Pareto multi-task deep learning

Pareto Multi-Task Deep Learning

Salvatore D. Riccio[1] Deyan Dyankov[2] Giorgio Jansen[3] Giuseppe Di Fatta[2]
Giuseppe Nicosia[3,4]

[1]School of Mathematical Sciences, Queen Mary University of London, UK
[2]Dept. of Computer Science, University of Reading, UK
[3]Systems Biology Centre, University of Cambridge, UK
[4]Dept. of Biomedical and Biotechnological Sciences, University of Catania, Italy

Abstract

Neuroevolution has been used to train Deep Neural Networks on reinforcement learning problems. A few attempts have been made to extend it to address either multi-task or multi-objective optimization problems. This research work presents the Multi-Task Multi-Objective Deep Neuroevolution method, a highly parallelizable algorithm that can be adopted for tackling both multi-task and multi-objective problems. In this method prior knowledge on the tasks is used to explicitly define multiple utility functions, which are optimized simultaneously. Experimental results on some Atari 2600 games, a challenging testbed for deep reinforcement learning algorithms, show that a single neural network with a single set of parameters can outperform previous state of the art techniques. In addition to the standard analysis, all results are also evaluated using the *Hypervolume indicator* and the *Kullback-Leibler divergence* to get better insights on the underlying training dynamics. The experimental results show that a neural network trained with the proposed evolution strategy can outperform networks individually trained respectively on each of the tasks.

1 Introduction

Deep Learning has successfully been adopted to solve many reinforcement learning problems. Recent results show the huge potential of this technique, with impressive results that are far beyond human abilities [1, 2, 3]. The training of a Deep Neural Network (DNN) consists in changing its parameters to minimize a defined cost function. The backpropagation algorithm [4] is used in most applications to compute the gradient of the loss function for training multilayer feedforward neural networks. Another possible algorithm to train a DNN is Deep Neuroevolution, a class of black-box algorithms inspired by biological evolution, which do not require the computation of the gradient. It was experimentally shown that networks trained with Deep Neuroevolution obtain results that can compete against gradient-descent deep learning algorithms in difficult reinforcement learning problems [5, 6, 7, 8]. Population-based Neuroevolution algorithms can be adopted to find the optimal solution by exploring the space of both parameters and hyperparameters [9] as well as the best neural network architecture [10].

Deep Neuroevolution is a population-based algorithm, so it can be naturally extended to multi-task learning. Some works in this direction addressed multi-task learning [8, 11], whereas other implemented a multi-objective strategy on a single task [12]. We propose a Deep Neuroevolution method for Multi-Task Multi-Objective problems, which is based on Evolution Strategy (MTMO-ES). This novel approach is used to optimize at the same time different tasks and different objectives with a single neural network associated with a single set of parameters. Multi-task learning aims at training a neural network to master multiple tasks simultaneously.

The proposed approach is tested on Atari 2600 games, a common benchmark for reinforcement learning algorithms. A network trained with MTMO-ES is shown to achieve better results than previous single-task single-objective deep reinforcement learning models trained individually on each task. A network trained for multi-task learning is expected to be inherently more general than one trained on a single task only.

Furthermore, we study the outcome of the network using two performance indices, the Kullback-Leibler divergence and the hypervolume indicator, that can be computed at each iteration, showing the underlying training dynamics. The rest of the paper is organized as follows: in Section 2 we discuss the theory behind the MTMO-ES algorithm. In Section 3 we present the analysis based on the hypervolume indicator and the Kullback-Leibler divergence, providing interesting insights of how to compute them for every iteration. Experimental results are shown in Section 4. Finally, conclusions are drawn in Section 5.

2 Multi-Task Multi-Objective Evolution Strategy

The goal of deep learning algorithms is to train a network so that its parameter vector θ maximizes a specific utility function $u(\theta)$, which depends on the output obtained by the neural network associated with θ. For the sake of conciseness, u will be used instead of $u(\theta)$ unless otherwise stated. Such an optimization problem can be solved by Evolution Strategies (ES) [13], a class of nature-inspired black-box optimization methods [14]. In general, an Evolution Strategy algorithm works as follows. Starting from one or more parents, a number of offspring is generated using mutation, crossover, or other techniques. Offspring are typically evaluated on a cost function (fitness function) and a new set of parents are selected according to a chosen criterion for the next generation.

