Uncertainty Handling in Evolutionary Direct Policy Search

Verena Heidrich-Meisner
Institut für Neuroinformatik
Ruhr-Universität Bochum
44780 Bochum, Germany
verena.heidrich-meisner
@neuroinformatik.rub.de

Christian Igel
Institut für Neuroinformatik
Ruhr-Universität Bochum
44780 Bochum, Germany
christian.igel
@neuroinformatik.rub.de

Abstract

Uncertainty arises in reinforcement learning from various sources. Therefore it is necessary to consider statistics based on several roll-outs for evaluating behavioral policies. An adaptive uncertainty handling is added to the CMA-ES, a variable metric evolution strategy proposed for direct policy search. The uncertainty handling dynamically adjusts the number of episodes considered in each evaluation of a policy. It controls the signal to noise ratio such that it is just high enough for a sufficiently good ranking of candidate policies, which is in turn sufficient for the CMA-ES to find better solutions. This significantly increases the learning speed without impairing the quality of the final solutions. The CMA-ES resembles natural policy gradient methods, which serve as a baseline for comparison.

1 Introduction

Dealing with uncertainty is one of the major issues in reinforcement learning (RL). When solving (partially observable) Markov decision processes solely based on observations and interactions with the environment, uncertainty and randomness arise from several sources. The initial state usually varies, state-transitions and reward signals can be stochastic, and the state observations may be noisy.

We consider RL methods that search in a parametrized policy space. The search direction is determined using estimates of the performance of behavioral policies or estimates of performance gradients. Uncertainty and randomness require that these estimates are based on a sample of several episodes (roll-outs). The sample size is a crucial parameter. If too few episodes are considered, the estimates are not reliable enough to allow for learning. If too many episodes are considered, the learning process gets too slow. Unfortunately, it is usually not possible to determine an appropriate sample size for a given problem a priori (in practice we just make it "large enough") and the optimal number may vary in the course of learning.

We promote the covariance matrix evolution strategy (CMA-ES, [6]) for direct policy search, which gives striking results on RL benchmark problems [11, 5, 9, 10, 8]. The CMA-ES adapts the policy as well as parameters of its own search strategy (such as a variable metric) based on ranking policies. This is already much less susceptible to noise than estimating absolute performances or performance gradients [10]. Still, the ranking must be sufficiently accurate to evolve better policies, and this depends on the degree of uncertainty as well as the number of roll-outs considered per performance estimation of each candidate solution. We propose to augment the CMA-ES for RL with a new adaptive uncertainty handling scheme [7], which dynamically adapts the number of episodes for evaluating a policy such that the ranking of new candidate solutions is just reliable enough to drive the learning process. The uncertainty handling scheme is independent of the CMA-ES and could be combined with other RL approaches and strategies for distributing evaluations among candidate solutions (e.g., for evolutionary online RL [19]).

Algorithm 1: rank-μ CMA-ES

initialize $\boldsymbol{m}^{(1)} = \boldsymbol{\theta}_{\text{init}}$, $\sigma^{(1)}$, evolution paths $\boldsymbol{p}_\sigma^{(1)} = \boldsymbol{p}_c^{(1)} = \mathbf{0}$ and covariance matrix $\boldsymbol{C}^{(1)} = \boldsymbol{I}$ (unity matrix), $n_{\text{eval}}^{(1)} = 1$

// k counts number of generations / policy updates:

for $k = 1, \ldots$ **do**

  for $l = 1, \ldots, \lambda$ **do** $x_l^{(k+1)} \sim \mathrm{N}(\boldsymbol{m}^{(k)}, {\sigma^{(k)}}^2 \boldsymbol{C}^{(k)})$ // create new offspring

  for $l = 1, \ldots, \lambda$ **do** $f_l \leftarrow \texttt{performance}(\boldsymbol{x}_l^{(k+1)}, n_{\text{eval}}^{(k)})$ // evaluate offspring

  $n_{\text{eval}}^{(k+1)} \leftarrow \texttt{uncertaintyHandling}\,(\{\boldsymbol{x}_l^{(k+1)} \,|\, l \in \{1, \ldots, \lambda\}\})$ // see section 3

  $\boldsymbol{m}^{(k+1)} \leftarrow \sum_{i=1}^{\mu} w_i \boldsymbol{x}_{i:\lambda}^{(k)}$ // selection and recombination

  // step size control:

  $\boldsymbol{p}_\sigma^{(k+1)} \leftarrow (1 - c_\sigma)\boldsymbol{p}_\sigma^{(k)} + \sqrt{c_\sigma(2 - c_\sigma)\mu_{\text{eff}}}\boldsymbol{C}^{(k)^{-\frac{1}{2}}} \frac{\boldsymbol{m}^{(k+1)} - \boldsymbol{m}^{(k)}}{\sigma^{(k)}}$

  $\sigma^{(k+1)} \leftarrow \sigma^{(k)} \exp \frac{c_\sigma}{d_\sigma} \left(\frac{\|\boldsymbol{p}_\sigma^{(k+1)}\|}{E\|\mathrm{N}(\mathbf{0}, \boldsymbol{I})\|} - 1 \right)$

  // covariance matrix update:

  $\boldsymbol{p}_c^{(k+1)} \leftarrow (1 - c_c)\boldsymbol{p}_c^{(k)} + \sqrt{c_c(2 - c_c)\mu_{\text{eff}}} \frac{\boldsymbol{m}^{(k+1)} - \boldsymbol{m}^{(k)}}{\sigma^{(k)}}$

  $\boldsymbol{C}^{(k+1)} \leftarrow (1 - c_{\text{cov}})\boldsymbol{C}^{(k)} + \frac{c_{\text{cov}}}{\mu_{\text{cov}}} \boldsymbol{p}_c^{(k+1)} \boldsymbol{p}_{\boldsymbol{c}}^{(\boldsymbol{k+1})^T} + c_{\text{cov}} \left(1 - \frac{1}{\mu_{\text{cov}}}\right) \sum_{i=1}^{\mu} w_i \boldsymbol{z}_{i:\lambda}^{(k)} \boldsymbol{z}_{i:\lambda}^{(k)^T}$

In the next section, we introduce the CMA-ES for direct policy search . In section 3, we describe the uncertainty handling. The empirical evaluation follows in section 4 before the conclusions.

2 Variable Metric Direct Policy Search

Evolution strategies are random search methods, which iteratively sample a set of candidate solutions from a probability distribution over the search space (i.e., the space of policies), evaluate these potential solutions, and construct a new probability distribution over the search space based on the gathered information [2]. In ESs, this search distribution is parametrized by a set of candidate solutions, the *parent population* with size μ, and by parameters of the variation operators that are used to create new candidate solutions (the *offspring population* with size λ) from the parent population.

In each iteration k of the CMA-ES, which is shown in Algorithm 1, the lth offspring $\boldsymbol{x}_l \in \mathbb{R}^n$, $l \in \{1, \ldots, \lambda\}$, is generated by multi-variate *Gaussian mutation* and *weighted global intermediate recombination*, i.e.,

$$\boldsymbol{x}_l^{(k+1)} \leftarrow \boldsymbol{m}^{(k)} + \sigma^{(k)} \boldsymbol{z}_l^{(k)} \quad , \text{ where } \boldsymbol{z}_l^{(k)} \sim \mathrm{N}(\mathbf{0}, \boldsymbol{C}^{(k)}) \text{ and } \boldsymbol{m}^{(k)} \leftarrow \sum_{l=1}^{\mu} w_l \boldsymbol{x}_{l:\lambda}^{(k)}$$

with $\boldsymbol{x}_{l:\lambda}^{(k)}$ denoting the lth best individual among $\boldsymbol{x}_1^{(k)}, \ldots, \boldsymbol{x}_\lambda^{(k)}$. This corresponds to rank-based selection, in which the best μ of the λ offspring form the next parent population. A common choice for the recombination weights is $w_l \propto \ln(\mu + 1) - \ln(l)$, $\|\boldsymbol{w}\|_1 = 1$.

The CMA-ES is a variable metric algorithm adapting both the n-dimensional covariance matrix $\boldsymbol{C}^{(k)}$ of the normal mutation distribution as well as the *global step size* $\sigma^{(k)} \in \mathbb{R}^+$. The covariance matrix update has two parts, the rank-1 update considering the change of the population mean over time and the rank-μ update considering the successful variations in the last generation. The rank-1 update is based on a low-pass filtered *evolution path* $\boldsymbol{p}^{(k)}$ of successful (i.e., selected) steps

$$\boldsymbol{p}_c^{(k+1)} \leftarrow (1 - c_c)\, \boldsymbol{p}_c^{(k)} + \sqrt{(c_c(2 - c_c)\mu_{\text{eff}})}\, \frac{\boldsymbol{m}^{(k+1)} - \boldsymbol{m}^{(k)}}{\sigma^{(k)}}$$

and aims at changing $\boldsymbol{C}^{(k)}$ to make steps in the promising direction $\boldsymbol{p}^{(k+1)}$ more likely by morphing the covariance towards $\left[\boldsymbol{p}_c^{(k+1)}\right] \left[\boldsymbol{p}_c^{(k+1)}\right]^T$. The backward time horizon of the cumulation process is approximately c_c^{-1}, where $c_c = 4/(n + 4)$ is roughly inversely linear in the dimension of the path

vector. The *variance effective selection mass* $\mu_{\text{eff}} = \left(\sum_{l=1}^{\mu} w_l^2\right)^{-1}$ is a normalization constant. The rank-μ update aims at making the single steps that were selected in the last iteration more likely by morphing $\boldsymbol{C}^{(k)}$ towards $\left[\boldsymbol{z}_{i:\lambda}^{(k)}\right]\left[\boldsymbol{z}_{i:\lambda}^{(k)}\right]^T$. Putting both updates together, we have

$$\boldsymbol{C}^{(k+1)} \leftarrow (1 - c_{\text{cov}})\boldsymbol{C}^{(k)} + \frac{c_{\text{cov}}}{\mu_{\text{cov}}}\boldsymbol{p}_c^{(k+1)}{\boldsymbol{p}_c^{(k+1)}}^T + c_{\text{cov}}\left(1 - \frac{1}{\mu_{\text{cov}}}\right)\sum_{i=1}^{\mu} w_i \boldsymbol{z}_{i:\lambda}^{(k)}{\boldsymbol{z}_{i:\lambda}^{(k)}}^T \quad .$$

The constants c_{cov} and μ_{cov} are fixed learning rates. The learning rate of the covariance matrix update $c_{\text{cov}} = \frac{2}{(n+\sqrt{2})^2}$ is roughly inversely proportional to the degrees of freedom of the covariance matrix. The parameter μ_{cov} mediates between the rank-μ update ($\mu_{\text{cov}} \to \infty$) and the rank-one update ($\mu_{\text{cov}} = 1$). The default value is $\mu_{\text{cov}} = \mu_{\text{eff}}$.

