J. Algorithms 64 (2009) 152–168

Contents lists available at ScienceDirect

Journal of Algorithms
Cognition, Informatics and Logic

www.elsevier.com/locate/jalgor

# Neuroevolution strategies for episodic reinforcement learning

Verena Heidrich-Meisner *, Christian Igel

*Institut für Neuroinformatik, Ruhr-Universität Bochum, 44780 Bochum, Germany*

ARTICLE INFO

*Article history:*
Received 30 April 2009
Available online 8 May 2009

*Keywords:*
Reinforcement learning
Evolution strategy
Covariance matrix adaptation
Partially observable Markov decision process
Direct policy search

ABSTRACT

Because of their convincing performance, there is a growing interest in using evolutionary algorithms for reinforcement learning. We propose learning of neural network policies by the covariance matrix adaptation evolution strategy (CMA-ES), a randomized variable-metric search algorithm for continuous optimization. We argue that this approach, which we refer to as CMA Neuroevolution Strategy (CMA-NeuroES), is ideally suited for reinforcement learning, in particular because it is based on ranking policies (and therefore robust against noise), efficiently detects correlations between parameters, and infers a search direction from scalar reinforcement signals. We evaluate the CMA-NeuroES on five different (Markovian and non-Markovian) variants of the common pole balancing problem. The results are compared to those described in a recent study covering several RL algorithms, and the CMA-NeuroES shows the overall best performance.



## 1. Introduction

*Neuroevolution* denotes the application of evolutionary algorithms to the design and learning of neural networks [54,11]. Accordingly, the term *Neuroevolution Strategies* (NeuroESs) refers to evolution strategies applied to neural networks. Evolution strategies are a major branch of evolutionary algorithms [35,40,8,2,7]. Although in principle applicable to general search and optimization problems, most evolution strategies are tailored to real-valued optimization, and thus NeuroESs are usually applied to adapt weights in neural networks.

There is a growing interest in using evolutionary algorithms for RL. "Although not often thought of in this way, genetic algorithms are, in a sense, inherently a reinforcement technique" [51] and indeed evolutionary methods have been successfully applied to RL (see, e.g., [51,32,9,43]) and performed well in comparison with alternative approaches (see [50,14] for recent studies). Still, evolutionary RL is often met with scepticism. The main argument is that a general purpose optimization technique such as an evolutionary algorithm, even if slightly tailored to the learning problem, is not likely to compete with highly specialized methods developed solely for the RL scenario. In this paper, we give strong empirical evidence and summarize some arguments to dispel this concern.

The covariance matrix adaptation evolution strategy (CMA-ES, [19,18,46]) reflects the state-of-the-art in continuous evolutionary optimization [7]. It is a variable-metric method efficiently adapting the metric of the search distribution according to the structure of the search space. This and the rank-based selection makes the CMA-ES ideal for neuroevolution with fixed network structures. The resulting algorithm, the CMA-NeuroES, was proposed for RL in [24] and has been applied successfully in several studies [33,41,47,22,23,21].

In this article, we present the current version of the CMA-NeuroES. We apply the algorithm to the benchmark problems described in [13,14,47] for a fair comparison of the CMA-ES to alternative RL approaches. In [13,14] various RL algorithms are compared on different variants of the pole balancing control problem. The CMA-NeuroES performs well in that study

* Corresponding author.
*E-mail addresses:* Verena.Heidrich-Meisner@neuroinformatik.rub.de (V. Heidrich-Meisner), Christian.Igel@neuroinformatik.rub.de (C. Igel).

0196-6774/$ – see front matter 
doi:10.1016/j.jalgor.2009.04.002

even though the results for the CMA-NeuroES are taken from [24] where a by now outdated version of the CMA-ES was employed. This may bias the comparison. Further, small differences in the benchmark specification could have influenced the results.

In the following section, we present the CMA-ES and its application to neuroevolution. We briefly discuss the CMA-NeuroES in the context of policy gradient methods. Section 3 is devoted to the experiments and their results. The article ends with a discussion and conclusions.

## 2. Variable-metric NeuroES for reinforcement learning

In the standard reinforcement learning (RL) scenario, an agent interacts with its environment at discrete time steps $t$. At each time step the environment is in a state $s_t \in \mathcal{S}$. The agent perceives the environment to be in the observed state $O(s_t) \in \Omega$ and takes an action $a_t \in \mathcal{A}$ according to its policy $\pi : \Omega \to \mathcal{A}$ (we assume deterministic policies throughout this paper). After the execution of $a_t$, the environment makes a possibly stochastic transition to a state $s_{t+1}$ and emits a possibly stochastic numerical reward $r_{t+1} \in \mathbb{R}$. States, actions, transition dynamics, and expected rewards for each transition constitute a Markov decision process (MDP). The objective of the agent is to adapt its policy such that some notion of expected future reward is maximized. The observation of the agent is a possibly stochastic function of the previous state and action. If the agent's observation maps one-to-one to the current state of the environment the MDP is called fully observable and partially observable otherwise.

Most RL algorithms search in the space of value functions that predict future rewards [44]. For example, in temporal-difference learning a state–value function $V : \Omega \to \mathbb{R}$ or a state–action–value function $Q : \Omega \times \mathcal{A} \to \mathbb{R}$ for judging states or state–action pairs, respectively, is learned. The policy $\pi$ is then defined on top of this function. Actor-only algorithms are an alternative RL approach. These methods, which include evolutionary methods [32] and some policy gradient methods [45], search directly in the space of policies. In contrast to temporal difference learning, there is no need to learn to predict future rewards.