The aim of this work is to train a neural network with evolution strategy, so that the trained network achieves good performance in all tasks using a single set of parameters. The best parent, P, selected in the last iteration is the parameter vector of the trained network.

Offspring at iteration k are generated from a gaussian distribution centered in $\theta_P(k)$. To increase the robustness, we perform mirrored sampling (i.e., two offspring are generated symmetrically or mirrored with respect to their parent). In practice, each offspring is associated with another one that is symmetric with respect to the parent values $\theta_P(k)$. Evaluating the offspring allows to estimate the gradient without resorting to differentiation. This is useful especially when the rewards are sparse, which is a property that reinforcement learning problems commonly have. Multi-Task Multi-Objective based on Evolution Strategies (MTMO-ES) is based on this evolutionary approach and the details of the proposed method are described in the reminder of this section. At iteration 0, the parameter vector $\theta_P(0)$ (subscript P stands for Parent, whereas 0 is the iteration) is randomly initialized. In a single task, the new parent parameter vector can be computed at every iteration k as shown in eq.(1):

$$\theta_P(k+1) = g(\theta_P(k), \nabla_\theta f(\theta(k)))\ ,\ \theta(k) \in \Theta_k \tag{1}$$

where $\nabla_\theta f(\theta(k))$ is the gradient of the score function estimator, f is the distribution from which the chosen feature is obtained using $\theta(k)$, g is a chosen optimizer (e.g. Adam or stochastic gradient descent), and Θ_k is the set of offspring generated at iteration k. It is assumed that an utility u depends on the measured feature f for the single-task scenario. If there are two or more goals, the corresponding multi-objective optimization problem over $m \in \mathbb{N}$ goals is stated as $\text{argmax}_\theta\{u_1(\theta), u_2(\theta), \ldots, u_m(\theta)\}$ We introduce a change to eq.(1) to address both the multi-task and the multi-objective problem. Let $\mathcal{T}$ be a finite set of chosen tasks (i.e.: $|\mathcal{T}| = t \in \mathbb{N}$), $\{u_1, u_2, \ldots, u_m\}$ is the set of goals, $\{f_1, f_2, \ldots, f_n\}$ is the set of measurable features. Every different task must be associated with at least one feature, so $t \leq n$. Note that, in general for a given i, f_i is not necessarily associated with τ_i. The association between features and tasks is explicitly defined. Let $\delta_{ij} = \{0, 1\}$ be a binary variable that is 1 if and only if f_i is evaluated on task τ_j. It is also required that every feature f_i is associated with at most one task τ_j: $\sum_{j=1}^{t} \delta_{ij} = 1\ ,\ \forall i \in \{1, \ldots, n\}$. Instead of computing only one gradient on one feature f_i evaluated on one task τ_j, we compute n gradients based on n features evaluated on t tasks. Every i-th gradient is then multiplied by a scalar α_i to obtain the overall Multi-Task Multi-Objective (MTMO-ES) gradient:

$$\theta_P(k+1) = g\left(\theta_P(k), \Sigma_{i=1}^{n} \alpha_i \nabla_\theta f_i(\theta(k))\right)\ ,\ \theta(k) \in \Theta_k \tag{2}$$

$$\sum_{i=1}^{n} |\alpha_i| = 1\ ,\ \alpha_i \in \mathbb{R}\ \forall i \tag{3}$$

$$\sum_{i=1}^{n} |\alpha_i| \delta_{ij} = \frac{1}{t}\ ,\ \forall j \in \{1, \ldots, t\} \tag{4}$$