The global step size $\sigma^{(k)}$ is adapted on a faster timescale. It is increased if the selected steps are larger and/or more correlated than expected and decreased if they are smaller and/or more anticorrelated than expected:

$$\sigma^{(k+1)} \leftarrow \sigma^{(k)} \exp\left(\frac{c_\sigma}{d_\sigma}\left(\frac{\|\boldsymbol{p}_\sigma^{(k+1)}\|}{\mathrm{E}\{\|\mathrm{N}(\boldsymbol{0}, \boldsymbol{I})\|\}} - 1\right)\right) \ ,$$

where $\mathrm{E}\{\|\mathrm{N}(\boldsymbol{0}, \boldsymbol{I})\|\}$ is the expectation of the χ_n distribution, and the (*conjugate*) evolutions path is

$$\boldsymbol{p}_s^{(k+1)} \leftarrow (1 - c_\sigma)\,\boldsymbol{p}_s^{(k)} + \sqrt{c_\sigma(2 - c_\sigma)\mu_{\text{eff}}}\,{\boldsymbol{C}^{(k)}}^{-\frac{1}{2}}\frac{\boldsymbol{m}^{(k+1)} - \boldsymbol{m}^{(k)}}{\sigma^{(k)}} \quad .$$

Again, $c_\sigma = \frac{\mu_{\text{eff}}+2}{n+\mu_{\text{eff}}+3}$ is a fixed learning rate and $d_\sigma = 1 + 2\max\left(0, \sqrt{\frac{\mu_{\text{eff}}-1}{n+1}}\right) + c_\sigma$ is a damping factor. The matrix $\boldsymbol{C}^{-\frac{1}{2}}$ is defined as $\boldsymbol{B}\boldsymbol{D}^{-1}\boldsymbol{B}^T$, where $\boldsymbol{B}\boldsymbol{D}^2\boldsymbol{B}^T$ is an eigendecomposition of $\boldsymbol{C}$ ($\boldsymbol{B}$ is an orthogonal matrix with the eigenvectors of $\boldsymbol{C}$ and $\boldsymbol{D}$ a diagonal matrix with the corresponding eigenvalues) and sampling $\mathrm{N}(\boldsymbol{0}, \boldsymbol{C})$ is done by sampling $\boldsymbol{B}\boldsymbol{D}\mathrm{N}(\boldsymbol{0}, \boldsymbol{I})$.

The function `performance(`$\boldsymbol{x}$`,` $n_{\text{eval}}^{(k)}$`)` in Algorithm 1 corresponds to the evaluation of the policy with parameters $\boldsymbol{\theta} = \boldsymbol{x}$. The parameter $n_{\text{eval}}^{(k)}$ determines over how many episodes a performance (here the return, i.e., accumulated reward) average is computed. The uncertainty handling adapting the parameter $n_{\text{eval}}^{(k)}$ will be described in section 3.

The values of the learning rates and the damping factor are well considered and have been validated by experiments on many basic test functions [6]. *They need not be adjusted dependent on the problem and are therefore* no *hyperparameters of the algorithm.* Also the population sizes can be set to default values, which are $\lambda = \max(4 + \lfloor 3\ln n \rfloor, 5)$ and $\mu = \lfloor \frac{\lambda}{2} \rfloor$ for offspring and parent population, respectively [6]. If we fix $\boldsymbol{C}^{(0)} = \boldsymbol{I}$ and ignore the sample size $n_{\text{eval}}^{(k)}$, the only hyperparameter to be chosen problem dependent is the initial global step size $\sigma^{(0)}$.

The highly efficient use of information and the fast adaptation of σ and $\boldsymbol{C}$ make the CMA-ES one of the best direct search algorithms for real-valued optimization [2]. For a detailed description of the CMA-ES we refer to the articles by Hansen et al. [6]. The CMA-ES was proposed for RL in [11]. In a more recent study, the CMA-ES (without rank-μ update) was compared to 8–12 (depending on the task) other RL algorithms including value-function and policy gradient approaches [4]. On the four test problems where the CMA-ES was considered, it ranked first, second (twice), and third. Further examples of successful applications of the CMA-ES for RL can be found in [13, 17] and additional comparisons on RL benchmarks in [9].