When $\Omega$ or $\mathcal{A}$ is too large or generalization from experiences to new states and actions is desired, function approximators such as neural networks are used to model $Q$, $V$, or $\pi$. Here we consider direct search for neural network policies. In the following, we outline the CMA-ES, an evolutionary algorithm for real-valued optimization we propose for neuroevolution and RL. The description follows [46].

### 2.1. Covariance matrix adaptation evolution strategy

Evolution strategies are random search methods [8,7]. They 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. In evolution strategies, this search distribution is parametrized by a set of candidate solutions, the *parent population* with size $\mu$, and by parameters of the variation operators that are used to create new candidate solutions (the *offspring population* with size $\lambda$) from the parent population.

In the following, we present the covariance matrix adaptation evolution strategy (CMA-ES), a variable-metric algorithm for real-valued optimization, and one of the most powerful evolutionary algorithms for continuous search spaces as shown in a recent competition [4,16]. It can be regarded as "state-of-the-art in evolutionary optimization in real-valued $\mathbb{R}^n$ search spaces" [7].

**1** 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), $k = 1$
// $k$ counts number of generations / policy updates:
**2** **repeat**
**3** **for** $l = 1, \ldots, \lambda$ **do** $\boldsymbol{x}_l^{(k+1)} = \boldsymbol{m}^{(k)} + \sigma^{(k)} \boldsymbol{z}_l^{(k)}$ with $\boldsymbol{z}_l^{(k)} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{C}^{(k)})$ // create new offspring
**4** **for** $l = 1, \ldots, \lambda$ **do** $f_l \leftarrow \texttt{performance}(\boldsymbol{x}_l^{(k+1)})$ // evaluate offspring
**5** $\boldsymbol{m}^{(k+1)} \leftarrow \sum_{i=1}^{\mu} w_i \boldsymbol{x}_{i:\lambda}^{(k+1)}$ // selection and recombination
// step size control:
**6** $\boldsymbol{p}_\sigma^{(k+1)} \leftarrow (1 - c_\sigma)\boldsymbol{p}_\sigma^{(k)} + \sqrt{c_\sigma(2 - c_\sigma)\mu_{\text{eff}}}\boldsymbol{C}^{(k)^{-\frac{1}{2}}} \frac{\boldsymbol{m}^{(k+1)} - \boldsymbol{m}^{(k)}}{\sigma^{(k)}}$
**7** $\sigma^{(k+1)} \leftarrow \sigma^{(k)} \exp\left(\frac{c_\sigma}{d_\sigma}\left[\frac{\|\boldsymbol{p}_\sigma^{(k+1)}\|}{\chi_n} - 1\right]\right)$
// covariance matrix update:
**8** $\boldsymbol{p}_c^{(k+1)} \leftarrow (1 - c_c)\boldsymbol{p}_c^{(k)} + \sqrt{c_c(2 - c_c)\mu_{\text{eff}}} \frac{\boldsymbol{m}^{(k+1)} - \boldsymbol{m}^{(k)}}{\sigma^{(k)}}$
**9** $\boldsymbol{C}^{(k+1)} \leftarrow (1 - c_{\text{cov}})\boldsymbol{C}^{(k)} + \frac{c_{\text{cov}}}{\mu_{\text{cov}}} \boldsymbol{p}_c^{(k+1)} \boldsymbol{p}_c^{(k+1)^{\mathrm{T}}} + c_{\text{cov}}\left(1 - \frac{1}{\mu_{\text{cov}}}\right) \sum_{i=1}^{\mu} w_i \boldsymbol{z}_{i:\lambda}^{(k)} \boldsymbol{z}_{i:\lambda}^{(k)^{\mathrm{T}}}$
**10** $k \leftarrow k + 1$
**11** **until** *some stopping criterion is met*

**Algorithm 1.** rank-$\mu$ CMA-ES.

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

$$\boldsymbol{x}_l^{(k+1)} = \underbrace{\boldsymbol{m}^{(k)}}_{\text{recombination}} + \underbrace{\sigma^{(k)} \boldsymbol{z}_l^{(k)}}_{\text{mutation}}.$$

Here $\boldsymbol{z}_l^{(k)} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{C}^{(k)})$ is the realization of a normally distributed random vector with zero mean and covariance matrix $\boldsymbol{C}^{(k)}$ and

$$\boldsymbol{m}^{(k)} = \sum_{l=1}^{\mu} w_l \boldsymbol{x}_{l:\lambda}^{(k)},$$

where $\boldsymbol{x}_{l:\lambda}^{(k)}$ denotes the $l$th best individual among $\boldsymbol{x}_1^{(k)}, \ldots, \boldsymbol{x}_\lambda^{(k)}$. This corresponds to rank-based selection, in which the best $\mu$ of the $\lambda$ 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 quality of the individuals is determined by the function `performance`($\boldsymbol{x}$) (see line 5), which corresponds to the evaluation of the policy (controller) with parameters $\boldsymbol{x}$.