It can be recognized that the MTMO-ES gradient is a linear combination of all single-task gradients. The condition in eq.(3) is required to avoid a change in the modulus of the MTMO-ES gradient, provided that all single-task gradients are normalized. The condition in eq.(4) avoids biases towards one of the tasks. In other terms, with eq.(2)-(3) it is possible to rotate the final gradient accordingly to the measured features. This way it is possible to force the evolution of the vector θ towards a common direction that maximizes every goal. Before computing the gradient for every feature f_i, offspring scores are rank-normalized. Without this step, the network can get easily stuck in a local maximum within the first iterations [7]. The computation time required to compute every gradient in eq.(2) is t times the one required for eq.(1), because every mutation $\theta \in \Theta_k$ needs to be evaluated on every task. However, this algorithm can be easily parallelized exploiting the same approach implemented for the single-task algorithm [6]. With this formulation it is possible to address both multi-task and multi-objective learning at the same time using Evolution Strategies. It is also easy to minimize some goals and to maximize others. It is enough to associate with a negative α the features that have to be minimized.

3 Hypervolume and Kullback-Leibler Divergence

Hypervolume. Given a set of solutions Θ, its associated Pareto front is defined as the set P such that $P(\Theta) = \{\theta' \in \Theta \mid \nexists \theta \in \Theta : \theta \preceq \theta'\}$ where, using the standard notation, $\theta \preceq \theta'$ means that θ dominates the solution θ'. The hypervolume indicator [15] h is the volume between the Pareto front P and the reference point. This latter point is a parameter that must be carefully chosen to avoid distorted results [16]. In this work, the easiest and most natural choice is to select the origin as the reference point. The Pareto front can be computed at every iteration given the set of offspring Θ_k at iteration k. This means that also the hypervolume indicator h can be computed as a function of k. The difference between the maximum value of two different goals u_i and u_j can be high. To avoid biases in the hypervolume indicator, it is important to normalize each goal with respect to its maximum value over all iterations. This indicator shows how much effectively the network is learning to achieve good results on every goal. It can be analytically computed if $t = 2$. The computation is nontrivial for higher dimension and good estimations can be obtained with efficient algorithms [17].

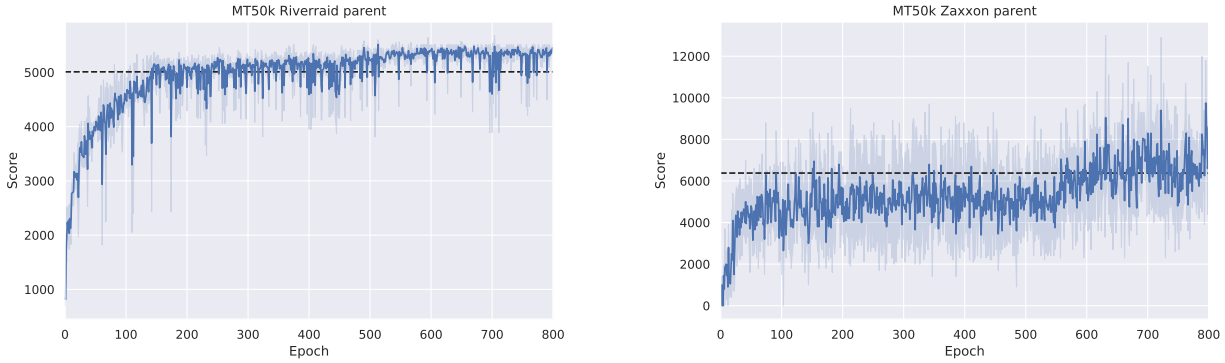

Figure 1: Multi-task single-objective ES trained on River Raid and Zaxxon, evaluated on River Raid (left) and on Zaxxon (right) for 800 iterations. Elite mean scores, with minimum and maximum score per iteration. The dashed lines are the results obtained by ES single task as shown in [6]. The network trained with the new approach is able to perform better in both tasks, given that it is trained for a sufficient number of iterations.