It is interesting to compare CMA-ES for RL and policy gradient methods (PGMs). Both search directly in policy space, but ES for RL are actor-only methods while PGMs often have actor-critic architectures. In contrast to the CMA-ES, PGMs require a differentiable structure on the search space and stochastic policies for learning. Exploration of the search space is realized by random perturbations in both ESs and PGMs. Evolutionary methods usually perturb a deterministic policy by mutation and recombination, while in PGMs the random variations are an inherent property of the stochastic policies. In ESs the search is driven solely by ranking policies and not by the absolute values of performance estimates or even their gradients. The reduced number of random events and

Procedure `uncertaintyHandling(`$\{\boldsymbol{x}_l \,|\, l \in \{1,\ldots,\lambda\}\}$`)`

// reevaluate λ_{reev} solutions (w.l.o.g. the first λ_{reev}):

for $l = 1,\ldots,\lambda_{\text{reev}}$ **do** $f_l^{\text{reeval}} \leftarrow$ `performance`$(\boldsymbol{x}_l, n_{\text{eval}})$

for $l = \lambda_{\text{reev}}+1,\ldots,\lambda$ **do** $f_l^{\text{reeval}} \leftarrow f_l$

// construct joint performance sample:

$\mathcal{L} \leftarrow ((\boldsymbol{x}_1, f_1, f_1), \ldots, (\boldsymbol{x}_\lambda, f_\lambda, f_\lambda), (\boldsymbol{x}_1, f_1, f_1^{\text{reeval}}), \ldots, (\boldsymbol{x}_\lambda, f_\lambda, f_\lambda^{\text{reeval}}))$

for $l = 1,\ldots,\lambda_{\text{reev}}$ **do**

  // compute rank change:

  $\Delta_l \leftarrow |\operatorname{rank}_{\mathcal{L}}^{\text{reeval}}(l) - \operatorname{rank}_{\mathcal{L}}(l)| - 1$

  // determine uncertainty level:

  $s \leftarrow \frac{1}{\lambda_{\text{reev}}} \sum_{l=1}^{\lambda_{\text{reev}}} \left(2\Delta_l - \Delta_\theta^{\lim}(\operatorname{rank}_{\mathcal{L}}^{\text{reeval}}(l) - \mathbb{I}\{f_l^{\text{reeval}} > f_l\}) - \Delta_\theta^{\lim}(\operatorname{rank}_{\mathcal{L}}(l) - \mathbb{I}\{f_l > f_l^{\text{reeval}}\})\right)$

// adjust number of evaluations:

if $s > 0$ **then** $n_{\text{eval}} \leftarrow \alpha n_{\text{eval}}$ **else** $n_{\text{eval}} \leftarrow \frac{1}{\alpha} n_{\text{eval}}$

// update fitness values:

for $l = 1,\ldots,\lambda_{\text{reev}}$ **do** $f_l \leftarrow \frac{f_l + f_l^{\text{reeval}}}{2}$

return n_{eval}

the rank-based evaluation are decisive differences and we hypothesise that they allow ESs to be more robust.

By memorizing successful steps in the evolution path, the CMA-ES infers a search direction from scalar reward signals. The adaptation of the covariance matrix in the CMA-ES is similar to learning the (Fisher) metric in natural PGMs [12, 16, 14]. Arguably one of the most elegant state-of-the-art natural PGMs is the episodic natural actor-critic algorithm (NAC, [16, 14]), which serves as a baseline for comparison in our experiments.

3 Uncertainty Handling for Ranking Policies

Evolutionary algorithms are well suited for optimization in noisy environments [1]. The population-based approach, the averaging in the recombination process, and the rank-based, non-elitist selection are inherent features that make the CMA-ES less vulnerable to noise. However, if the signal to noise ratio is too small, special uncertainty handling is required. Here we use a slightly simplified version of the uncertainty handling heuristics proposed in [7]. It is called *UH-CMA-ES* and relies on adaptive reevaluation of solutions. The method is implemented in Procedure 2. Because the selection process is rank-based, we only care about noise if it changes the ranking of offspring. In our scenario, individuals can be reevaluated and computing the mean or median of several evaluations reduces the level of uncertainty. However, the signal to noise ratio changes in the course of learning. Because every fitness evaluation is time consuming, we implement a strategy that adapts the number of evaluations per individual such that individuals are not evaluated too often, but still often enough that the fitness values can guide the optimization.

We use an algorithm to detect the effective noise by monitoring the stability of the offspring ranking. Following [7], we construct a multi-set $\mathcal{L}$ containing 2λ triples of individuals with two corresponding sampled performance values. Let f_i, $i \in \{1,\ldots,\lambda\}$, be the sampled performance of candidate solution $\boldsymbol{x}_i$. We reevaluate λ_{reev} candidate solutions and define f_i^{reeval} to be the newly obtained performance estimate if individual i was reevaluated and $f_i^{\text{reeval}} = f_i$ otherwise. We define

$$\mathcal{L} \leftarrow ((\boldsymbol{x}_1, f_1, f_1), \ldots, (\boldsymbol{x}_\lambda, f_\lambda, f_\lambda), (\boldsymbol{x}_1, f_1, f_1^{\text{reeval}}), \ldots, (\boldsymbol{x}_\lambda, f_\lambda, f_\lambda^{\text{reeval}})) \ .$$

That is, each policy parameter vector occurs twice in $\mathcal{L}$. In one of the corresponding triples the third component is the new performance estimate (which is always the same as the original estimate for all but λ_{reev} individuals). In the other triple the third component is the same as the second one (i.e., the original performance estimate).