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-$\mu$ 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 $[\boldsymbol{p}_c^{(k+1)}][\boldsymbol{p}_c^{(k+1)}]^{\mathrm{T}}$. The evolution path represents a memory of a successful search direction and thus contains information additional to the current distribution mean. 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*

$$\mu_{\text{eff}} = \left( \sum_{t=1}^{\mu} w_i^2 \right)^{-1}$$

is a normalization constant. The factor $\sqrt{\mu_{\text{eff}}}$ compensates for the loss of variance due to weighted averaging of random variables during recombination. The rank-$\mu$ update aims at making the single steps that were selected in the last iteration more likely by morphing $\boldsymbol{C}^{(k)}$ towards $[\boldsymbol{z}_{i:\lambda}^{(k)}][\boldsymbol{z}_{i:\lambda}^{(k)}]^{\mathrm{T}}$. The complete update now reads

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

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

$$\sigma^{(k+1)} \leftarrow \sigma^{(k)} \exp\left( \frac{c_\sigma}{d_\sigma} \left[ \frac{\|\boldsymbol{p}_\sigma^{(k+1)}\|}{\hat{\chi}_n} - 1 \right] \right),$$

where $\hat{\chi}_n$ denotes the expectation of the $\chi_n$ distribution, and the *conjugate evolution path* is

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

The learning rate is set to $c_\sigma = \frac{\mu_{\text{eff}}+2}{n+\mu_{\text{eff}}+3}$ 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 $\boldsymbol{B}\boldsymbol{D}^{-1}\boldsymbol{B}^{\mathrm{T}}$, where $\boldsymbol{B}\boldsymbol{D}^2\boldsymbol{B}^{\mathrm{T}}$ is an eigendecomposition of $\boldsymbol{C}$ ($\boldsymbol{B}$ is an orthogonal matrix with the eigenvectors of $\boldsymbol{C}$ and $\boldsymbol{D} = \operatorname{diag}(\lambda_1, \ldots, \lambda_n)$ a diagonal matrix with the corresponding eigenvalues) and sampling $\mathcal{N}(\boldsymbol{0}, \boldsymbol{C})$ is done by sampling $\boldsymbol{B}\boldsymbol{D}\mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$.

Sometimes it is advisable to have a lower bound on the step size. To prevent that the search distribution degenerates in one dimension, this bound should be on $\sigma^{(k+1)}\lambda_n^{(k+1)}$, where $\lambda_n^{(k+1)}$ denotes the smallest eigenvalue of $\boldsymbol{C}^{(k+1)}$. If $\sigma^{(k+1)}\lambda_n^{(k+1)}$ falls below a given threshold $\sigma_{\min}$, we simply set $\sigma^{(k+1)} \leftarrow \sigma_{\min}/\lambda_n^{(k+1)}$.

The values of the learning rates and the damping factor are well considered. Their asymptotic behaviors reflect the natural scaling of the adapted entities dependent on the problem dimension, and their exact forms have been derived and validated by experiments on many basic test functions [18]. *They need not be adjusted dependent on the problem and should therefore not be regarded as hyperparameters of the algorithm.* For a more detailed discussion of the learning rates and the damping factor we refer to [19,18,17,46]. Also the population sizes can be set to default values by using the rules $\lambda = \max(4 + \lfloor 3 \ln n \rfloor, 5)$ and $\mu = \lfloor \frac{\lambda}{2} \rfloor$ for offspring and parent population, respectively [18]. If we fix $\boldsymbol{C}^{(1)} = \boldsymbol{I}$ and $\sigma_{\min} = 0$, the only problem dependent hyperparameter to be chosen is the initial global step size $\sigma_{\text{init}} = \sigma^{(1)}$.

The highly efficient use of information and the fast adaptation of $\sigma$ and $\boldsymbol{C}$ make the CMA-ES to be one of the best direct search algorithms for real-valued optimization [7]. For a detailed description of the CMA-ES we refer to the articles by Hansen et al. [19,18,17,46], for a recent performance assessment to [4,16], and for theoretical analyses of evolution strategies to [6,3,27].

### 2.2. CMA-ES and neuroevolution

The good results in real-valued optimization suggest to employ the CMA-ES for adapting the weights of neural networks. In supervised learning, gradient-based optimization of neural network weights usually turns out to be faster than evolutionary optimization (see [42,26,29] for applications of the CMA-ES for supervised neural network learning), although it might be more prone to getting stuck in undesired local minima. In RL, however, evolutionary algorithms have proven to be powerful and competitive approaches (e.g., [32,50,14,11,22,23,21]).

The CMA-ES has been proposed for RL in [24]. In this approach, a parametrized class of policies is chosen a priori. The individuals in the evolution strategy encode policy parameters $\boldsymbol{x}$ accordingly, and the performance function `performance`($\boldsymbol{x}$) (i.e., the fitness function) in Algorithm 1 evaluates the corresponding policy. In the simplest case, this evaluation is just the return (i.e., the potentially weighted sum of rewards over time) at the end of an episode or some statistics of the returns collected from several roll-outs or episodes. That is, our algorithm has been designed for *episodic* RL, which means that the interaction between agent and environment "naturally breaks down into a sequence of separate episodes" [44]. Within an episode, intermediate reward signals are not exploited in the canonical form of our method.

If the class of policies are neural networks, we refer to our approach as CMA Neuroevolution Strategy (CMA-NeuroES). In a more recent study, the CMA-NeuroES (without rank-$\mu$ update) was compared to 8–12 (depending on the task) other RL algorithms including neuroevolution as well as value-function and policy gradient approaches [13]. On the four test problems where the NeuroCMA-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 [33,41] and additional comparisons on RL benchmarks have been conducted by the authors in [22,23,21] using linear policies.