Kullback-Leibler Divergence. Given two discrete distributions P and Q defined over the same support $\mathcal{X}$, the Kullback-Leibler divergence is described by eq.(5):

$$D_{KL}(P, Q) = \sum_{x \in \mathcal{X}} p(x) ln \frac{p(x)}{q(x)} \tag{5}$$

In this work, distributions P and Q are associated with a given goal u and they can change as a function of different samples of the set Θ. For hard reinforcement learning problems, those distributions are unknown and cannot be computed analytically. A viable approach is to exploit a frequentist approach, using relative frequencies on sampled values to get an estimate of $p(x)$ and $q(x)$. In practice, $\theta_P(k)$ is evaluated $N \in \mathbb{N}$ times to get N i.i.d. samples from $f(\theta_P(k))$. The interval from the minimum and the maximum values in all iterations is partitioned into $M \in \mathbb{N}$ subintervals with same length. Each subinterval is associated with the number of perturbations that have a score within the subinterval range. To overcome numerical issues coming from some subintervals having zero frequency, Laplace smoothing is applied to the data before computing D_{KL}. This smoothing technique consists in adding a small ϵ to each subinterval, followed by a normalization of P and Q to 1. The Kullback-Leibler divergence is computed using the distribution estimated at the current iteration k for P, and the one at a reference iteration $\bar{k}$ for Q. Some possible choices for $\bar{k}$ are the first iteration, the last iteration, or $\bar{k} = k - \Delta k$, where $\Delta k \in \mathbb{Z}$ is a fixed lag. The Kullback-Leibler divergence is also called information gain. It is a measure of the amount of information that the algorithm is able to extract going from Q to P [18]. When P and Q are identical, $D_{KL}(P, Q) = 0$. On the other hand, if for all $x \in \mathcal{X}$ $p(x) \neq 0$ and $q(x) = 0$, then the two distributions are disjointed. In this case, the D_{KL} will be large and it depends both on ϵ and on the distribution P. In practice, a large D_{KL} means that the two distributions are different, whereas a small value implies similar distributions.

4 Experimental Analysis and Results

All experiments are run on a single deep neural network. The architecture is the same used in [1], which consists of three convolutional layers, followed by a fully connected layer. The output layer is fully connected too, and

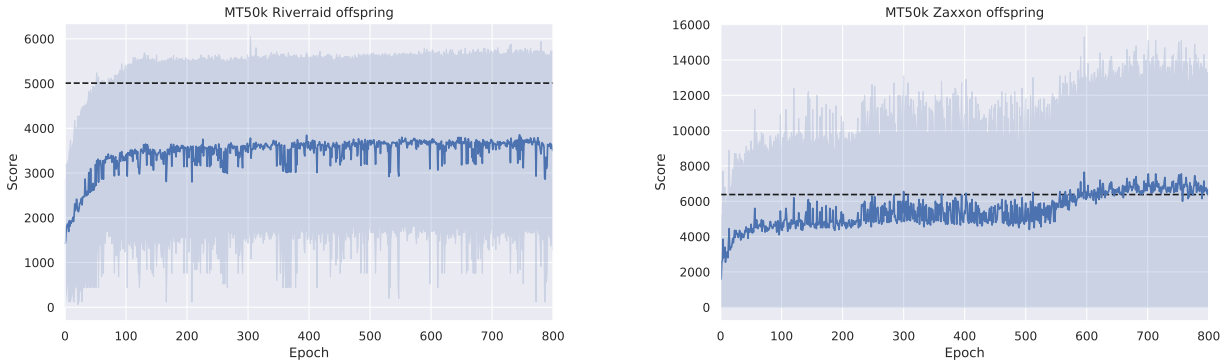

Figure 2: Multi-task single-objective ES trained on River Raid and Zaxxon, evaluated on River Raid (left) and on Zaxxon (right) for 800 iterations. Offspring mean scores, with minimum and maximum score per iteration. The dashed lines are the mean elite scores obtained by ES single task in their respective game as shown in [6]. Interestingly, even the mean score obtained by Zaxxon offspring is slightly better than the mean elite score obtained with single-task ES.