Then we sort $\mathcal{L}$ twice using the first and second performance estimate (the second and the third component of the triples), respectively, and determine the corresponding ranks $\mathrm{rank}_{\mathcal{L}}(i)$ and $\mathrm{rank}^{\text{reeval}}_{\mathcal{L}}(i)$, respectively, of each reevaluated individual $\boldsymbol{x}_i$. Now we compute the *rank change*

$$\Delta_i \leftarrow |\,\mathrm{rank}^{\text{reeval}}_{\mathcal{L}}(i) - \mathrm{rank}_{\mathcal{L}}(i)| - 1 \ .$$

Considering the duplicated population $\mathcal{L}$ for ranking at first appears to be cumbersome and overly complex. However, by doing so it is ensured that the rank change is symmetric in the sense that it does not depend on the order in which the different performance estimates are obtained (e.g., sampled performance estimates such as $f_1 = 16$, $f_2 = 20$, $f_1^{\text{reeval}} = 12$, $f_2^{\text{reeval}} = 14$ lead to the same results as $f_1 = 16$, $f_2 = 14$, $f_1^{\text{reeval}} = 12$, $f_2^{\text{reeval}} = 20$).

The *uncertainty level* s is now defined by

$$s \leftarrow \frac{1}{\lambda_{\text{reev}}} \sum_{i, x_i \text{ reevaluated}} \big(2\Delta_i - \Delta_\theta^{\text{lim}}(\mathrm{rank}^{\text{reeval}}_{\mathcal{L}}(i) - \mathbb{I}\{f_i^{\text{reeval}} > f_i\}) - \Delta_\theta^{\text{lim}}(\mathrm{rank}_{\mathcal{L}}(i) - \mathbb{I}\{f_i > f_i^{\text{reeval}}\})\big) \ .$$

The indicator function $\mathbb{I}$ is one if its argument is true and zero otherwise. The parameter $\theta \in [0, 1]$ (in our experiments set to 0.5 for the swimmer tasks and to 0.2 in all other cases) controls the level of noise we tolerate and $\Delta_\theta^{\text{lim}}(r)$ denotes the $\theta \times 50$ percentile of the possible rank changes (given by the $2\lambda - 1$ values $|1 - r|, |2 - r|, \ldots, |2\lambda - 1 - r|$) when having the original rank r.

If $s > 0$ we increase the number of evaluations in the computation of a fitness value by a factor of α. Otherwise we decrease the number of evaluations by $1/\alpha$. We set $\alpha = 1.5$ in our experiments.

The reevaluation is done before the environmental selection in the standard CMA-ES, which then uses the median of the fitness values of the reevaluated individuals for ranking. The additional fitness evaluations increase the computational costs per generation. However, we reevaluate on average only $\overline{\lambda}_{\text{reev}} = \max(\lambda/10, 2)$ individuals in each generation.

4 Experiments

Benchmark problems. We consider four RL benchmarks taken from the literature. In the mountain car task an underpowered car is to be driven out a valley to the goal state at the hilltop [18]. The state $\boldsymbol{s}$ of the system is given by the position $x \in [-1.2, 0.6]$ of the car and by its current velocity $\dot{x} \in [-0.07, 0.07]$. Actions are discrete forces applied to the car $a \in \{-a_{\text{max}}, 0, a_{\text{max}}\}$. In every time step a reward of -1 is given to the agent. The initial state is uniformly drawn from $\{(-0.5, \dot{x}) | \dot{x} \in [-0.07, 0.07]\}$. A trial is *successful* if the final policy allows the car to reach the hilltop in less than 100 time steps on average. We examine two variants of pole balancing, single-pole balancing as in [16, 9] and double-pole balancing as in [11, 4]. One or two poles are mounted on a 1-dimensional cart and the objective is to balance the poles as long as possible. The state is given by the current position x and velocity $\dot{x}$ of the cart and the current angle ζ (or angles ζ_1, ζ_2) and angular velocity $\dot{\zeta}$ (or velocities $\dot{\zeta_1}$, $\dot{\zeta_2}$). In the single-pole balancing scenario the agents receives a reward of 0 for every time step the cart is close to the center ($|x| < 0.05$) and the pole is almost perpendicular ($|\zeta| < 0.05$), a reward of $-2(N - t)$ (with $N = 500$ as the maximal episode length) if the pole crashes ($|\zeta| > 0.7$) or leaves the allowed x-range ($|x| > 2.4$) at time step t, and a reward of -1 in every other time step. A trial is regarded as successful if the accumulated reward of the final policy is larger than -200 (implying that the pole is balanced at least most of the time). The starting point for an episode is drawn from the set $\{(x, 0, \zeta, 0) | x \in [-2, 2], \zeta \in [-0.6, 0.6]\}$. In the double-pole balancing task the reward is 1 for every time step and we regard a trial as successful if the final policy avoids crashing for 1000 time steps. The system state is initialized with $(x, \dot{x}, \zeta_1, \dot{\zeta_1}, \zeta_2, \dot{\zeta_2}) = (0, 0, \mathrm{N}(1^\circ, 1), 0, 0, 0)$. The highest dimensional task we consider is the swimmer problem [3]. The swimmer consists of n_{c} compartments floating on a liquid surface. The swimmer is supposed to move its center of gravity as fast as possible in a predefined direction. The state description $\boldsymbol{s} = [A_0, \zeta_1, \ldots, \zeta_n, \dot{A}_0, \dot{\zeta_1}, \ldots, \dot{\zeta_n}]^T$ includes the position of the end point of the first compartment A_0 (marking the "head" of the swimmer), the angle of the ith part ($i = 1, \ldots, n$) with respect to the x-axis, the corresponding velocity of the "head" $\dot{A}_0$, and the angular velocities for each part $\dot{\zeta_1}, \ldots, \dot{\zeta_n}$. Actions $\boldsymbol{a} = [a_1, \ldots, a_{n-1}]^T$ are torques applied between body parts. The

