Importance Weighted Evolution Strategies

Víctor Campos
Barcelona Supercomputing Center
victor.campos@bsc.es

Xavier Giro-i-Nieto
Universitat Politècnica de Catalunya
xavier.giro@upc.edu

Jordi Torres
Barcelona Supercomputing Center
jordi.torres@bsc.es

Abstract

Evolution Strategies (ES) emerged as a scalable alternative to popular Reinforcement Learning (RL) techniques, providing an almost perfect speedup when distributed across hundreds of CPU cores thanks to a reduced communication overhead. Despite providing large improvements in wall-clock time, ES is data inefficient when compared to competing RL methods. One of the main causes of such inefficiency is the collection of large batches of experience, which are discarded after each policy update. In this work, we study how to perform more than one update per batch of experience by means of Importance Sampling while preserving the scalability of the original method. The proposed method, Importance Weighted Evolution Strategies (IW-ES), shows promising results and is a first step towards designing efficient ES algorithms.

1 Introduction

The pace of advances in machine learning is frequently upper bounded by the time taken to train models. Even though hardware manufacturers continuously provide improvements in computational power [14], the community has turned to distributed solutions to further reduce training times [5] and training larger models [22]. However, accelerating an algorithm by distributing it across several computing devices is not always a trivial task. The communication overhead precludes the distribution of some methods beyond a reduced number of machines [3, 2], and sometimes parallel training can even hinder the final performance of the model when done naively [7]. This motivates research efforts towards developing algorithms that are well suited for parallel training, from both learning and computational standpoints.

Evolution Strategies (ES) [18] were proposed as a scalable alternative to popular Reinforcement Learning techniques. Thanks to a reduced communication overhead, ES can be scaled to over a thousand CPU cores with almost linear speedup, providing massive improvements in wall-clock time when training agents in well-known RL benchmarks. However, this speedup comes at the cost of a reduced data efficiency, i.e. ES need more interactions with the environment to achieve the same score as competing methods. Even though this trade-off might not be problematic for simulated tasks, where one can turn compute into data, data efficiency is crucial for the deployment of RL agents in real world scenario, e.g. robot manipulation tasks [17]. Research has been conducted to improve the data efficiency of other RL methods [19, 8], and we believe that ES would benefit from similar efforts as well.

We aim at improving the data efficiency of ES, while maintaining the scalability of the original method. Our contributions can be summarized as follows: (1) we propose Importance Weighted Evolution Strategies (IW-ES), an extension of ES that can perform more than one update per batch of

Preprint. Work in progress.

experience, (2) analyze the scalability of IW-ES from the computational standpoint, and (3) report preliminary results for IW-ES under different configurations that provide insight on the potential of the method and possible improvements to overcome its current limitations.

2 Background: Evolution Strategies

The term *Evolution Strategies* (ES) [16] makes reference to a class of black box optimization algorithms which implement heuristics inspired by natural evolution. However, throughout this manuscript, we will use the term to refer to the particular algorithm proposed by Salimans et al. [18]. This method, which belongs to the class of Natural Evolution Strategies [28, 27], was shown to be competitive for solving RL problems while exhibiting some attractive features. These features include invariance to action frequency and reward distribution, the possibility to optimize non-differentiable policies, and being highly parallelizable thanks to an efficient communication strategy.

Let F denote the objective function acting on parameters θ, defined in RL problems as the stochastic score experienced by an agent after a complete trajectory. ES seeks to maximize $\mathbb{E}_{\theta \sim p_\psi} F(\theta)$, the average objective over a population of solutions p_ψ, using the score function estimator for the gradient. Salimans et al. [18] instantiate the population as a multivariate Gaussian with diagonal covariance matrix centered at θ, thus obtaining the following estimator:

$$\nabla_\theta \, \mathbb{E}_{\epsilon \sim N(0,\sigma^2 \mathbf{I})} \, F(\theta + \epsilon) = \frac{1}{\sigma^2} \, \mathbb{E}_{\epsilon \sim N(0,\sigma^2 \mathbf{I})} \left[F(\theta + \epsilon)\epsilon \right] \tag{1}$$

which in practice is estimated with samples:

$$\nabla_\theta F(\theta) \approx \frac{1}{n\sigma^2} \sum_{i=1}^{n} F(\theta + \epsilon_i)\epsilon_i \tag{2}$$

Notice that this reduces to sampling Gaussian perturbation vectors $\epsilon_i \sim N(0, \mathbf{I})$, evaluating the performance of the perturbed policies and aggregating the results over a batch of samples. The communication overhead between workers is drastically reduced by sharing random seeds, resulting in a highly parallelizable method.