each node is respectively associated with one out of the 18 valid actions. Our implementation is an extension of the work of Conti et al. [7] and inherits its parallelizability. Indeed, all simulations are run on a single machine using parallel GPU computing. Although the approach is general and can be applied to a large number of tasks, in this work we limited the analysis to scenarios with a few tasks for the sake of simplicity. First, the multi-task approach is tested. The two Atari games Zaxxon and River Raid are chosen from OpenAI Gym [19] because they share similar game dynamics. Scores are compared with those obtained with a single-task ES approach (River Raid: 5009.0, Zaxxon: 6380.0) [6] and with IMPALA, a multi-task algorithm based on V-trace [11] (River Raid: 2850.15, Zaxxon: 6497.00). The mean of the scores obtained by the *parent candidate*

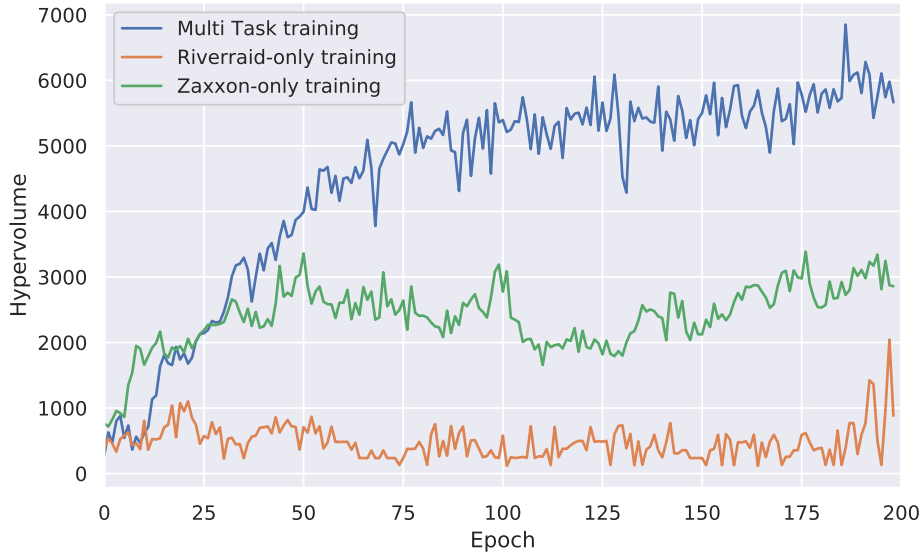

Figure 3: Comparison between three training strategies: multi-task, single-task trained on one game, single-task trained on the other game. A network trained on a single task cannot master both games, whereas the network trained with the new approach is able to play well both games.

solutions evaluated over 200 episodes is shown in Fig. 1. After an initial convergence toward a local minimum, the network is able to find new strategies and to outperform previous results on both tasks. Fig. 2 shows the mean score obtained evaluating $5,000$ offspring for every iteration. The range between minimum and maximum value is large, which means that the algorithm is exploring new possible solutions at every iteration. We test other two runs, evaluating the score on both tasks for a single network trained with single-task single-objective ES. The hypervolume indicator is computed for three independent simulations. Results are shown in Fig. 3. This plot shows that the Pareto front obtained with the multi-task ES algorithm covers a larger area than the