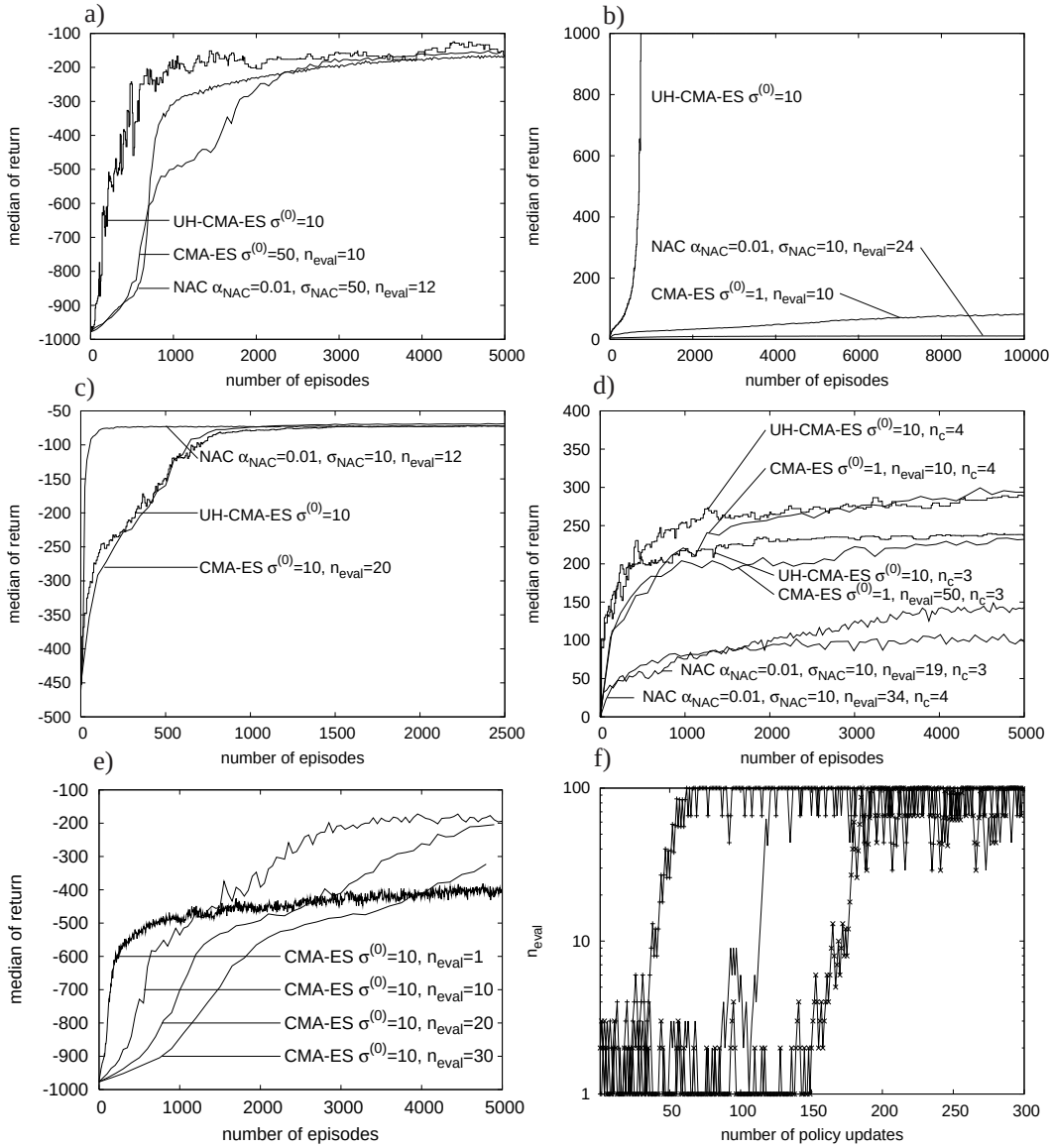

Figure 1: Median of performance over 500 trials for NAC, CMA-ES and UN-CMA-ES for the a) single-pole balancing, b) double-pole balancing and c) mountain car task. d) For the swimmers with 3 and 4 segments the performance median over 20 trials is plotted. e) Performance of the CMA-ES for different sample sizes $n_{\text{eval}} \in \{1, 10, 20, 30\}$ in the single-pole balancing experiment. f) Sample sizes computed by the UH-CMA-ES for three typical trials on the single-pole balancing task are plotted on a logarithmic scale.

reward given to the swimmer is the velocity component of the swimmers center of gravity parallel to the x-axis. We considered two swimmers with $n_{\text{c}} \in \{3, 4\}$. The swimmers initial position is set to 0 and the initial angular velocities are each drawn uniformly from $[0, \pi]$. A swimmer trial is called successful if the average velocity of the swimmers center of gravity is larger than $\frac{3n_{\text{c}}}{40} \frac{\text{m}}{\text{s}}$, i.e., the swimmer covers a distance of at least one and a half of its length in the desired direction in the simulated time span of $20\,\text{s}$.

