similar. We tested layering on Acrobot, Bipedal Walker, LunarLander, LunarLanderContinuous, MountainCarContinuous, and Pendulum. Layering quickly solved all tasks except for Bipedal Walker and Pendulum. For reasons of space, we only include plots for MountainCarContinuous.

We test the effect of GrES on sample efficiency by fixing the population size, while doubling the group size for each subsequent experimental setup. We observe a nearly-linear increase in sample-efficiency with the group size (when compared to a group size of 1). There will be some size with theoretically maximum efficiency, after which increasing the group size no longer helps, because bigger groups mean individual policies play for a shorter time, introducing additional variance.

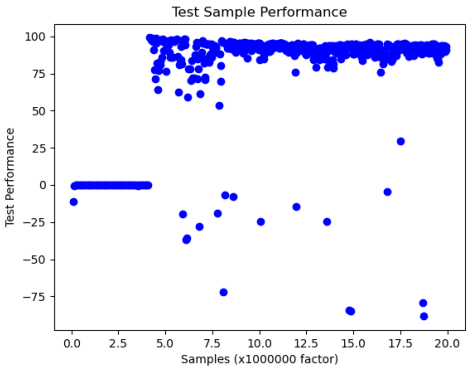
MountainCarContinuous provides a clear example.



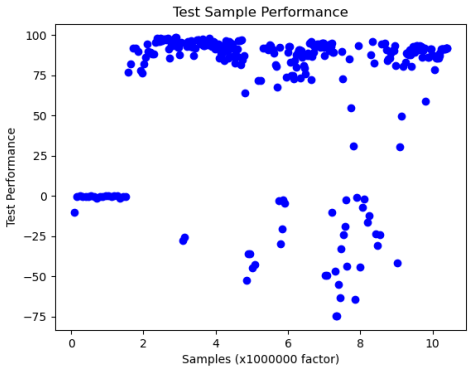
(a) MountainCarContinuous without Grouping.



(b) MountainCarContinuous with Group Size 2 and layering



(a) MountainCarContinuous with Group Size 4 and layering



(b) MountainCarContinuous with Group Size 4 and no layering

Since MountainCarContinuous can already be solved extremely quickly with group size 2, increasing group size then becomes detrimental.

We compare GrES with layering (ie where offspring differ from the parent only in 1 layer) to standard GrES to standard ES. This compares favourably to the no-grouping plot shown beforehand (standard ES). GrES without layering converges faster but takes more memory and deviates from the optimum more. Speculatively, this might be because evolution along each layer is more correlated.

5 Conclusion

Hence, we have demonstrated ES can become substantially more sample efficient, as well as use much less memory.

We ascribe GrES's superior performance to several factors. Agents must perform well with other policies in their group, since part of their fitness comes from other policies. This could be a way of reducing variance, given that the grouping strategy increases variance. It also parallels biological evolution, since in biological evolution agents are selected not just based on direct individual characteristics, but also social characteristics. Agents must be more robust to starting at different points, because they are randomly swapped into a given environment (rather than playing through it all themselves), giving a regularisation effect. We can pick an optimal credit-assignment mechanism along the bias-variance tradeoff, since playing for a shorter time increases the variance, but changing credit assignment changes bias. The layering strategy decouples training along each layer of the network, allowing more efficient exploration of the policy space. Finally, the layering strategy is substantially more memory efficient as the number of hidden layers increases.

Some important further directions are to test on harder environments, like MuJoCo and Atari, and perform more sensitivity and ablation tests. In particular, more sensitivity and ablation tests should focus on investigating the credit assignment mechanism. We should also investigate further which of our causal effects contributes most. Finally, we should pursue theoretical results as an underlying basis for this work, especially the 'decoupling' of evolution along each layer in Layered GrES.

6 Contributions

Coen and Diego worked on the implementation together, with help from Mario when problems were encountered. Coen ran the experiments on his machine and in GCP. Mario contributed the original idea and an approach to the problem. Both Coen and Diego contributed to the write up.

References

[1] Mohammad Babaeizadeh, Iuri Frosio, Stephen Tyree, Jason Clemons, and Jan Kautz. Reinforcement learning through asynchronous advantage actor-critic on a gpu. In *ICLR*, 2017.

[2] Steven Dalton, Iuri Frosio, and Michael Garland. Gpu-accelerated atari emulation for reinforcement learning. *CoRR*, abs/1907.08467, 2019.

[3] Madalina M. Drugan. Reinforcement learning versus evolutionary computation: A survey on hybrid algorithms. *Swarm and Evolutionary Computation*, 44:228 – 246, 2019.

[4] Lasse Espeholt, Hubert Soyer, Rémi Munos, Karen Simonyan, Volodymyr Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. IMPALA: scalable distributed deep-rl with importance weighted actor-learner architectures. *CoRR*, abs/1802.01561, 2018.

[5] Borja Fernandez-Gauna, Ismael Etxeberria-Agiriano, and Manuel Graña. Learning multirobot hose transportation and deployment by distributed round-robin q-learning. *PloS one*, 10:e0127129, 07 2015.

[6] M. Hausknecht, J. Lehman, R. Miikkulainen, and P. Stone. A neuroevolution approach to general atari game playing. *IEEE Transactions on Computational Intelligence and AI in Games*, 6(4):355–366, 2014.

[7] G. Heinrich and I. Frosio. Metaoptimization on a distributed system for deep reinforcement learning. In *2019 IEEE/ACM Workshop on Machine Learning in High Performance Computing Environments (MLHPC)*, pages 19–30, 2019.

[8] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[9] Shauharda Khadka, Somdeb Majumdar, Tarek Nassar, Zach Dwiel, Evren Tumer, Santiago Miret, Yinyin Liu, and Kagan Tumer. Collaborative evolutionary reinforcement learning. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 3341–3350. PMLR, 2019.

[10] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *J. Mach. Learn. Res.*, 18(1):6765–6816, January 2017.

[11] Horia Mania, Aurelia Guy, and Benjamin Recht. Simple random search of static linear policies is competitive for reinforcement learning. In *NeurIPS*, 2018.

[12] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1928–1937, New York, New York, USA, 20–22 Jun 2016. PMLR.

[13] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015.

[14] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning, 2017.

[15] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization. *CoRR*, abs/1502.05477, 2015.

[16] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[17] Frank Sehnke, Christian Osendorfer, Thomas Rückstieß, Alex Graves, Jan Peters, and Jürgen Schmidhuber. Parameter-exploring policy gradients. *Neural Networks*, 23(4):551 – 559, 2010. The 18th International Conference on Artificial Neural Networks, ICANN 2008.

[18] Adam Stooke and Pieter Abbeel. Accelerated methods for deep reinforcement learning, 2018.

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

[20] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[21] Daan Wierstra, Tom Schaul, Tobias Glasmachers, Yi Sun, Jan Peters, and Jürgen Schmidhuber. Natural evolution strategies. *J. Mach. Learn. Res.*, 15(1):949–980, January 2014.

[22] Kaiqing Zhang, Zhuoran Yang, and Tamer Başar. Multi-agent reinforcement learning: A selective overview of theories and algorithms, 2019.