IFAC World Congress 2020

Fine-tuning Deep RL with Gradient-Free Optimization [⋆]

Tim de Bruin [*] Jens Kober [*] Karl Tuyls [**] Robert Babuška [*]

[*] *Cognitive Robotics Department, Delft University of Technology, Delft, The Netherlands (e-mail: {t.d.debruin, j.kober, r.babuska}@tudelft.nl).*
[**] *Deepmind, Paris, France (e-mail: karltuyls@google.com)*

Abstract: Deep reinforcement learning makes it possible to train control policies that map high-dimensional observations to actions. These methods typically use gradient-based optimization techniques to enable relatively efficient learning, but are notoriously sensitive to hyperparameter choices and do not have good convergence properties. Gradient-free optimization methods, such as evolutionary strategies, can offer a more stable alternative but tend to be much less sample efficient. In this work we propose a combination, using the relative strengths of both. We start with a gradient-based initial training phase, which is used to quickly learn both a state representation and an initial policy. This phase is followed by a gradient-free optimization of only the final action selection parameters. This enables the policy to improve in a stable manner to a performance level not obtained by gradient-based optimization alone, using many fewer trials than methods using only gradient-free optimization. We demonstrate the effectiveness of the method on two Atari games, a continuous control benchmark and the CarRacing-v0 benchmark. On the latter we surpass the best previously reported score while using significantly fewer episodes.

Keywords: Reinforcement Learning, Deep Learning, Optimization, Neural Networks, Control

1. INTRODUCTION

Deep Reinforcement Learning (DRL) is a learning framework that has recently enjoyed significant successes (Silver et al., 2018; Mnih et al., 2015; Bansal et al., 2017; Andrychowicz et al., 2018). In this framework, Reinforcement Learning (RL) is used to train Deep Neural Network (DNN) controllers. DNN control policies are especially promising for tasks that require making decisions directly from natural data such as images. This is because the functional decomposition of deep neural networks mirrors the hierarchical nature of the physical processes that generate the sensor data (Lin et al., 2017), making them more statistically and computationally efficient than alternatives that do not have a hierarchical structure (Bengio et al., 2013).

While the DNN controllers are often trained end-to-end to map raw sensor data to actuator commands, the hierarchy of functions that is encoded by their layers could be thought of as representing two distinct sub-functions. The first is a mapping from the high-dimensional sensor data to a concise lower-dimensional representation of the task-relevant aspects of the environment state. The second is a mapping from this state representation to the action that needs to be taken in that state to accomplish the task at hand.

To learn these functions, stochastic gradient-based optimization techniques are most commonly used. When good enough estimates of the true parameter gradients can be obtained, these techniques can efficiently find good values for the network parameters. In this work we will take the view that for the mapping from observation to state representation this is indeed often the case. For an interesting class of problems, many—if not most—of the parameters of the DNN controller will be used to encode this mapping. This mapping can be learned either implicitly through end-to-end reinforcement learning, or explicitly by using state representation learning objectives (e.g. Jonschkowski and Brock, 2015; Lange et al., 2012; Finn et al., 2016; de Bruin et al., 2018b). Intuitively, the sensory observations are a direct result of the latent state of the environment. While we do not have access to the true state, there are many objectives that will allow us to learn to infer the task-relevant aspects of this state relatively easily. These include reconstructing observations, predicting action-dependent changes to the state representation or observations, predicting the instantaneous rewards, and more. Even when hand-crafted state representations are available, learned state representations can sometimes enable better policies (Levine et al., 2016).

The mapping from the state representation to the optimal action in that state often requires fewer parameters. However, this mapping can still be much harder to learn. This is because the effect of any single action on the eventual task performance is often rather small, and there might be delays before the consequences of actions become apparent. Getting high-quality estimates of the gradients

[⋆] This work is part of the research programme Deep Learning for Robust Robot Control (DL-Force) with project number 656.000.003, which is (partly) financed by the Netherlands Organisation for Scientific Research (NWO).

of the task performance with respect to the network parameters through the chosen actions is therefore difficult. These difficulties show up in different forms, depending on how the parameter gradients are obtained. Techniques using policy rollouts are able to get unbiased estimates of the policy gradient, but suffer from very high variance, while techniques using bootstrapping suffer from biased gradients (Schulman et al., 2015b; Marbach and Tsitsiklis, 2003). For both extremes, as well as the methods that trade off bias and variance by using both rollouts and bootstrapping, the low-quality parameter gradients can make the stochastic gradient optimization process diverge (Sutton and Barto, 2018; Henderson et al., 2017; Gu et al., 2017). Many different strategies have been used to deal with these problems, generally trading learning speed and data efficiency for learning stability. Examples include target networks (Mnih et al., 2015), experience replay buffers (Lin et al., 2017; Mnih et al., 2015), trust region updates (Schulman et al., 2015a, 2017; Wang et al., 2017) and very large batch sizes (Bansal et al., 2017). While these techniques ameliorate the problem, DRL is still notoriously sensitive to hyperparameter tuning and prone to divergence (Henderson et al., 2017).

An alternative to these attempts to deal with low-quality parameter gradients is to use gradient-free optimization techniques, such as Evolutionary Strategies (ES) (e.g. Koutník et al., 2013; Salimans et al., 2017). Rather than trying to assign credit to the policy parameters through the individual actions that were taken, gradient-free techniques assign credit to parameter vectors directly based on entire trajectories. These techniques tend to be much less data efficient than gradient-based techniques, but they can lead to more stable convergence of the policy performance. They also have other benefits like their ability to optimize policies that need to make decisions at high sampling frequencies, and the fact that their training is highly parallelizable (Salimans et al., 2017).