Experimental setup. In all four tasks uncertainty arises from random start states. We always consider the same type of linear policies in order to allow for a fair comparison. The linear policies considered here are typically used with the NAC [16, 15, 14, 9]). More sophisticated

choices of policy classes certainly improve the performance of the CMA-ES, which for example works fine with non-linear neural networks [11, 13, 5]. Thus all methods operate on the same policy class $\pi^{\text{deter}}_{\boldsymbol{\theta}}(\boldsymbol{s}) = \boldsymbol{\theta}^T \boldsymbol{s}$ with $\boldsymbol{s}, \boldsymbol{\theta} \in \mathbb{R}^n$. For learning, the NAC uses the stochastic policy $\pi^{\text{stoch}}_{\boldsymbol{\theta}}(\boldsymbol{s}, a) = \text{N}(\pi^{\text{deter}}_{\theta}(\boldsymbol{s}), \sigma_{\text{NAC}})$, where the variance σ_{NAC} is considered as an additional adaptive parameter of the PGM. The NAC is evaluated on the corresponding deterministic policy. The policy parameters (except the exploration parameter σ_{NAC} for the NAC) are always initialized with zero. For the swimmer task the action consists of $n-1$ continuous values and we apply an independent linear policy for each action component, thus the search space is $2(n+1)(n-1)$-dimensional. For evaluating the algorithms we use 50 roll-outs, except for the swimmer task where only 20 roll-outs are used. For the mountain car problem and the balancing tasks we test for the CMA-ES all combinations of initial global step size $\sigma^{(0)} \in \{0.1, 1, 5, 10, 15, 25, 50, 100\}$ and sample size $n_{\text{eval}} \in \{1, 10, 20, 30\}$, and for the NAC all combinations of initial exploration $\sigma_{\text{NAC}} \in \{0.1, 1, 5, 10, 15, 25, 50, 100\}$, learning rate $\alpha_{\text{NAC}} \in \{0.1, 0.01, 0.001, 0.0001\}$, and sample size $n_{\text{eval}} \in \{n+2, 2(n+2), 3(n+2)$ (the NAC needs a minimum of $n+2$ roll outs per policy update). For the UH-CMA-ES we vary $\sigma^{(0)} \in \{0.1, 1, 5, 10, 15, 25, 50, 100\}$ and set $n^{(0)}_{\text{eval}} = 1$. Since the swimmer problem is the most computational demanding we restricted $\sigma^{(0)}$ and σ_{NAC} to $\{1, 10, 50\}$ for this task.

Results. In Fig.1 the performances of CMA-ES, NAC, and UH-CMA-ES are shown for the four test scenarios. For all three methods the performance for the best respective parameter configuration is plotted. Both the NAC and the CMA-ES benefit from larger sample sizes n_{eval}. As can be seen in Fig.1e) for single-pole balancing, too small sample sizes result in fast learning in the beginning but lead to early stagnation (i.e., the uncertainty is too high to make progress). On the other hand too large sample sizes allow the CMA-ES to find much better solutions but lead to unnecessarily slow learning. In Fig.1f) the sample sizes chosen by the UH-CMA-ES are shown for typical single trials. At the beginning small sample sizes are used and only when the accuracy is no longer sufficient the sample size is increased. Finally for fine tuning the maximal allowed sample size of $n_{\text{eval}} = 100$ is used most of the time. Altogether the uncertainty handling clearly improves the learning speed, while the CMA-ES and NAC without uncertainty handling perform roughly equally well when looking at the best respective parameter configurations. Only in the mountain car problem the UH-CMA-ES is not faster than both the NAC and the CMA-ES. Here the NAC clearly outperforms both CMA-ES versions, since the NAC benefits from the policy parameter initialization close to a global optimum. The uncertainty handling improves the CMA-ES's results but suffers from discontinuities of the accumulated reward originating in the large initial state interval that includes initial velocities almost sufficient to drive the car directly uphill.

But the uncertainty handling does not only improve the learning speed. For the mountain car task the UH-CMA-ES is successful in more than 80% of the 500 trials for 6 out of 8 tested parameter configurations, while the standard CMA is successful in more than 80% of the trials in only 18 of 32 cases and the NAC in 60 of 96 cases. The same effect can be seen in the two pole balancing tasks. In the single-pole balancing task the UH-CMA-ES is in more than 50% of the trials successful for 4 of 8 parameter configurations, while the CMA-ES and the NAC achieve the same success level with 21 of 32 and with 7 of 96 parameter configurations, respectively. For the double-pole balancing task (which is even without noise a difficult problem when considering only linear policies) the results are most striking. Here the UH-CMA-ES is successful in 100% of the trials for all 8 tested parameter configurations (which is even better than the CMA-ES for a fixed start state) while the CMA-ES and the NAC only achieve a success level of 5% with 5 of 32 and 27 of 96 parameter configurations, respectively. For swimmers both with 3 and 4 compartments the UH-CMA-ES is also robust. The UH-CMA-ES is successful in more than 50% of all trials for 4 of 6 parameter configurations. The CMA-ES achieves the same success level in 16 of 24 cases while the NAC reaches this success level in 0 of 72 cases. Thus the rank based uncertainty handling remarkably increases the learning speed while at the same time improving the robustness compared to the CMA-ES without uncertainty handing, which is already much more robust than the PGM.