When using a neural network for representing directly the policy of an agent (e.g., a control strategy), the network weights parametrize the space of policies the network can realize. This parametrization is usually complex, there are strong correlations and thus the optimization problem is far from being separable. Hence, the ability of an optimization algorithm to detect dependencies between network parameters seems to be crucial for its performance in this case, and using a variable-metric algorithm such as the CMA-ES is likely to pay off. The CMA-ES efficiently adapts the covariance matrix of the mutation distribution and thereby accounts for correlations between parameters. By doing so, the CMA-ES infers a promising search direction from the scalar reinforcement signals. It is often argued that RL problems are so difficult to solve because the reinforcement signals do not provide a direction for adaptation – in contrast to a gradient in supervised learning. The evolution path in the CMA-ES can be viewed as a means to learn such a direction. This is a decisive difference of the CMA-ES compared to other direct search methods such as canonical genetic algorithms.

### 2.3. NeuroES and policy gradient methods

It is interesting to compare the CMA-NeuroES to policy gradient methods (PGMs), see [22,23,21]. Both search directly in a space of policies with a fixed parametrization, but the CMA-NeuroES is an actor-only method while PGMs often have actor-critic architectures, that is, they maintain an explicit representation of value functions. In contrast to the NeuroES, 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 methods. The CMA-NeuroES perturbs a deterministic policy by mutation and recombination, while in PGMs the random variations are an inherent property of stochastic policies. In the CMA-NeuroES there is only one initial stochastic variation per generation. In contrast, the stochastic policy introduces perturbations in every time step of every episode. While the number $n$ of parameters of the policy determines the $n$-dimensional random variation in evolution strategies, in PGMs the usually lower dimensionality of the action corresponds to the dimensionality of the random perturbations. The adaptation of the global step size and the covariance matrix in the CMA-NeuroES resembles learning the (Fisher) metric in natural PGMs [28,36,34].

We think that a decisive difference between the CMA-NeuroES and PGMs is that the evolution strategy is based on ranking policies and not on the absolute values of performance estimates or even their gradients. We hypothesize that this makes the CMA-NeuroES more robust. This claim is supported by our experiments using linear policies [22,23,21].

In more biological terms (e.g., see [10]), the CMA-NeuroES implements trial-and-error learning based on extrinsic dynamic perturbations of synaptic weights, where the mutations correspond to extrinsic perturbations [20]. In contrast, most

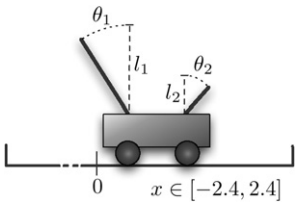


**Fig. 1.** Illustration of the pole balancing task with two poles.

PGMs can be viewed as trial-and-error learning relying on intrinsic perturbations – intrinsic, because the exploration is an inherent property of the stochastic policies – on the level on activities of action encoding neurons.

## 3. Experimental evaluation on pole balancing problems

In the following, we empirically evaluate the CMA-NeuroES on five instances of the common pole balancing benchmark. First, we describe the RL problems. Then we discuss the details of the CMA-NeuroES, in particular the neural network architectures and hyperparameter settings. After that we will present the main results of our experiments.

### 3.1. Pole balancing problems

Pole balancing problems, which are also known as inverted pendulum, pole–cart, broom balancer, or stick balancer problems, are standard benchmark tasks for the design of controllers for unstable systems. They have been used as example problems since the early work on machine intelligence [30], RL [5], and neuroevolution [52,51] as well as in most recent studies on comparing different approaches to RL [14].

The task is to balance one or several poles hinged on a wheeled cart, which can move on a finite length track, by exerting forces either left or right on the cart. The movements of the cart and the poles are constrained within the vertical plane. Control of carts with more than one pole becomes possible when the poles have different lengths. Fig. 1 illustrates the task, and the corresponding equations of motion, which we solve by a fourth-order Runge–Kutta method, are given in Appendix A. The problem of designing a controller for the cart can be viewed as an RL task, where the actions are the applied forces and the perceived state corresponds to the information about the system provided to the controller.

In this article, we consider five different pole balancing scenarios. First, the simple *single pole task*, where only one pole is hinged to the cart and the controller gets as inputs the offset $x$ of the cart from the middle of the track, the velocity $\dot{x}$ of the cart, the angle $\theta_1$ of the pole, and the angular velocity $\dot{\theta}_1$. Second, the more difficult *double pole task* with two poles and two additional inputs, namely the angle $\theta_2$ and the angular velocity $\dot{\theta}_2$ of the second pole. Both tasks are Markovian in the sense that the controller perceives the complete information to predict the system (i.e., the complete real system state). The remaining scenarios are only partially observable and lack the Markov property. In the *single pole without velocities* and *double pole without velocities* tasks again one or two poles have to be balanced, but this time without velocity information. That is, the input to the controllers is just $(x, \theta_1)$ and $(x, \theta_1, \theta_2)$, respectively. For the most difficult task, balancing two poles without velocity information, additionally an alternative performance function is considered. We refer to this fifth scenario as *modified double pole without velocities*. Special care has been taken to make our experiments directly comparable to the results obtained in the comprehensive study by Gomez, Schmidhuber, and Miikkulainen [14]. In the following, we provide some details about the problem settings.

In all simulations, each discrete time step corresponds to 0.02 s. The length of the second, shorter pole is $l_2 = 0.1 l_1$. The initial state of the first, longer pole is always $\zeta_1 = 0.07$ rad, the shorter pole starts from $\zeta_2 = 0$ rad. In all five scenarios a trial is considered to be successful if a controller can balance the pole for $10^5$ time steps. If the cart leaves the track (i.e., $x \notin [-2.4, 2.4]$) or a pole becomes too unstable ($\theta_1, \theta_2 \notin [-0.2 \text{ rad}, 0.2 \text{ rad}]$ in case of the single pole and $\theta_1, \theta_2 \notin [-0.63 \text{ rad}, 0.63 \text{ rad}]$ in case of the double pole tasks) a balancing attempt is regarded as a failure.