In this work we combine the desirable aspects of both gradient-based and gradient-free optimization techniques for DNN controllers. We start with a gradient-based phase in which we use standard deep reinforcement learning techniques. This allows us to quickly learn a policy that is good enough to collect relevant training data and learn a state representation. We then freeze the state encoder part of the policy, and tune the final action-selection parameters further using a gradient-free technique. Since we are only tuning relatively small number of parameters, we use the CMA-ES algorithm (Hansen and Ostermeier, 2001). This algorithm is not only relatively sample efficient for an ES algorithm, but also includes a natural way of decaying the amount of exploration (Stulp and Sigaud, 2012). This makes it possible to perfect the policy while reducing the probability of poor performances. Note that reducing the amount of exploration while training the entire policy—including the state encoder—using gradient-based optimization can lead to over-fitting and destabilize the learning process (de Bruin et al., 2018a).

2. RELATED WORK

Our method is perhaps most closely related to that of Ha and Schmidhuber (2018). In their work, a random policy is used to collect training data, which is used to train a state encoder using state-representation learning objectives. Then, the action-selection subnetwork of the policy is trained from scratch using CMA-ES. While we consider the use of state representation learning objectives *in addition* to reinforcement learning, we rely mainly on RL to learn the whole policy during the gradient-based learning phase. This not only results in more relevant training samples (enabling a better representation to be learned (Pérez Dattari et al., 2019)), but also a good initialization of the action-selection parameters. We show in Section 5 how these changes allow us to solve the CarRacing-v0 task using forty times fewer episodes, while still obtaining a considerably better final policy.

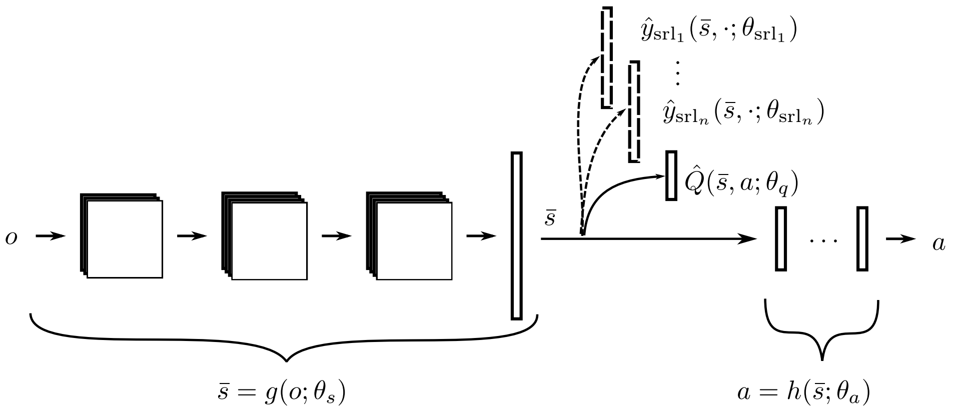
Evolutionary Strategies (ES) have also been used to optimize all parameters of deep neural network controllers (e.g. Hausknecht et al., 2014; Salimans et al., 2017). These methods are able to scale across many CPUs in an efficient way. They are however not very sample efficient. The fact that we only optimize a small number of parameters using ES helps us to limit the sample inefficiency of the method. It also allows us to use CMA-ES which does not scale to large parameter vectors, but is able to optimize small parameter vectors in a more sample-efficient manner than other ES strategies (Hansen and Ostermeier, 2001). We additionally show in Section 5 that even when we do optimize all parameters of a policy using ES, initializing the parameters using a short gradient-based optimization phase before starting the ES optimization can help to both speed up the learning as well as improve the eventual performance.

The idea that the last layer of a DNN is harder to train using gradient-based reinforcement learning than the preceding layers was previously explored by Levine et al. (2017). In their work, the instability of the DQN method was limited by performing least-squares updates of the parameters of the final layer in addition to the standard gradient updates.

Plappert et al. (2018) also consider the final layers of the policy to represent the action selection part and add noise to these parameters to enable exploration. We use their technique during the gradient-based learning phase and update only these parameters through CMA-ES during the policy fine-tuning phase. Here, the CMA-ES can be understood as an optimization algorithm for both the exploration policy as well as the policy parameters (Stulp and Sigaud, 2012). Decaying exploration is desirable when fine-tuning the policy, but continuing to update the entire network using gradient-based methods can lead to divergence due to a lack of training data variation (de Bruin et al., 2018a).