other two, which means that the new algorithm is finding strategies able to master both tasks at the same time. Using the single-task algorithm, the Pareto front moves along one of the two axes. With the multi-task algorithm, the direction of the Pareto front is common to both axes. The Kullback-Leibler divergence is then computed on parents score. All the results obtained using different lags Δk are plotted in Fig. 4. When the divergence is evaluated with respect to the initial distribution, we expect a convergence towards a large value. This is exactly the behavior showed in the figure, which means that the distribution at the end is completely different than the original one. When using a fixed lag Δk, an increasing smooth line shows that the network is changing its distribution faster and faster. This is usually followed by a negative slope, which means that the distribution is still changing, but slower than before. In the last iterations the line is noisy, showing that the two distributions have similar mean values, so the network is not really learning anymore. We also investigate how many strategies found during multi-task training are better in both tasks than the best elite score found with a single-task approach. This result is plotted in Fig. 5, which shows the percentage of the set of parameters per iteration that can outperform results obtained with single-task ES. It is notable that up to an impressive 20% (i.e.: around $1,000$) offspring during the last iterations are good candidates to be the next parent. Multi-task ES is able to find $2-3\%$ of offspring with outstanding results even in the first hundred iterations. Finally, single-task multi-objective ES is tested. We want to maximize the score of Zaxxon using two objectives: the score itself and the number of played frames. The idea is that explicitly forcing the network towards long-run strategies can help finding a better solution. The results of this new multi-objective strategy are compared with the ones of the single-objective ES strategy and the plots are shown in Fig. 6. Although the new method shows a slower improvement in the first iterations, at the end it achieves better results with an increasing trend whereas the single-objective version seems to have converged to a local maximum.

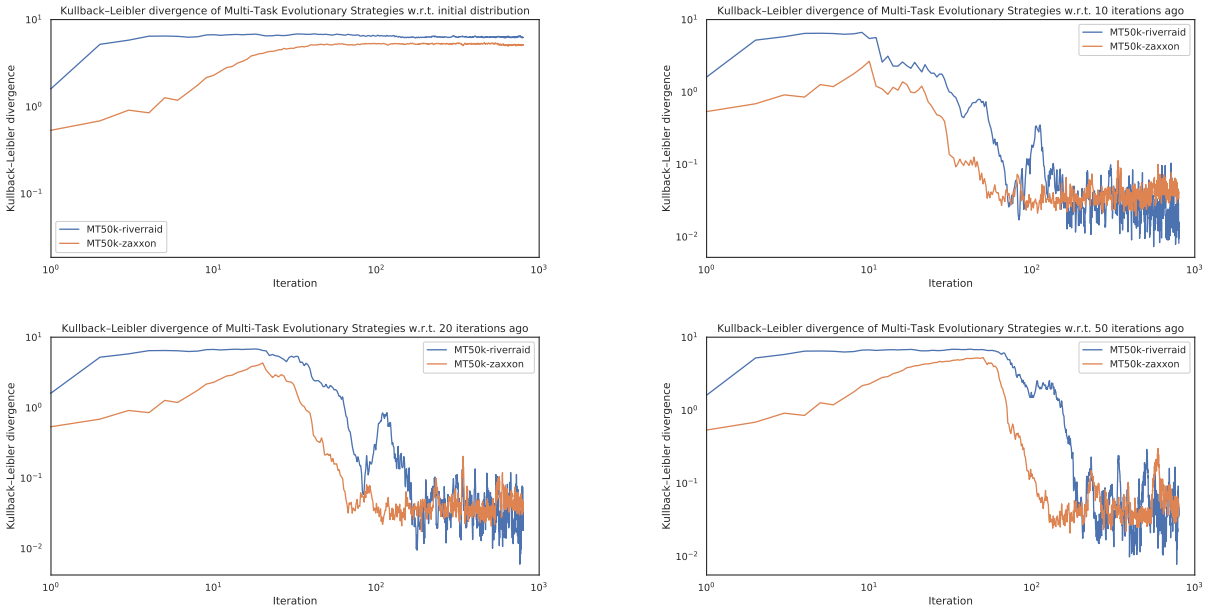

Figure 4: D_{KL} computed on parent scores with respect to the first iteration (first row, left), to the previous 10 iterations (first row, right), to the previous 20 iterations (second row, left), to the previous 50 iterations (second row, right), shown as a log-log plot. $M = 100$ is the chosen number of subintervals. The index is computed separately on the two tasks, namely River Raid and Zaxxon. When the index is evaluated with respect to the first iteration, the two lines converge towards a value different from zero. Indeed, the sampled distribution obtained at iteration k diverges from the initial one for both tasks. When the index is evaluated using a fixed delay Δk, it is high for both tasks within the first hundred iterations, showing that the network is learning. Then the values decrease because the difference between the distributions becomes smaller.