In the first four scenarios, the performance – the reinforcement signal – is the number of steps the controller manages to balance the pole(s). In the *modified double pole without velocities* task, which was introduced by [15] and is also used in [12,43,14], the performance function is the weighted sum of two components $0.1 f_1 + 0.9 f_2$ defined over 1000 time steps given by

$$f_1 = t/1000,$$
$$f_2 = \begin{cases} 0 & \text{if } t < 100, \\ \frac{0.75}{\sum_{t'=t-99}^{t} (|x| + |\dot{x}| + |\dot{\theta}_1| + |\dot{\theta}_2|)} & \text{otherwise.} \end{cases}$$

Here $t$ denotes the number of steps the controller is able to balance the poles starting from the same initial state as for the standard fitness function. The first addend rewards successful balancing. The component $f_2$ penalizes oscillations. The idea behind this second term is to prohibit the control strategy from balancing the poles by just moving the cart quickly back and forth from the set of solutions (see [15,12,43,14]).

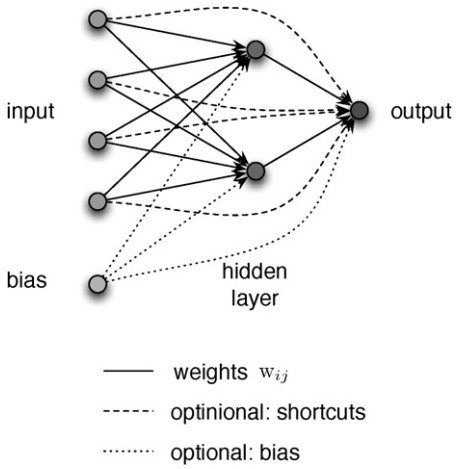


**Fig. 2.** Structure of the feedforward neural networks for the *single pole with velocities* task with 4 input neurons, 2 hidden neurons, and one output neuron with optional bias and shortcut connections.

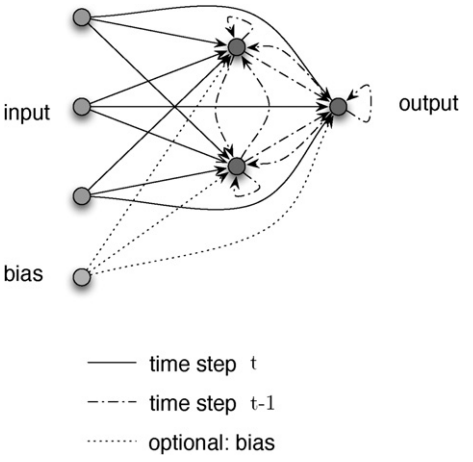


**Fig. 3.** Structure of the recurrent neural networks for the *double pole without velocities* task with 3 input neurons, 2 hidden neurons, and one output neuron with optional bias. The output neuron does not differ functionally from the hidden neurons except that its activity is regarded as the control signal.

### 3.2. Setup of the CMA-NeuroES

Given a learning problem, the CMA-ES has basically only one hyperparameter that may need some tuning, namely the initial global step size $\sigma_{\text{init}}$. However, the performance of the CMA-NeuroES depends on the a priori fixed family of policies, that is, on the neural network structure. In this study, we therefore test many different architectures in order to investigate this dependency.

For the Markovian tasks, standard feedforward networks with a single hidden layer with and without shortcut connections are used. We try architectures with and without bias (threshold) parameters. As the pole balancing problem has inherent symmetries it is reasonable not to use a bias parameter [24].

Recurrent neural networks are used in the partially observable balancing problems without velocity information. Recurrent neural networks store history information which can be fused with the actual input signals to represent *belief states* on which the policy can be defined. We use a simple family of architectures as proposed in [24]. Let $x_i(t)$ for $0 < i \leqslant n_{\text{inputs}}$ be the activation at time step $t$ of the $n_{\text{inputs}}$ input units of a network with $n_{\text{neurons}}$ neurons (including input, hidden, and output units). The activation of the other neurons is given by

$$x_i(t) = f\left(\sum_{j=1}^{n_{\text{inputs}}} w_{ij}x_j(t) + \sum_{j=n_{\text{inputs}}+1}^{n_{\text{neurons}}} w_{ij}x_j(t-1)\right)$$

for $n_{\text{inputs}} < i \leqslant n_{\text{neurons}}$, where the parameters $w_{ij}$ are the weights and $f(a) = a/(|a|+1)$ is a sigmoidal transfer function.

The network structures we use are illustrated in Figs. 2 and 3. All neural networks have a single output neuron and the input signals provided to the networks are appropriately scaled. The number of parameters of each network depends on the structure (feedforward or recurrent), on the number of input neurons (here 4 and 6 for the fully observable single and double pole and 2 and 3 for the partially observable cases, respectively), the number of output neurons (here always one output neuron), the number of hidden neurons (here set to either 1, 2, 4, or 8) and whether bias and shortcuts are used. Table 1 summarizes the resulting degrees of freedom for different network architectures used in our experiments.

We also want to study the influence of the global step size parameter $\sigma_{\text{init}}$ controlling the initial exploration. In the Markovian tasks, we therefore vary $\sigma_{\text{init}} \in \{0.1, 1, 10, 50\}$. As generally the weights of recurrent neural networks are more "sensitive" to changes compared to feedforward architectures, we just consider $\sigma_{\text{init}} \in \{0.1, 1\}$ in the non-Markovian tasks.