3. PRELIMINARIES

We consider a reinforcement learning setting in which an agent takes actions $a \in \mathcal{A}$ to change the state $s \in \mathcal{S}$ of a dynamical environment. At every discrete time step k, the agent receives sensory observations $o_k = \mathcal{F}(s_k)$ of the state, based on which it chooses an action $a_k = \pi(o_k; \theta)$ which changes the state according to the environment transition dynamics: $\mathcal{P}(s'|s, a)$. In this work, π is a DNN control policy parameterized by θ. A reward is given based on the transition: $r_k = \rho(s_k, a_k, s_{k+1})$. We assume that the reward comes from the environment, which has access to the true



$\hat{y}_{\mathrm{srl}_1}(\bar{s},\cdot;\theta_{\mathrm{srl}_1})$
$\hat{y}_{\mathrm{srl}_n}(\bar{s},\cdot;\theta_{\mathrm{srl}_n})$
$\hat{Q}(\bar{s},a;\theta_q)$
o
$\bar{s}$
a
$\bar{s} = g(o;\theta_s)$
$a = h(\bar{s};\theta_a)$

Fig. 1. We learn DNN control policies that map observations o to actions a: $a = \pi(o;\theta)$. We consider the first n layers of m-layer DNN to encode a mapping from observations to a state representation $\bar{s}$: $\bar{s} = g(o;\theta_s)$. The parameters θ_s that encode this mapping are learned through gradient-based optimization by fitting value functions, optionally supplemented with state representation learning objectives. The final $m - n$ layers encode the mapping from the state representation to actions: $a = h(\bar{s};\theta_a)$. The parameters θ_a are initialized during the gradient-based learning phase and then fine-tuned during a gradient-free optimization phase.

state s. For the simulated control problems considered in this paper, this is indeed the case. For some real problems, the reward might be a function of the observation o instead. We consider episodic tasks, where the end of the episode is indicated with the terminal indicator $\mathbb{T}$, which is 1 for terminal states and 0 otherwise. The aim is to maximize the return $R = \sum_{k=0}^{K} \gamma^k r_k$ where K is the final time-step of the episode. We consider the true objective to be the undiscounted return ($\gamma = 1$).

The aim of learning will be to find the parameter vector θ that corresponds to a policy that maximizes the expected returns R from the initial states. As is common practice, we will optimize for a surrogate objective with $\gamma < 1$ during the gradient-based optimization of θ, which makes the optimization easier (Marbach and Tsitsiklis, 2003; Schulman et al., 2015b).

4. METHOD

In this work, the training of the deep neural network controller consists of two distinct phases; a gradient-based optimization phase and a gradient-free phase. The aim of the initial gradient-based learning phase is twofold: we want to efficiently learn a state representation that can be used for control and we want to find a good initialization of the action-selection subnetwork of the policy. After this phase is complete, we will trade learning speed for stability by further tuning the action-selection parameters using a gradient-free evolutionary strategy. This will allow for stable convergence of the policy performance, while optimizing for the true undiscounted return objective, rather than the discounted return surrogate objective.

4.1 DNN Controllers

The control policy that maps observations o to actions a is parameterized as a deep neural network with parameters θ: $a = \pi(o;\theta)$. The network consists of m layers, of which we consider the first n to represent the state encoder, which maps observations to a state representation: $\bar{s} = g(o,\theta_s)$. The final $m-n$ layers are considered to represent the action selection subnetwork, which encodes the mapping from this state representation to the policy action: $a = h(\bar{s};\theta_a)$ with $\pi(o;\theta) = h(g(o;\theta_s);\theta_a)$. In addition to the policy action, several other predictions can be made based on the state representation $\bar{s}$. In the DRL methods considered in this work these include the return estimates $\hat{Q}(\bar{s},a;\theta_Q)$ and optionally additional predictions such as reconstructed sensor observations, next states or instantaneous rewards used for state representation learning, as shown in Figure 1. We use the notation $\theta_{x,y}$ to indicate a subset of the parameters θ consisting of θ_x and θ_y.

4.2 Gradient-based optimization

For the gradient-based optimization of θ we use two simple and popular deep reinforcement-learning algorithms. For policies with discrete actions we use the DQN algorithm (Mnih et al., 2015). For continuous actions we use the DDPG algorithm (Lillicrap et al., 2016). Both algorithms rely on bootstrapping to learn state-action value functions. Experience samples $\{o, a, o', r, \mathbb{T}\}$ are collected by following an exploratory policy $\tilde{\pi}$ (defined below). These samples are stored in an experience buffer, from which they are sampled uniformly at random to calculate training targets $q_t(o,a;\theta^-_{s,Q,a}) = r + (1-\mathbb{T})\gamma\hat{Q}(o',\pi(o';\theta^-_{s,a});\theta^-_{s,Q})$. Here, θ^- are older versions of the network parameters θ. The state-action estimation function $\hat{Q}(o,a;\theta_{s,Q})$ is trained by minimizing the squared error between the predictions $\hat{Q}(o,a;\theta_{s,Q})$ and the training targets $q_t(o,a;\theta^-_{s,Q,a})$ through stochastic gradient descent. For the DQN algorithm, the return predictions for all actions are estimated for a given state representation $\bar{s}$ by a linear layer. In the DDPG algorithm, a neural network is used that takes both o and a as inputs and outputs the predicted expectation of the return $\hat{Q}(o,a;\theta_{s,Q})$.

For the DQN algorithm, we follow Plappert et al. (2018) in having an explicit policy head that is separate from the value function estimation (but uses the same state representation). This head is trained using the negative log likelihood objective to predict the action with the highest Q-value, given the state representation. For the DDPG algorithm, the policy is also separate from the value function. It is trained to output the actions that maximize the Q-value estimates using the deterministic policy gradient (Lillicrap et al., 2016). For further details we refer the reader to (Mnih et al., 2015; Lillicrap et al., 2016; Plappert et al., 2018) and the supplementary material of this paper [1].

Through fitting value functions, a state encoder is trained implicitly. It can sometimes be beneficial to add explicit state representation learning objectives as well, for instance to enable learning when rewards are sparse, and to learn more general features (Jaderberg et al., 2017; de Bruin et al., 2018b).