3 IW-ES: Importance Weighted Evolution Strategies

ES samples large batches of data, in the order of thousands of trajectories, which are discarded after performing a single policy update. When coupled with SGD and small step sizes, this translates into a poor data efficiency. Such inefficiency is found in most on-policy RL methods, which are unable to leverage previous experience once the policy is updated.

Inspired by the multiple SGD updates per batch of experience in PPO [21], we propose to modify the ES algorithm to perform several updates to the policy before moving on to collecting a new batch of experience. Should each of these updates be small, it is likely that the population distributions before and after the update will have some overlap, thus making it possible to take more advantage of previous computations and reducing the number of interactions with the environment.

3.1 Formulation

Let $\theta^t \in \mathbb{R}^{|\theta|}$ denote the population mean after t updates, and $\epsilon_i^t \in \mathbb{R}^{|\theta|}$ denote the perturbations for which we computed fitness scores, $F(\theta^t + \epsilon_i^t)$. We can reuse those samples to update θ^{t+k} by relying on Importance Sampling to account for the discrepancy between the distribution of the current population and the distribution from which we are actually sampling:

$$\nabla_\theta F(\theta) \approx \frac{1}{\sigma^2 \sum_i c_i} \sum_{i=1}^{n} F(\theta^t + \epsilon_i^t)(\theta^t + \epsilon_i^t - \theta^{t+k}) c_i \tag{3}$$

where $c_i \in \mathbb{R}$ is the importance weight for the i-th perturbation vector. For perturbations drawn from a multivariate Gaussian distribution with diagonal covariance matrix, the computation of c_i can be decomposed as follows:

$$c_i = \frac{N(\theta^t + \epsilon_i^t - \theta^{t+k}; 0, \sigma^2 \mathbf{I})}{N(\epsilon_i^t; 0, \sigma^2 \mathbf{I})} = \frac{\prod_{j=1}^{|\theta|} N(\theta_j^t + \epsilon_{i,j}^t - \theta_j^{t+k}; 0, \sigma^2)}{\prod_{j=1}^{|\theta|} N(\epsilon_{i,j}^t; 0, \sigma^2)} \tag{4}$$

This process can be repeated iteratively for $k = (0, \ldots, K)$, updating the policy up to $K + 1$ times before collecting a new batch of experience[1]. In this manuscript we consider K as a fixed hyperparameter, although future work will study strategies that optimally adapt K for each batch.

3.2 Scalability analysis

One of the most appealing features of ES is its almost perfect scalability to hundreds of CPU cores, and any modification to the original method should retain such property. We base our analysis on the code released by Salimans et al. [18], which uses a master-worker architecture. The master broadcasts the parameters at the beginning of each iteration, and the workers send back returns after running rollouts with perturbed versions of the policy.

The proposed method requires the computation of importance weights, which has a complexity of $\mathcal{O}(batch\_size \cdot |\theta|)$. If those computations are performed sequentially in the master, the time taken by sequential operations might eclipse the benefits of distributing the rollouts across hundreds of workers. This issue can be alleviated by parallelizing the computation of importance weights across all cores in the node hosting the master process. This was enough to provide a throughput close to the baseline method in most of our experiments, but setups with larger models or batch sizes might benefit from a higher level of parallelization. In that case, the computation of importance weights can be distributed across all workers just like the rollouts are: the master broadcasts the updated parameter vector, and the workers send back the scalar importance weights. Note that this incurs in a very little communication overhead, which is key to achieve an efficient distributed computation.