**Table 1**
Number of weights (degrees of freedom, DOF) for $i$ input neurons, $h$ hidden neurons, and a single output neuron for feedforward neural networks (FFNNs) and recurrent neural networks (RNNs).

| | shortcuts ($s$) | bias ($b$) | DOF |
|---|---|---|---|
| FFNN | no | no | $h(i+1)$ |
| FFNN | yes | no | $h(i+1)+i$ |
| FFNN | no | yes | $h(i+1)+(h+1)$ |
| FFNN | yes | yes | $h(i+1)+i+(h+1)$ |
| RNN | – | no | $(h+1)^2+(h+1)i$ |
| RNN | – | yes | $(h+1)^2+(h+1)i+(h+1)$ |

**Table 2**
Summary of tested parameter configurations for the CMA-NeuroES in all experiments. The parameters in parentheses are only tested for feedforward networks.

| | Parameter | Values |
|---|---|---|
| number of hidden neurons | $h$ | 1, 2, 4, 8 |
| initial global step size | $\sigma_{\text{init}}$ | 0.1, 1 (10, 25, 50) |
| bias active | $b$ | `yes`, `no` |
| shortcuts active | $s$ | `yes`, `no` |
| lower bound on global step size | $\sigma_{\text{min}}$ | 0, $\frac{1}{2}\sigma_{\text{init}}$ |

We additionally consider a lower bound on the global step size $\sigma_{\text{min}} = \frac{1}{2}\sigma_{\text{init}}$ as suggested for the CMA-NeuroES and recurrent networks in [24]. Table 2 summarizes all configurations.

The experiments can be reproduced using the CMA-ES implementation in the Shark machine learning library [25].

### 3.3. Previous work

We give a brief overview of other RL methods that have been applied to inverted pendulum problems. These methods will serve as a reference for the performance evaluation of the CMA-NeuroES. In the literature there are often small differences regarding the formulation of the task. For example, sometimes the control signal is continuous between $-10$ N and 10 N and sometimes only two actions, the full forces $-10$ N and 10 N, are possible. Often the search spaces (e.g., the possible NN architectures) differ, see also the comparison by Gomez, Schmidhuber, and Miikkulainen [14].

We took the results of other RL algorithm from the literature and consider the same studies as in the extensive comparison by Gomez et al. [14]. In the following, we briefly list the alternative RL strategies. We refer to the original publications and the overview in [14] for details.

*Random weight guessing* (*RWG*) repeatedly draws all weights of the neural network randomly from fixed, a priori fixed intervals, evaluates the corresponding policy, and keeps the best policy found so far [39]. This simple strategy provides a baseline for comparison.

Policy gradient methods [45] are considered, namely *Policy Gradient RL* (*PGRL*) and *Recurrent Policy Gradients* (*RPG*) [53].

We consider $Q$-learning [49] with feedforward NN function approximator, referred to as *Q-Learning with MLP* (*Q-MLP*).

*SARSA*($\lambda$) *with CMAC* (*SARSA-CMAC*) is an on-policy temporal difference method using eligibility traces. In [37] a CMAC (Cerebellar Model Articulation Controller [1]) neural network is used as function approximator to map the continuous state–action space to a discrete space.

We consider several neuroevolution methods. *Conventional Neuroevolution* (*CNE*) refers to an early approach using a genetic algorithm [52], *Cellular Encoding* (*CE*) to the genetic programming approach in [15], *Evolutionary Programming* (*EP*) to the work in [38], *Symbiotic Adaptive Neuro-Evolution* (*SANE*) to [31], *Enforced Sub-Population* (*ESP*) to [12], *NeuroEvolution of Augmenting Topologies* (*NEAT*) to [43], and finally *Cooperative Synapse Neuroevolution* (*CoSyNE*) to the approach favored in [14].

It is important to note that CNE, EP, ESP, CoSyNE, and CMA-NeuroES as well as the non-evolutionary approaches operate on a priori fixed neural network structures. In contrast, CE, SANE, and NEAT evolve the neural network structure and the weights simultaneously. In all neuroevolution methods (1) no discretization of the state spaces is used (as opposed, e.g., to the solutions of pole balancing problems in the seminal papers [30] and [5]), and (2) the function approximators considered here are non-linear in the parameters.

## 4. Results

For each scenario in Table 2, 50 independent trials were conducted. If after $10^5$ evaluations no successful control strategy was found, a trial was regarded as a failure. In such a case, the number of evaluations was set to $10^5$ when computing the average number of pole balancing attempts.

We compare the CMA-NeuroES performance with results from the literature taken from the study by Gomez et al. [14]. To this end, we considered different hyperparameter settings. In the following, we first present the results achieved by the best CMA-NeuroES setting. All results can be found in the appendix showing the robustness of our approach. We want to stress that the results reported in [14] have also been obtained with different hyperparameter settings (for CoSyNE just two settings have been used depending on the Markov property, see Section 4.5).

### 4.1. Fully observable tasks

In the *single pole with velocities* scenario, on average 91 balancing attempts were sufficient to find a control strategy for a network with 2 hidden neurons, shortcuts, and no bias. Thus, the CMA-NeuroES outperformed the previously best method CoSyNE, see Table 3. The complete results for the Markovian single pole balancing are given in Table B.1.

On the *double pole with velocities* the best results, 585 evaluations on average, were achieved with networks having 2 hidden neurons and no bias. This is almost two times better than the best result so far, see Table 4. The complete results for the Markovian double pole balancing are given in Table B.2.