4.3 Parameter space exploration

During both the gradient-based and the gradient-free phases of the optimization, parameter space exploration is used to explore the state-action space. The exploratory policy is given by $\tilde{\pi} = h(g(o; \theta_s); \theta_{\tilde{a}})$. The parameters $\theta_{\tilde{a}}$, which represent the exploratory version of the action-selection part of the policy, are re-sampled at the start of every Kth episode according to:

$$\theta_{\tilde{a}} \sim \mathcal{N}(\mu, \sigma C), \tag{1}$$

where the choice and evolution of μ, σ and C depend on the optimization phase.

During the gradient-based (reinforcement learning) phase of the algorithm, a new exploratory policy is sampled every episode ($K = 1$). The parameters are sampled from an isotropic mutation distribution ($C = I$) which is centered around the parameters of the current policy ($\mu = \theta_a$). The scaling of the parameter noise σ is adjusted according to the method of Plappert et al. (2018):

$$\sigma_{k+1} = \begin{cases} \alpha \sigma_k & \text{if } d(\pi, \tilde{\pi}) \leq \delta, \\ \frac{1}{\alpha} \sigma_k & \text{otherwise.} \end{cases} \tag{2}$$

To ensure that the scale of the parameters to which this noise is applied is not too different, layer normalization (Ba et al., 2016) is used on the perturbed layers (Plappert et al., 2018). The distance measure d and threshold δ relate the exploration scale σ to action space exploration, allowing for more intuitive hyperparameter choices. For DDPG, we follow Plappert et al. (2018) in using

$$d_{\text{DDPG}}(\pi, \tilde{\pi}) = \sqrt{\frac{1}{N} \sum_{i=1}^{N} \mathbb{E}\left[(\pi(o_i) - \tilde{\pi}(o_i))^2 \right]},$$

where N is the dimensionality of the actions and the expectation is estimated over a batch of samples o from the experience buffer. Using this distance measure, the threshold value can be chosen as $\delta \doteq \sigma_a$ to get exploration with the same standard deviation from the policy in the action space as normally distributed noise with a standard deviation of σ_a. For DQN, we do deviate from the method of Plappert et al. (2018) and simply count the fraction of observations per episode for which $\pi \neq \tilde{\pi}$ and compare this directly to the desired epsilon greedy action space exploration fraction: $\delta = \epsilon$.

[1] `https://github.com/timdebruin/drl-gradient-free-finetuning`

4.4 Gradient-free optimization phase

During the second phase of learning, we first restore all network parameters θ to the values θ^* that resulted in the highest undiscounted return so far [2]. We then use the gradient-free CMA-ES (Hansen and Ostermeier, 2001) optimization procedure to further optimize the action-selection parameters θ_a.

We start by initializing a normal distribution (1) with:

- $\mu_0 = \theta^*_{\tilde{a}}$ (the parameters that led to the best performance during training),
- σ_0: we use the procedure of (2) to adapt σ based on a desired exploration intensity in the action space.
- $C_0 = I$ (the mutation distribution starts out isotropic, but is adapted over time in contrast to the exploration during the gradient-based phase).

After this initialization, the CMA-ES algorithm then adapts μ, σ and C by iteratively sampling λ parameter vectors $\tilde{\theta}_a$ from the current distribution $\mathcal{N}(\mu_k, \sigma_k C_k)$ and updating the distribution based on their fitness. For the evaluation of the sampled parameter values $\theta_{\tilde{a}}$, we perform K roll-outs of the exploration policy $\tilde{\pi}(\cdot; \theta_{s,\tilde{a}})$ and average the returns. The values of μ, σ and C are updated so that the parameters corresponding to the higher fitness scores are more likely under the updated distribution. In this update, previous updates are also taken into account to speed up the learning. For a more detailed description of the CMA-ES procedure we refer to (Hansen and Ostermeier, 2001; Hansen et al., 2019). One important aspect of the CMA-ES algorithm is that the intensity (σ) and shape C of the exploration are adapted automatically. During the final phase of learning, this can allow the optimization procedure to reduce the exploration intensity in a controlled manner (Stulp and Sigaud, 2012).

5. EXPERIMENTS

We write `DRL(time)`$\rightarrow$CMA-ES(`exploration`) in the following to indicate that we use the `DRL` deep reinforcement learning algorithm (either DQN or DDPG) for `time` episodes or environment steps before switching to CMA-ES where we initialize σ (1) to `exploration`. We use the CMA-ES implementation of Hansen et al. (2019). In all experiments, we use their default population size of $\lambda = 4 + \text{floor}(3 \ln(n))$, where n is the number of elements in θ_a. Our code is available online [1].

We start with experiments on the OpenAI Gym CarRacing-v0 benchmark (Brockman et al., 2016). In this task, the observation o is a top-down image of a car on a randomly generated racing track on which it needs to complete a lap as quickly as possible. Only a single image is provided at each time-step, where the car's translational and angular velocities, wheel speeds and steering wheel position are encoded as bars in the image. The task of the state encoder $g(o; \theta_s)$ is therefore to decode this information, along with all other relevant information, and include it in the state representation $\bar{s}$. We discretize the action space (of throttle, breaking and steering inputs) into 7 actions. The aim of

[2] Note that each episode is run on a randomly generated track. The highest reward so far was therefore not necessarily obtained by the best policy so far, but likely obtained by one that is close tot that.