5 Conclusions

Multi-Task Multi-Objective Evolution Strategy (MTMO-ES) is a novel training algorithm for Deep Neural Networks that can simultaneously address both multi-task and multi-objective problems. Tests on some Atari 2600 games show that a neural network trained with this evolution strategy can obtain excellent results in more than one task or one objective, using only one set of parameters. It is indeed remarkable that a single network with a single set of parameters can outperform networks trained respectively on one task or one objective only. Thanks to its underlying structure, this algorithm can be easily parallelized using workers on many CPUs or GPUs.

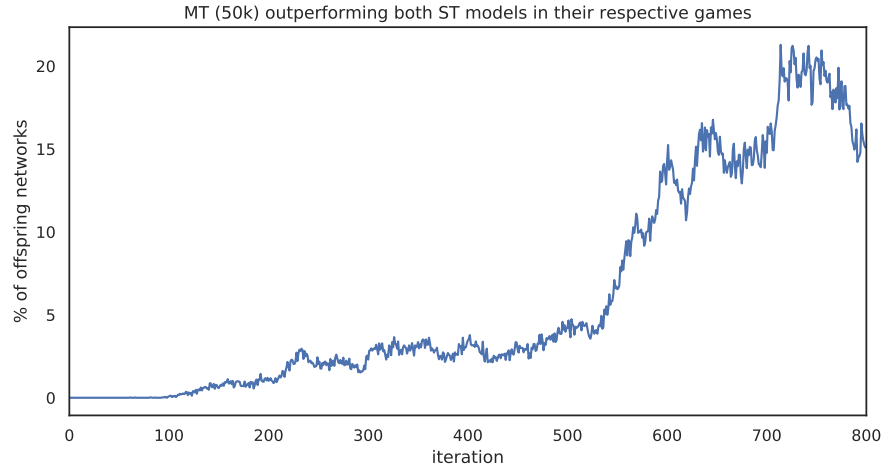

Figure 5: Percentage of offspring that can outperform both single-task ES state-of-the-art scores for every iteration. Up to 20% of the networks generated for every iteration obtain better results than the scores obtained with single-task ES.

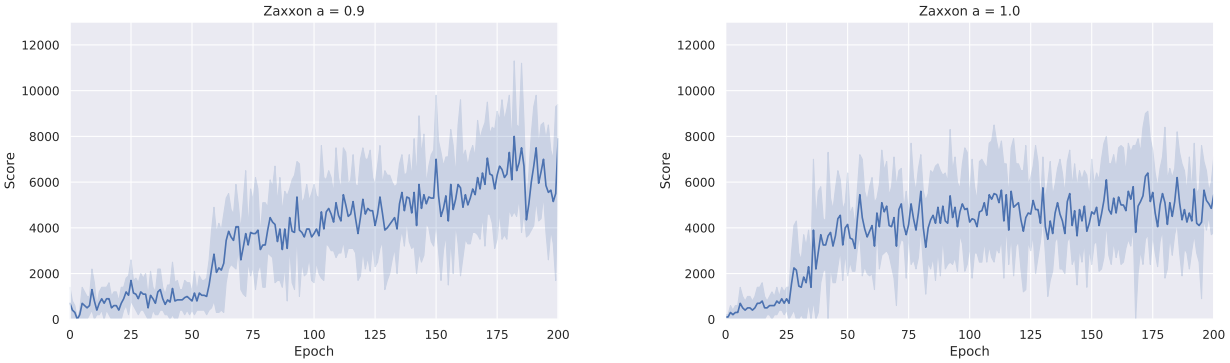

Figure 6: Mean score obtained from 20 evaluations per iteration using single-task multi-objective ES on Zaxxon. The area shows the score range (from minimum to maximum) per iteration. The algorithm is maximizing both the score and the number of frames (left), compared with single-task single-objective ES where the only objective is the score (right). The score improves thanks to the additional requirement on the number of frames.