Another implementation trick that can accelerate the computation of importance weights consists in computing $N(\epsilon_{i,j}^t; 0, \sigma^2)$ for all possible perturbations at the start of training, trading off memory for computation. It takes advantage of the fact that each worker instantiates a large block of Gaussian noise at the start of training, and ϵ_i is obtained by sampling $|\theta|$ consecutive parameters at a random index in the noise block. This trick might provide important savings for large models, as the computation of the denominator in Equation 4 becomes $\mathcal{O}(1)$ instead of $\mathcal{O}(|\theta|)$.

4 Experiments

We implement our method on top of the code released by OpenAI[2]. All experiments run on 720 CPU cores, distributed across 15 machines with 48 cores each. The master process runs on a single core, but the computation of importance weights is parallelized across the 48 cores in the node hosting the master process to accelerate the execution.

We evaluate the method on the Ant-v2 environment[3] in OpenAI Gym [1], which uses the Mujoco physics engine [24]. We use the default hyperparameters provided by Salimans et al. [18] unless otherwise stated. The policy is parameterized by a neural network with two hidden layers of 64 units each and a linear layer that emits continuous actions. Hidden layers are followed by *tanh* non-linearities. Importance weights are clipped at 1 for numerical stability [26, 13]. Following previous works [18, 4], we evaluate the median reward over ~30 stochastic rollouts at each iteration. All reported results are the average over five different runs.

4.1 Effect of the number of IW updates

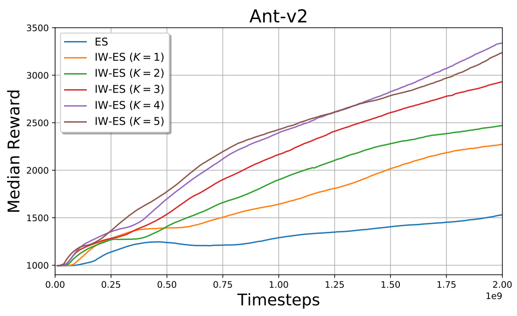
The proposed method relies on a high overlap between the population distributions before and after each update, otherwise the variance of the Importance Sampling estimate might become excessively large. For this reason, we first evaluate the effectiveness of additional updates using a low learning rate of 10^{-4} that prevents large updates to the policy parameters. As depicted in Figure 4a, we observe that additional importance weighted updates provide a faster convergence for a given budget of interactions with the environment. Increased data efficiency also translate in shorter wall-clock times thanks to a reduced computational overhead (Figure 4a). However, performance does not

[1]The first update for each batch always reduces to the original gradient estimate in ES (Equation 2), as $\theta^{t+k} = \theta^t$ for $k = 0$. This is followed by K importance weighted updates.

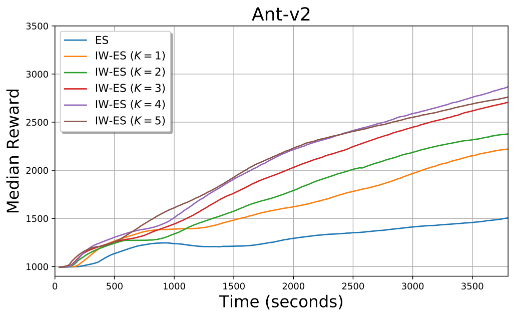
[2]`https://github.com/openai/evolution-strategies-starter/`

[3]Although we provide an extensive analysis of IW-ES only on Ant-v2, we have observed similar behaviors on other complex environments, e.g. Humanoid-v2.

always improve when increasing K, e.g. setting $K = 5$ instead of $K = 4$ results in a performance degradation. This behavior is likely caused by an increased variance in the importance weighted updates for large values of K. These results suggest that IW-ES might benefit from strategies that adapt K for each iteration, omitting updates with excessive variance.