**Table 3**
Comparison of various learning methods on the Markovian single pole balancing problem with continuous control. Results for all methods except CMA-NeuroES are taken from [14] and are averages over 50 independent runs. The RPG results are in parentheses because they refer to experiments in which the goal was to balance the poles just for $10^4$ steps.

| Method | Evaluations |
|---|---|
| PGRL | 28,779 |
| Q-MLP | 2056 |
| SARSA-CMAC | 540 |
| RPG | (863) |
| CNE | 352 |
| SANE | 302 |
| NEAT | 743 |
| ESP | 289 |
| RWG | 199 |
| CoSyNE | 98 |
| *CMA-NeuroES* | 91 |

**Table 4**
Two poles with complete state information. The EP results are taken from [38].

| Method | Evaluations |
|---|---|
| RWG | 474,329 |
| EP | 307,200 |
| CNE | 22,100 |
| SANE | 12,600 |
| Q-MLP | 10,582 |
| RPG | (4981) |
| NEAT | 3600 |
| ESP | 3800 |
| CoSyNE | 954 |
| *CMA-NeuroES* | 585 |

**Table 5**
One pole with incomplete state information.

| Method | Evaluations |
|---|---|
| SARSA-CMAC | 13,562 |
| Q-MLP | 11,331 |
| RWG | 8557 |
| RPG | (1893) |
| NEAT | 1523 |
| SANE | 1212 |
| CNE | 724 |
| ESP | 589 |
| *CMA-NeuroES* | 192 |
| CoSyNE | 127 |

Tables 3 and 4 show that the CMA-NeuroES outperformed all other approaches on the fully observable single pole and double pole problems.

If no bias neuron was used (see Section 4.4), the evolution strategy was quite robust against the choice of the initial global step size $\sigma_{\text{init}}$ on both the single and double pole balancing task, as long as it was not too small ($\sigma_{\text{init}} < 1$), and the number of hidden neurons $h$ was not too large ($h < 8$).

### *4.2. Partially observable tasks*

The complete results for the *single pole without velocities* task are given in Table B.3 and a comparison with other methods in Table 5. The best result, 192 steps on average, was achieved for one hidden neuron without bias, initial global step size $\sigma_{\text{init}} = 50$ and with a lower bound on the global step size.

This is the only scenario considered in this study where CoSyNE outperformed the best CMA-NeuroES, which was however still about three times faster than the next best method.

The complete results for the two non-Markovian double pole balancing tasks are given in Tables B.4 and B.5. The best results (successfully balancing the poles after on average 1141 evaluations for the damping and 860 for the standard performance function) were achieved for networks without bias, a lower bound on the global step size and 1 hidden neuron for both the damping and standard performance function. The optimal initial global step size was with $\sigma_{\text{init}} = 0.1$ in both cases smaller than in the fully observable experiments.

**Table 6**
Two poles with incomplete state information. The table shows the number of evaluations, using the standard performance function and using the damping performance function, respectively.

| Method | Evaluations standard | Evaluations damping |
|---|---|---|
| RWG | 415,209 | 1,232,296 |
| CE | – | (840,000) |
| SANE | 262,700 | 451,612 |
| CNE | 76,906 | 87,623 |
| ESP | 7374 | 26,342 |
| NEAT | – | 6929 |
| RPG | (5649) | – |
| CoSyNE | 1249 | 3416 |
| *CMA-NeuroES* | 860 | 1141 |

**Table 7**
The effect of bias neurons in pole balancing tasks. The respective best results with and without bias are given.

| Task | w/o bias | | | |
|---|---|---|---|---|
| | Evaluations | | | |
| | mean value | median | 25th percentile | 75th percentile |
| single pole with velocities | 91 | 98 | 53 | 120 |
| double pole with velocities | 585 | 491 | 376 | 736 |
| single pole w/o velocities | 192 | 179 | 130 | 255 |
| double pole w/o velocities | 860 | 735 | 563 | 1038 |
| modified double pole w/o velocities | 1141 | 750 | 530 | 1168 |

| Task | with bias | | | |
|---|---|---|---|---|
| | Evaluations | | | |
| | mean value | median | 25th percentile | 75th percentile |
| single pole with velocities | 167 | 129 | 92 | 221 |
| double pole with velocities | 1090 | 864 | 673 | 1272 |
| single pole w/o velocities | 3599 | 2018 | 1560 | 5256 |
| double pole w/o velocities | 2978 | 2794 | 2382 | 3278 |
| modified double pole w/o velocities | 3432 | 3091 | 2393 | 3971 |

As in the fully observable experiments a small number of hidden neurons was more beneficial. The comparison with other methods is shown in Table 6. Here the CMA-NeuroES again outperformed all other algorithms.

### 4.3. Ensuring ongoing exploration

As can be seen in Tables B.1, B.3, B.2, B.4 and B.5, a lower bound $\sigma_{\min} = \frac{1}{2}\sigma_{\text{init}}$ on the global step always improved the performance especially for difficult tasks.

### 4.4. Influence of network structure

It is striking that the architectures without bias parameters clearly outperformed those with bias parameters and the same number of hidden neurons, see Table 7. This was not a mere consequence of the increase in the degrees of freedom when adding thresholds, because larger neural networks without bias gave better results than smaller ones with bias, see Tables B.1 and B.2.

### 4.5. Most successful configurations

Table 8 gives a summary of the most successful configurations across all tasks. The five scenarios inevitably require different network architectures, because of the different number of inputs and the need for recurrent connections in the non-Markovian tasks. However, the question arises whether a *unique* hyperparameter setting can be found for the CMA-NeuroES performing well in comparison to the other methods with tuned hyperparameters.