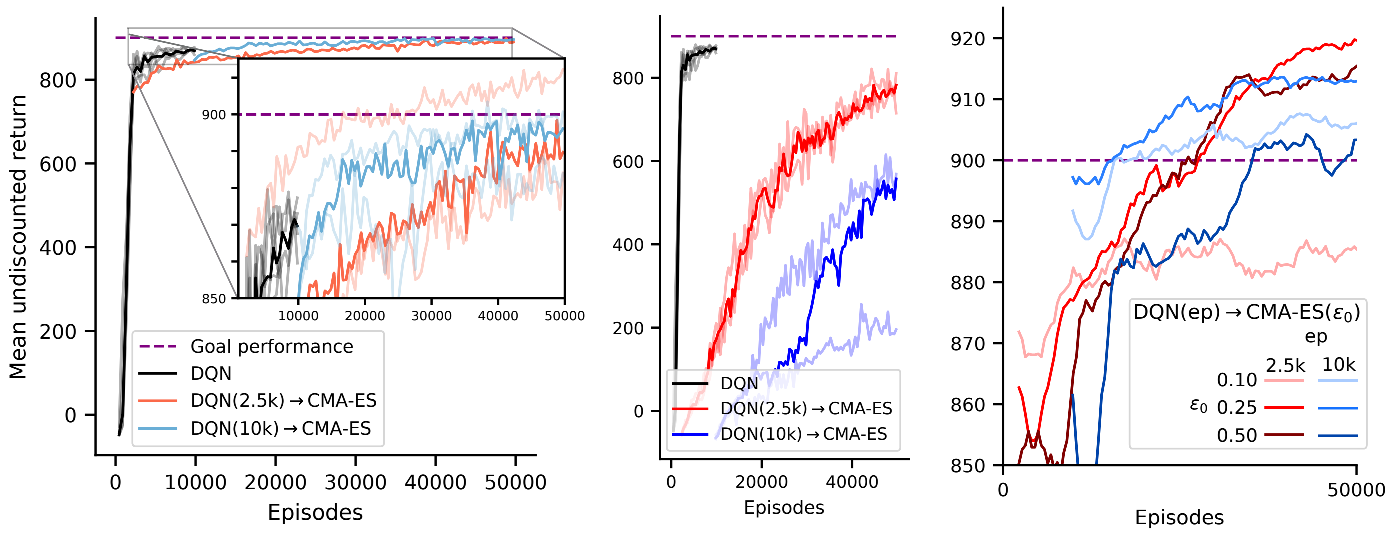


Fig. 2. Mean undiscounted return over $K = 16$ episodes on the CarRacing-v0 benchmark. Median results over 3 experiment repetitions are boldened in the left and middle plots. Population size $\lambda = 28$. *Left)* Mean undiscounted return of the population when fine-tuning $\theta^*_{\tilde{a}}$. Here, σ_0 is chosen such that $\epsilon_0 = 0.25$. *Middle)* Training θ_a from scratch ($\sigma_0 = 1$), *Right)* highest mean undiscounted return in the population per iteration when fine-tuning $\theta^*_{\tilde{a}}$ depending on switch episode (ep) and initial CMA-ES exploration magnitude ϵ_0. Each line represents a single run.

the action selection subnetwork $h(\bar{s}; \theta_a)$ is to select these actions in a very reliable way; the car should drive along the track quickly without mistakes. This is because solving the task is defined as getting a mean score of at least 900 over 100 subsequent episodes. While obtaining an undiscounted sum of rewards over an episode of a little more than 900 is not too hard, leaving the track at any point will quickly result in much lower scores.

The need for a very precise and reliable policy based on a learned state representation makes this benchmark interesting for our proposed method. It also allows for a comparison with the related work of Ha and Schmidhuber (2018) who chose this benchmark for similar reasons. On this benchmark we use the DQN architecture (Mnih et al., 2015) with an added policy head (Plappert et al., 2018). We consider all but the final layer to be the state encoder g, making the state representation 512 dimensional. This leaves a final layer with $\theta_a \in \mathbb{R}^{3591}$ for the action selection h. For the CMA-ES phase we sample $\lambda = 28$ values of the parameter vector $\theta_{\tilde{a}}$ and evaluate each over $K = 16$ episodes. Each iteration of the CMA-ES algorithm therefore consists of 448 episodes.

We use this benchmark to test the assumptions on which our method is based: that the state-encoder part of a good policy can quickly be learned through deep reinforcement learning and that stable convergence to a better performing policy can be achieved through gradient-free fine-tuning of the final action-selection parameters. The left panel of Figure 2 shows both parts in action. The main graph shows the relative speed with which the gradient-based DRL phase can learn the values of the 1.7 million parameters in θ_s and initialize the 3591 parameters in θ_a to a point where the task is performed reasonably well. Note that getting the DQN algorithm to perform this well required careful tuning. The inset shows the stability with which the parameters of θ_a can subsequently be tuned further using the gradient-free CMA-ES procedure, learning not just to solve the task but also beating what is to the best of our knowledge the highest reported score in the literature (Ha and Schmidhuber, 2018) while using considerably fewer episodes, as shown in Table 1. This gradient free optimization required significantly less tuning, but would not work without the initialization from the gradient-based phase.