(a)

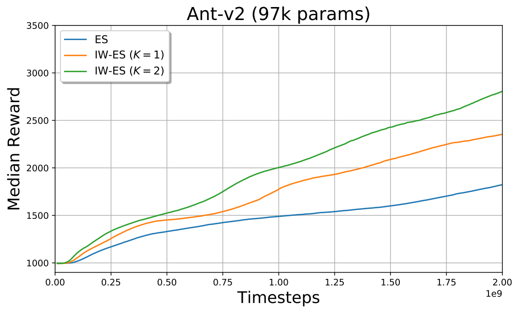


(b)

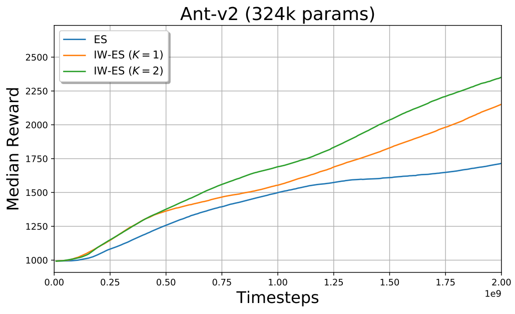
Figure 1: Performance of ES and IW-ES as a function of **(a)** the number of interactions with the environment, and **(b)** wall-clock time. K denotes the number of additional importance weighted updates after each standard update. We observe that additional updates increase the data efficiency of the method in the low learning rate regime, but performing too many importance weighted updates can be detrimental due to an increased variance, e.g. $K = 5$ underperforms $K = 4$. A similar trend is observed in terms of wall-clock time.

4.2 Effect of the model size

A potential source of instability for the proposed method is the computations of importance weights for large models, as they might approach zero or infinity much faster for large values of $|\theta|$ (see Equation 4). We experimentally evaluate whether this hinders the performance of IW-ES by training larger networks, with 256 and 512 units in each hidden layer. These larger models have 97k and 324k parameters, respectively, whereas previous experiments considered a much smaller network with 12k parameters. Results reported in Figure 2 suggest that IW-ES is robust to the number of parameters in the model, as the benefit of adding additional updates per batch are similar to those observed for smaller models.



(a)



(b)

Figure 2: Performance of ES and IW-ES for larger networks with **(a)** 256 units per hidden layer, and **(b)** 512 units per hidden layer.

Figure 3 shows the throughput degradation introduced by IW-ES for each model size and number of importance weighted updates. Since our implementation only leverages 48 of the 720 available CPU cores for computing the importance weights, such computation becomes a bottleneck for larger models and hinders the scalability of the method. This observation motivates the distributed implementation described in Section 3.2, which should accelerate IW-ES considerably for large models thanks to the reduced communication overhead between machines.

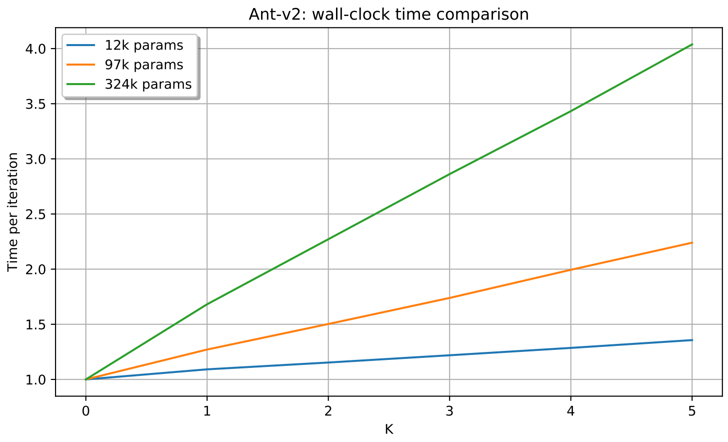
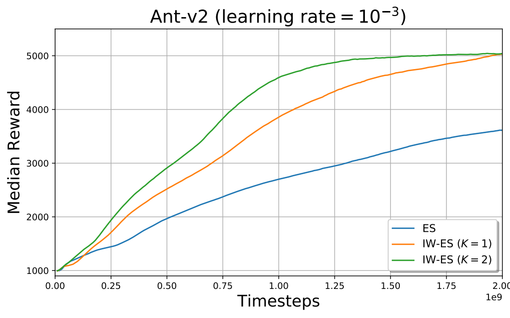


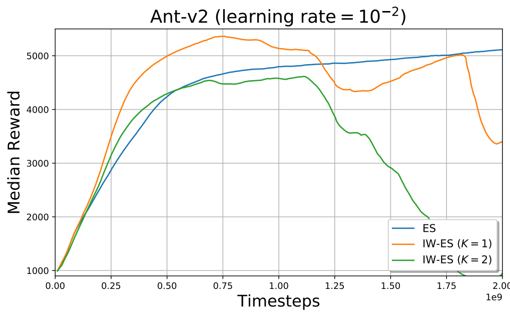
Figure 3: Time per iteration for different values of K, normalized by the time taken by ES (i.e. $K = 0$). Our implementation parallelizes the computation of importance weights only across the CPU cores in the node hosting the master process, which becomes a bottleneck for larger models.