In principle, an arbitrarily large number of tasks can be chosen without affecting the total time required. In practice, the overhead will become not negligible after a certain number of tasks, and the algorithm will take longer. The hypervolume indicator and the Kullback-Leibler divergence, when computed at each iteration, can give useful insights on the learning dynamics. MTMO-ES is a general optimization method and does not require prior knowledge of the environment dynamics. For this reason, we believe it can be easily adopted in many diverse domains and obtain interesting results with relatively low implementation effort.

References

[1] V. Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (Feb. 2015), pp. 529–533.

[2] D. Silver et al. "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play". In: *Science* 362 (2018), pp. 1140–1144.

[3] O. Vinyals, I. Babuschkin, Czarnecki, and W. et al. "Grandmaster level in StarCraft II using multi-agent reinforcement learning". In: *Nature* 575 (2019), pp. 350–354.

[4] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. "Learning representations by back-propagating errors". In: *Nature* 323 (Oct. 1986), pp. 533–536.

[5] M. Hausknecht, J. Lehman, R. Miikkulainen, and P. Stone. "A Neuroevolution Approach to General Atari Game Playing". In: *Computational Intelligence and AI in Games, IEEE Transactions on* 6 (Dec. 2014), pp. 355–366.

[6] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever. "Evolution Strategies as a Scalable Alternative to Reinforcement Learning". In: *arXiv e-prints*, arXiv:1703.03864 (Mar. 2017), arXiv:1703.03864.

[7] E. Conti, V. Madhavan, F. Petroski Such, J. Lehman, K. O. Stanley, and J. Clune. "Improving Exploration in Evolution Strategies for Deep Reinforcement Learning via a Population of Novelty-Seeking Agents". In: *NeurIPS 2018, Montreal, Canada* (2018).

[8] D. Dyankov, S. D. Riccio, G. D. Fatta, and G. Nicosia. "Multi-task Learning by Pareto Optimality". In: *Machine Learning, Optimization, and Data Science - 5th International Conference, LOD 2019, Siena, Italy, September 10-13, 2019, Proceedings*. 2019, pp. 605–618.

[9] M. Jaderberg et al. "Population Based Training of Neural Networks". arXiv:1711.09846. Nov. 2017.

[10] K. Stanley, J. Clune, J. Lehman, and R. Miikkulainen. "Designing neural networks through neuroevolution". In: *Nature Machine Intelligence* (Jan. 2019).

[11] L. Espeholt et al. "IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures". In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by J. Dy and A. Krause. Vol. 80. 2018, pp. 1407–1416.

[12] T. G. Tan, J. Teo, and C. On. "Single- versus Multiobjective Optimization for Evolution of Neural Controllers in Ms. Pac-Man". In: *International Journal of Computer Games Technology* 2013 (Feb. 2013).

[13] I. Rechenberg. "Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution". PhD thesis. Technical University of Berlin, Department of Process Engineering, 1971.

[14] K. De Jong. *Evolutionary Computation – A Unified Approach*. The MIT Press, Jan. 2006.

[15] E. Zitzler and L. Thiele. "Multiobjective optimization using evolutionary algorithms — A comparative case study". In: *Proceedings of the 30th International Conference on Machine Learning*. Ed. by E. A.E., B. T., S. M., and S. HP. Vol. 1498. 1998, pp. 292–301.

[16] A. Auger, J. Bader, D. Brockhoff, and E. Zitzler. "Theory of the Hypervolume Indicator: Optimal μ-Distributions and the Choice of the Reference Point". In: *FOGA*. Orlando, Florida, USA, 2009, pp. 87–102.

[17] C. M. Fonseca, L. Paquete, and M. López-Ibáñez. "An Improved Dimension-Sweep Algorithm for the Hypervolume Indicator". In: *2006 IEEE International Conference on Evolutionary Computation* (2006), pp. 1157–1163.

[18] G. Stracquadanio and G. Nicosia. "Computational energy-based redesign of robust proteins". In: *Computers and Chemical Engineering* (2010).

[19] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. *OpenAI Gym*. 2016.