Given that we have two distinct optimization phases, one important question is when we should switch from the gradient-based phase to the gradient-free phase. Unfortunately, there is no simple answer to this question. As the gradient-based training progresses, the performance increases rapidly. For the learning speed, it is therefore beneficial not to switch too early. At the same time, the results in the middle and right panels of Figure 2 seem to indicate that the gradient-based optimization will eventually seek out areas in the parameter space that are harder to optimize for the gradient-free optimizer. This might partially be a result of using ADAM for the gradient-based optimization (Wilson et al., 2017)—a very common choice in DRL.

A closely related question is how much (initial) exploration around the action-selection parameters $\theta^*_{\tilde{a}}$ is beneficial. The right panel of Figure 2 shows the mean undiscounted return for the best parameter vector $\theta_{\tilde{a}}$ sampled per iteration of the CMA-ES procedure, as a function of the number of DRL episodes and the initial exploration ϵ_0 that σ_0 is adapted to using (2). It can be seen that more gradient based optimization steps might limit the potential for improvement. To see whether this effect is related to over-fitting in the state encoder or the action-selection parameters, we perform two experiments.

In the first one we train θ_a from scratch using CMA-ES, rather than starting from $\theta^*_{\tilde{a}}$. This is shown in the middle panel of Figure 2. We see again that when using a state encoder that is trained for fewer episodes, the subsequent gradient free optimization of the action selection subnetwork is easier. As an aside, comparing the left and

Table 1. Performance over 100 test episodes on the CarRacing-v0 benchmark. Tested policies were those that set the highest (mean) undiscounted return during training for the first (of 3) training experiment repetitions.

Method	Episodes	Score
DQN(2500ep) [ours]	2,500	871 ± 86
DQN Gerber et al. (2018)	3,000	892 ± 41
SRL → CMA-ES Ha and Schmidhuber (2018)	1,843,200	906 ± 21
DQN(2500ep) → CMA-ES($\epsilon_0 = 0.10$) [ours]	50,000	890 ± 34
DQN(2500ep) → CMA-ES($\epsilon_0 = 0.25$) [ours]	50,000	**918 ± 20**
DQN(2500ep) → CMA-ES($\epsilon_0 = 0.50$) [ours]	50,000	915 ± 28
Genetic Algorithm Risi and Stanley (2019)	240,000	903 ± 72

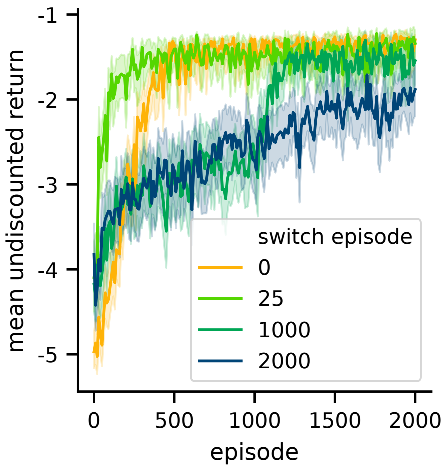


Fig. 3. Magman benchmark results. Mean undiscounted return per episode for DDPG(switch episode)→CMA-ES($\sigma_{a0} = 0.25$) Results are from 50 trials, with the 95% bootstrapped confidence bounds of the means shown.

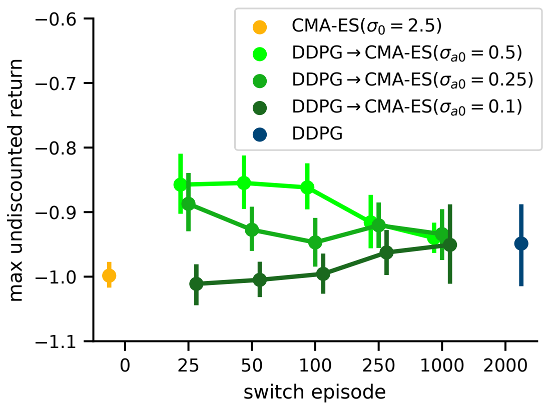


Fig. 4. Magman benchmark results. Maximum undiscounted return per learning trial. Results are from 50 trials, with the 95% bootstrapped confidence bounds of the means shown.

middle panels of Figure 2 also clearly shows the sample efficiency benefit of starting from the pre-trained $\theta^*_{\tilde{a}}$.

Since at least part of the performance loss when switching later seems to be related to over-fitting in the state encoder, we perform an experiment where state representation losses are added to the state encoder learning objective to help regularize the state representation (de Bruin et al., 2018b). While the details of this experiment as well as the resulting learning curves are left out for brevity, we indeed observed that switching later now lead to better performance. While this seems to confirm that the state-encoder was over fitting and that the SRL objectives can help prevent this over fitting, the eventual performance of these experiments was worse than those of the experiments without SRL objectives.

Due to the computational complexity of these methods, these results were from three experiment repetitions. For more statistically significant results we perform experiments on the Magman benchmark (de Bruin et al., 2018a). This benchmark is low-dimensional, but requires a very precise control policy with continuous actions. We use two smaller networks with 2 hidden layers of 64 units each for a policy and a Q-function. During the gradient based optimization we use the DDPG algorithm to train these networks. During the CMA-ES phase, we optimize all parameters of the policy. The results, shown in Figures 3 and 4, again demonstrate the benefits of this two stage optimization. Switching early, with sufficient initial exploration, results in both faster learning as well as a higher maximum performance than either DDPG or CMA-ES alone.