4.3 Effect of the learning rate

ES benefits from larger learning rates than those employed in previous experiments, as they provide faster convergence and thus increased data efficiency, but larger step sizes might increase the variance of IW-ES updates as well due to a larger mismatch between distributions. We evaluate this hypothesis by training policies with larger learning rates of 10^{-3} and 10^{-2}. Results reported in Figure 4 confirm that importance weighted updates not only become less effective with larger learning rates, but can even become unstable and underperform the baseline ES.



(a)



(b)

Figure 4: Performance of ES and IW-ES with learning rates of **(a)** 10^{-3}, and **(b)** 10^{-2}. Larger learning rates reduce the benefits of IW-ES due to a larger variance of the Importance Sampling estimate.

These experiments consider the learning rate as a proxy for controlling the overlap between the distributions before and after each update, which is the actual measure determining the variance of importance weighted updates. Even though a finer grained search over learning rate values could be carried out in order to determine whether IW-ES can outperform ES under optimal hyperparameters, we argue that next steps should aim at controlling the similarity between the distributions before and after each update. For instance, drawing a parallelism with trust region-based methods [20, 9], a constraint could be added on the KL divergence between distributions.

5 Related Work

Some works have proposed extensions or modifications to the original Evolution Strategies (ES) algorithm proposed by Salimans et al. [18]. These include an update rule inspired by genetic algorithms [23] and training a meta-population of agents that optimize both for reward and novelty [4]. The possibility of optimizing non-differentiable functions with ES has also allowed to learn loss functions for RL in a meta-learning setup [11].

The design of data-efficient methods for RL has garnered much research attention, mostly through off-policy methods that can leverage experience collected by policies other than the one being optimized. This advantage, often associated to value-based methods such as Q-learning [25, 12], usually results in an increased data efficiency. Policy-based methods may also leverage off-policy data by accounting for the discrepancy between the behavior and target policies [15, 13]. PPO [21] performs several SGD updates for every batch of collected experience, using Importance Sampling to leverage data collected by an outdated version of the policy, in a similar fashion to our IW-ES update rule.

The IMPALA architecture [6] can scale training of actor-critic methods across many machines to achieve a high troughput, enabling advances in multi-task RL [10]. This is achieved by a combination of algorithmic and engineering advances. Its main drawback comes from a fairly uncommon hardware setup, with each GPU paired with over 100 CPU cores, that may not be feasible to put together within many organizations. In comparison, ES requires from less engineering efforts to achieve high troughputs, thanks to the reduced communication overhead, and its hardware requirements are generally easier to meet.

6 Conclusion & Future Work

We introduced Importance Weighted Evolution Strategies (IW-ES), a variant of Evolution Strategies (ES) [18] that can perform several model updates with a single of batch of data. Under the desired conditions, i.e. when samples from the population distribution before the update are still likely under the updated distribution, IW-ES demonstrated a higher data efficiency than that of ES. For small models, these benefits can be introduced with a small increase in sequential computational load that maintains the scalability of ES. For larger models, we describe how to leverage distributed hardware to distribute further parallelize the added computation and achieve higher throughput rates.

Besides implementing the completely distributed version of IW-ES that can make the most of the available hardware, future work will focus on making IW-ES more resilient to large divergences between distributions that increase the variance of the Importance Sampling estimates. First, an adaptive strategy for K can be designed so that importance weighted updates are made only when their variance is sufficiently low. On the other hand, controlling the divergence after an update through a constraint in the training objective can make IW-ES more robust for large learning rates, and avoid the collapse observed in some experiments. Although applied to policy space instead of parameter space, similar motivations have led to more efficient and stable policy gradient methods [20, 21]. These lines of research may also lead to revisiting the role of σ, which controls the spread of the perturbation vectors in ES, but also plays an important role in determining the importance weights in IW-ES.