5 Discussion and Conclusion

Evolution strategies (ES) are powerful direct policy search methods. One of their main advantages is their ability to cope with noise. Still, random elements in the environment require gathering statistics over several episodes for the evaluation of candidate policies. We added an adaptive uncertainty

handling to evolutionary reinforcement learning. It adjusts the number of roll-outs per evaluation of a policy such that the signal to noise ratio is just high enough for a sufficiently good ranking of candidate policies, which in turn suffices for the ES to find better solutions. The uncertainty handling exploits the advantage of small sample sizes in the beginning and only increases the sample size when a higher accuracy is necessary thus mediating the trade-off between fast learning and sufficient accuracy. This significantly increases both learning speed and robustness and allows to find highly accurate final solutions. Here this has been shown for a variable metric ES, but the uncertainty handling can also be combined with other rank-based reinforcement learning algorithms.

Acknowledgments The authors acknowledge support from the German Federal Ministry of Education and Research within the Bernstein group "The grounding of higher brain function in dynamic neural fields".

References

[1] D. V. Arnold. *Noisy Optimization With Evolution Strategies*. Kluwer Academic Publishers, 2002.

[2] H.-G. Beyer. Evolution strategies. *Scholarpedia*, 2(8):1965, 2007.

[3] R. Coulom. Apprentissage par renforcement utilisant des reseaux de neurones, avec des applications au controle moteur. *These de doctorat, Institut National Polytechnique de Grenoble*, 2002.

[4] F. Gomez, J. Schmidhuber, and R. Miikkulainen. Efficient non-linear control through neuroevolution. In *Proc. European Conference on Machine Learning (ECML 2006)*, volume 4212 of *LNCS*, pages 654–662. Springer-Verlag, 2006.

[5] F. Gomez, J. Schmidhuber, and R. Miikkulainen. Accelerated neural evolution through cooperatively coevolved synapses. *Journal of Machine Learning Research*, 9:937–965, 2008.

[6] N. Hansen, S. D. Müller, and P. Koumoutsakos. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). *Evolutionary Computation*, 11(1):1–18, 2003.

[7] N. Hansen, A. S. P. Niederberger, L. Guzzella, and P. Koumoutsakos. Evolutionary optimization of feedback controllers for thermoacoustic instabilities. In J. F. Morrison, D. M. Birch, and P. Lavoie, editors, *IUTAM Symposium on Flow Control and MEMS*. Springer-Verlag, 2008.

[8] V. Heidrich-Meisner and C. Igel. Evolution strategies for direct policy search. In G. Rudolph, editor, *Parallel Problem Solving from Nature (PPSN X)*, number 5199 in LNCS, pages 428–437. Springer-Verlag, 2008.

[9] V. Heidrich-Meisner and C. Igel. Similarities and differences between policy gradient methods and evolution strategies. In M. Verleysen, editor, *16th European Symposium on Artificial Neural Networks (ESANN)*, pages 149–154. Evere, Belgium: d-side publications, 2008.

[10] V. Heidrich-Meisner and C. Igel. Variable metric reinforcement learning methods applied to the noisy mountain car problem. In S. Girgin et al., editors, *European Workshop on Reinforcement Learning (EWRL 2008)*, number 5323 in LNAI, pages 136–150. Springer-Verlag, 2008.

[11] C. Igel. Neuroevolution for reinforcement learning using evolution strategies. In *Congress on Evolutionary Computation (CEC 2003)*, volume 4, pages 2588–2595. IEEE Press, 2003.

[12] S. Kakade. A natural policy gradient. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems (NIPS14)*. MIT Press, 2002.

[13] A. Pellecchia, C. Igel, J. Edelbrunner, and G. Schöner. Making driver modeling attractive. *IEEE Intelligent Systems*, 20(2):8–12, 2005.

[14] J. Peters and S. Schaal. Natural actor-critic. *Neurocomputing*, 71(7-9):1180–1190, 2008.

[15] J. Peters, S. Vijayakumar, and S. Schaal. Reinforcement learning for humanoid robotics. In *Proc. 3rd IEEE-RAS Int'l Conf. on Humanoid Robots*, pages 29–30, 2003.

[16] M. Riedmiller, J. Peters, and S. Schaal. Evaluation of policy gradient methods and variants on the cart-pole benchmark. In *Proc. IEEE Int'l Symposium on Approximate Dynamic Programming and Reinforcement Learning (ADPRL 2007)*, pages 254–261, 2007.

[17] N. T. Siebel and G. Sommer. Evolutionary reinforcement learning of artificial neural networks. *International Journal of Hybrid Intelligent Systems*, 4(3):171–183, 2007.

[18] R. Sutton and A. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[19] S. Whiteson and P. Stone. Evolutionary function approximation for reinforcement learning. *Journal of Machine Learning Research*, 7:877–917, 2006.