Finally, we tested the method on two Atari games. We followed the architecture of (Plappert et al., 2018) where the policy head is implemented as a single fully connected layer, right after the convolutional layers of the DQN architecture. The $\hat{Q}$ head still had the usual 512 dimensional fully connected intermediate layer Mnih et al. (2015). While we found that this architecture worked better during the DRL phase, it meant that the final action selection layer now contained more parameters than can feasibly be optimized using CMA-ES. Therefore, we trained a new policy head—which did have the 512 dimensional intermediate layer—after the DRL phase. This head was trained to minimize the KL divergence between its predictions and the predictions of the original policy head on samples from the replay buffer (Parisotto et al., 2015). We then used CMA-ES to optimize the final layer of this new policy head. When testing the controllers resulting in the highest undiscounted returns during both phases (evaluated with $K = 1$), we again observe that the CMA-ES procedure was able to noticeably improve the policy performance by fine-tuning the final action-selection parameters. Results are shown in Table 2. For the Enduro benchmark, where episodes can be very long and consequences of actions less immediate, the gradient-free optimization also resulted in the highest outright scores. On the freeway benchmark, with short episodes and more immediate consequences to actions, DRL was able to find a near optimal policy. The gradient-free optimization phase

Table 2. Performance over 100 test episodes on two Atari benchmarks.

	Enduro		Freeway	
Method	Steps	Mean ± SD	Steps	Mean ± SD
DQN(50m)	50m	1188 ± 240	50m	30.7 ± 0.9
DQN(50m) $\rightarrow$ CMA-ES($\epsilon_0 = 0.5$)	250m	**1483** ± **505**	100m	**32.6** ± **1.0**

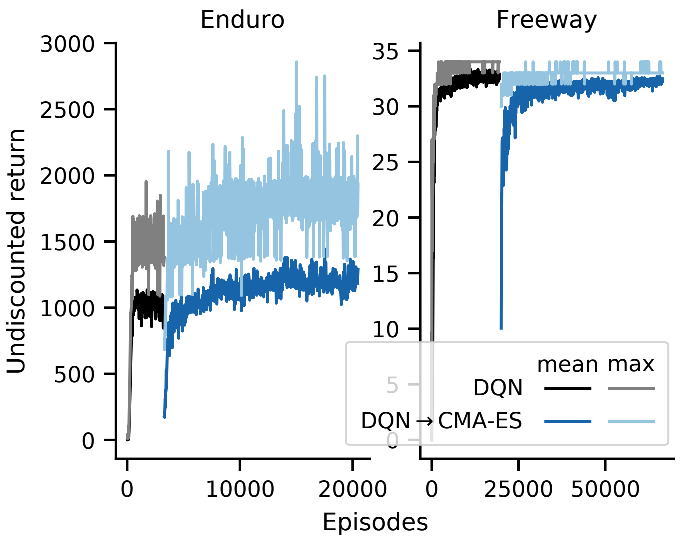


Fig. 5. Atari results, mean return (of population) indicated in dark and max return (best of population) per iteration in lighter colors.

here found a policy that has a higher mean performance over 100 episodes, but did not obtain the maximum score.

6. CONCLUSION AND FUTURE WORK

In this work we combined gradient-based deep reinforcement learning methods with a gradient-free evolutionary strategy. We showed how a relatively short initial gradient-based phase was able to learn a good state representation and a decent action selection strategy relatively quickly, while a subsequent gradient-free fine-tuning of the action-selection parameters resulted in stable convergence to a policy performance not achieved with gradient-based optimization alone. Experiments on a small scale benchmark, where no state encoder needed to be trained, also showed how the combination of gradient-based and gradient-free optimization was able to learn more quickly, and find better performing policies, than either method alone.

The results suggests several avenues for future work. When we consider part of the network to represent a state encoder, we freeze this encoder during the gradient-free fine-tuning of the policy. However, if the improved policy that is found through gradient-free optimization visits significantly different states, it might be worth updating the state encoder using the data obtained with this new policy. On the other hand, if we do keep the state encoder frozen, we can consider using model compression (Moniz et al., 2019; Hinton et al., 2015) on this part of the network to speed up the evaluation of the gradient-free phase further.

When optimizing all parameters of a neural network, it might be interesting to investigate a tighter integration of deep reinforcement learning with parameter-space exploration and CMA-ES. For instance by introducing the ability of CMA-ES to scale and shape the exploration to the gradient-based optimization phase, or by allowing the DRL gradients to bias the update direction of the population mean of CMA-ES.

REFERENCES

Andrychowicz, M., Baker, B., Chociej, M., Jozefowicz, R., McGrew, B., Pachocki, J., Petron, A., Plappert, M., Powell, G., Ray, A., et al. (2018). Learning dexterous in-hand manipulation. *arXiv preprint arXiv:1808.00177*.

Ba, J.L., Kiros, J.R., and Hinton, G.E. (2016). Layer normalization. ArXiv preprint arXiv:1607.06450.

Bansal, T., Pachocki, J., Sidor, S., Sutskever, I., and Mordatch, I. (2017). Emergent complexity via multi-agent competition. *arXiv preprint arXiv:1710.03748*.

Bengio, Y., Courville, A., and Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8), 1798–1828.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym. *CoRR*, abs/1606.01540. URL `http://arxiv.org/abs/1606.01540`.

de Bruin, T., Kober, J., Tuyls, K., and Babuška, R. (2018a). Experience selection in deep reinforcement learning for control. *Journal of Machine Learning Research*, 19(9), 1–56. URL `http://jmlr.org/papers/v19/17-131.html`.

de Bruin, T., Kober, J., Tuyls, K., and Babuška, R. (2018b). Integrating state representation learning into deep reinforcement learning. *IEEE Robotics and Automation Letters*, 3(3), 1394–1401.