Acknowledgments

We would like to thank Francesc Sastre for his help in deploying the code in MareNostrum, as well as the technical support team at the Barcelona Supercomputing Center.

This work was partially supported by the Spanish Ministry of Economy and Competitivity and the European Regional Development Fund (ERDF) under contracts TEC2016-75976-R and TIN2015-65316-P, by the BSC-CNS Severo Ochoa program SEV-2015-0493, and grant 2014-SGR-1051 by the Catalan Government. Víctor Campos was supported by Obra Social "la Caixa" through La Caixa-Severo Ochoa International Doctoral Fellowship program.

References

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

[2] Víctor Campos, Francesc Sastre, Maurici Yagües, Míriam Bellver, Xavier Giró-i Nieto, and Jordi Torres. Distributed training strategies for a computer vision deep learning algorithm on a distributed gpu cluster. *Procedia Computer Science*, 2017.

[3] Jianmin Chen, Xinghao Pan, Rajat Monga, Samy Bengio, and Rafal Jozefowicz. Revisiting distributed synchronous sgd. *arXiv preprint arXiv:1604.00981*, 2016.

[4] Edoardo Conti, Vashisht Madhavan, Felipe Petroski Such, Joel Lehman, Kenneth O Stanley, and Jeff Clune. Improving exploration in evolution strategies for deep reinforcement learning via a population of novelty-seeking agents. In *NIPS*, 2018.

[5] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al. Large scale distributed deep networks. In *NIPS*, 2012.

[6] Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymir Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *ICML*, 2018.

[7] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

[8] Shixiang Gu, Timothy Lillicrap, Zoubin Ghahramani, Richard E Turner, and Sergey Levine. Q-prop: Sample-efficient policy gradient with an off-policy critic. In *ICLR*, 2017.

[9] Nicolas Heess, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, Ali Eslami, Martin Riedmiller, et al. Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286*, 2017.

[10] Matteo Hessel, Hubert Soyer, Lasse Espeholt, Wojciech Czarnecki, Simon Schmitt, and Hado van Hasselt. Multi-task deep reinforcement learning with popart. *arXiv preprint arXiv:1809.04474*, 2018.

[11] Rein Houthooft, Richard Y Chen, Phillip Isola, Bradly C Stadie, Filip Wolski, Jonathan Ho, and Pieter Abbeel. Evolved policy gradients. *arXiv preprint arXiv:1802.04821*, 2018.

[12] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 2015.

[13] Rémi Munos, Tom Stepleton, Anna Harutyunyan, and Marc Bellemare. Safe and efficient off-policy reinforcement learning. In *NIPS*, 2016.

[14] NVIDIA. NVIDIA Tesla V100 GPU architecture. `http://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf`.

[15] Doina Precup, Richard S Sutton, and Satinder Singh. Eligibility traces for off-policy policy evaluation. In *ICML*, 2001.

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

[17] Martin Riedmiller, Roland Hafner, Thomas Lampe, Michael Neunert, Jonas Degrave, Tom Van de Wiele, Volodymyr Mnih, Nicolas Heess, and Jost Tobias Springenberg. Learning by playing - solving sparse reward tasks from scratch. *arXiv preprint arXiv:1802.10567*, 2018.

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

[19] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. In *ICLR*, 2016.

[20] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *ICML*, 2015.

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

[22] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *ICLR*, 2017.

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

[24] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *IROS*, 2012.

[25] Christopher John Cornish Hellaby Watkins. *Learning from delayed rewards*. PhD thesis, King's College, Cambridge, 1989.

[26] Paweł Wawrzyński. Real-time reinforcement learning by sequential actor–critics and experience replay. *Neural Networks*, 2009.

[27] Daan Wierstra, Tom Schaul, Tobias Glasmachers, Yi Sun, Jan Peters, and Jürgen Schmidhuber. Natural evolution strategies. *JMLR*, 2014.

[28] Daan Wierstra, Tom Schaul, Jan Peters, and Juergen Schmidhuber. Natural evolution strategies. In *WCCI*, 2008.