Finn, C., Tan, X.Y., Duan, Y., Darrell, T., Levine, S., and Abbeel, P. (2016). Deep spatial autoencoders for visuomotor learning. In *Int. Conf. Robotics and Automation (ICRA)*.

Gerber, P., Guan, J., , Nunez, E., Phamdo, T., Monsoor, N., and Malaya, N. (2018). Solving openai's car racing environment with deep reinforcement learningand dropout. URL `https://github.com/AMD-RIPS/RL-2018/blob/master/documents/nips/nips_2018.pdf`.

Gu, S., Lillicrap, T., Turner, R.E., Ghahramani, Z., Schölkopf, B., and Levine, S. (2017). Interpolated policy gradient: Merging on-policy and off-policy gradient estimation for deep reinforcement learning. In *Advances in Neural Information Processing Systems*, 3849–3858.

Ha, D. and Schmidhuber, J. (2018). World models. *arXiv preprint arXiv:1803.10122*.

Hansen, N., Akimoto, Y., and Baudis, P. (2019). CMA-ES/pycma on Github. Zenodo, DOI:10.5281/zenodo.2559634. doi:10.5281/zenodo.2559634. URL `https://doi.org/10.5281/zenodo.2559634`.

Hansen, N. and Ostermeier, A. (2001). Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, 9(2), 159–195.

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

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., and Meger, D. (2017). Deep reinforcement learning that matters. *arXiv preprint arXiv:1709.06560*.

Hinton, G., Vinyals, O., and Dean, J. (2015). Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

Jaderberg, M., Mnih, V., Czarnecki, W.M., Schaul, T., Leibo, J.Z., Silver, D., and Kavukcuoglu, K. (2017). Reinforcement learning with unsupervised auxiliary tasks. In *Int. Conf. Learning Representations (ICLR)*.

Jonschkowski, R. and Brock, O. (2015). Learning state representations with robotic priors. *Autonomous Robots*, 39(3), 407–428.

Koutník, J., Cuccu, G., Schmidhuber, J., and Gomez, F. (2013). Evolving large-scale neural networks for vision-based reinforcement learning. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, 1061–1068. ACM.

Lange, S., Riedmiller, M., and Voigtlander, A. (2012). Autonomous reinforcement learning on raw visual input data in a real world application. In *Int. Joint Conf. Neural Networks (IJCNN)*.

Levine, N., Zahavy, T., Mankowitz, D.J., Tamar, A., and Mannor, S. (2017). Shallow updates for deep reinforcement learning. In *Advances in Neural Information Processing Systems*, 3135–3145.

Levine, S., Finn, C., Darrell, T., and Abbeel, P. (2016). End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(39), 1–40.

Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2016). Continuous control with deep reinforcement learning. In *International Conference on Learning Representations (ICLR)*.

Lin, H.W., Tegmark, M., and Rolnick, D. (2017). Why does deep and cheap learning work so well? *Journal of Statistical Physics*, 168(6), 1223–1247.

Marbach, P. and Tsitsiklis, J.N. (2003). Approximate gradient methods in policy-space optimization of markov reward processes. *Discrete Event Dynamic Systems*, 13(1-2), 111–148.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540), 529–533.

Moniz, J.R.A., Patra, B., and Garg, S. (2019). Compression and localization in reinforcement learning for atari games. *arXiv preprint arXiv:1904.09489*.

Parisotto, E., Ba, J.L., and Salakhutdinov, R. (2015). Actor-mimic: Deep multitask and transfer reinforcement learning. *arXiv preprint arXiv:1511.06342*.

Pérez Dattari, R., Celemin, C., Ruiz Del Solar, J., and Kober, J. (2019). Continuous control for high-dimensional state spaces: An interactive learning approach. In *IEEE International Conference on Robotics and Automation (ICRA)*.

Plappert, M., Houthooft, R., Dhariwal, P., Sidor, S., Chen, R.Y., Chen, X., Asfour, T., Abbeel, P., and Andrychowicz, M. (2018). Parameter space noise for exploration. In *International Conference on Learning Representations (ICLR)*.

Risi, S. and Stanley, K.O. (2019). Deep neuroevolution of recurrent and discrete world models. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 456–462.

Salimans, T., Ho, J., Chen, X., Sidor, S., and Sutskever, I. (2017). Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. (2015a). Trust region policy optimization. In *International Conference on Machine Learning*, 1889–1897.

Schulman, J., Moritz, P., Levine, S., Jordan, M., and Abbeel, P. (2015b). High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al. (2018). A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419), 1140–1144.

Stulp, F. and Sigaud, O. (2012). Path integral policy improvement with covariance matrix adaptation. In *Proceedings of the 29th International Conference on Machine Learning (ICML)*.

Sutton, R.S. and Barto, A.G. (2018). *Reinforcement learning: An introduction*. MIT press.

Wang, Z., Bapst, V., Heess, N., Mnih, V., Munos, R., Kavukcuoglu, K., and de Freitas, N. (2017). Sample efficient actor-critic with experience replay. In *International Conference on Learning Representations (ICLR)*.

Wilson, A.C., Roelofs, R., Stern, M., Srebro, N., and Recht, B. (2017). The marginal value of adaptive gradient methods in machine learning. In *Advances in Neural Information Processing Systems*, 4148–4158.