Reasonably good results (ranking at least second compared to the other methods) were achieved for the following configuration in *all* experiments, see Table 9: 1 hidden neuron, no bias, $\sigma_{\text{init}} = 1$, $\sigma_{\min} = 0.5$ and shortcuts connections for the feedforward networks in the fully observable scenarios.

If we differentiate between the CMA-NeuroES applied to feedforward and recurrent network topologies, as done for CoSyNE in [14], two different "general" settings can be identified as shown in Table 9. Using just these two settings would

**Table 8**
Best CMA-NeuroES hyperparameter configurations for the different pole balancing problems.

| Task | $h$ | $s$ | $b$ | $\sigma_{\text{init}}$ | $\sigma_{\text{min}}$ | Evaluations |
|---|---|---|---|---|---|---|
| single pole with velocities | 2 | yes | no | 50 | 0 | 91 |
| double pole with velocities | 2 | yes | no | 10 | 5 | 585 |
| single pole w/o velocities | 1 | – | no | 1 | 0.5 | 192 |
| double pole w/o velocities | 1 | – | no | 1 | 0.5 | 860 |
| modified double pole w/o velocities | 1 | – | no | 1 | 0.5 | 1141 |

**Table 9**
Best CMA-NeuroES hyperparameter configurations for all five tasks as well as for feedforward and recurrent network architectures, respectively.

| Best overall configuration: $h = 1$, $s$ : yes, $b$ : no, $\sigma_{\text{init}} = 1$, $\sigma_{\text{min}} = 0.5$ | | | | |
|---|---|---|---|---|
| Task | Evaluations | | | |
| | mean value | median | 25th percentile | 75th percentile |
| single pole with velocities | 190 | 148 | 112 | 204 |
| double pole with velocities | 1185 | 525 | 381 | 898 |
| single pole w/o velocities | 192 | 179 | 130 | 255 |
| double pole w/o velocities | 860 | 735 | 563 | 1038 |
| modified double pole w/o velocities | 1141 | 750 | 530 | 1168 |

| Best configuration for both fully observable problems: $h = 2$, $s$ : yes, $b$ : no, $\sigma_{\text{init}} = 10$, $\sigma_{\text{min}} = 5$ | | | | |
|---|---|---|---|---|
| Task | Evaluations | | | |
| | mean value | median | 25th percentile | 75th percentile |
| single pole with velocities | 98 | 94 | 53 | 120 |
| double pole with velocities | 585 | 491 | 376 | 736 |

| Best configuration for all partially observable experiments: $h = 1$, $b$ : no, $\sigma_{\text{init}} = 1$, $\sigma_{\text{min}} = 0.5$ | | | | |
|---|---|---|---|---|
| Task | Evaluations | | | |
| | mean value | median | 25th percentile | 75th percentile |
| single pole w/o velocities | 192 | 179 | 130 | 255 |
| double pole w/o velocities | 860 | 735 | 563 | 1038 |
| modified double pole w/o velocities | 1141 | 750 | 530 | 1168 |

not change the ranking of the CMA-NeuroES in our comparison, except for the single pole with velocities task, where CoSyNE and CMA-NeuroES now perform equally well.

## 5. Discussion

### 5.1. Performance of the CMA-NeuroES

Using standard neural network architectures and the CMA evolution strategy for adapting the weights led to better results for solving pole balancing reinforcement learning tasks than reported so far, only in the non-Markovian single pole balancing scenario the CMA-NeuroES ranked second after CoSyNE.

All algorithms considered in the comparison depend on hyperparameters, which have been chosen depending on the task, and we reported the best results given in the original publications. While the CMA-ES has basically one crucial hyperparameter, the initial global set size, in the CMA-NeuroES we additionally have to choose the neural network topology. However, our experiments show that the choice of the hyperparameters of the CMA-NeuroES depends only weakly on the particular instance of the pole balancing benchmark. Only few hidden neurons are needed to solve the inverted pendulum tasks (actually, for the Markovian tasks no hidden neurons are needed at all [22]). In both fully observable problems 2 hidden neurons gave the best results. For the partially observable balancing problems, a single hidden neuron appears to be the overall preferable choice. Since the output neuron in our recurrent networks does not differ from the hidden neurons except that it is externally observed, this is analogous to the 2 hidden neurons needed in the fully observable experiments. Thus, the preferable structure is similar in all setups. The optimal initial global step size differs between the fully and partially observable tasks. This is a direct consequence of the different properties of feedforward and recurrent neural networks. For the fully observable case the CMA-NeuroES is very robust w.r.t. different choices of $\sigma_{\text{init}}$. For the partially observable cases the step size needs to be small since recurrent neural networks are more sensitive to changes in their weights.

Network architectures without bias parameters turned out to be better suited for pole balancing than those with bias parameters and the same number of hidden neurons. This is not simply due to the increase of the degrees of freedom, but

the reason for these differences in performance is the inherent symmetry of the inverted pendulum tasks, which is broken by bias terms.

The Markovian single pole task can be easily solved by random weight guessing. Choosing a large initial global step size in the CMA-NeuroES mimics this. In this case the search is mainly driven by exploration which is – as the CMA-NeuroES nevertheless finds a successful solution in a small number of evaluations – sufficient to solve this simple task.

Further, fine tuning of the weights seems to be not very important – this may be typical for many RL tasks (e.g., compared to standard regression problems) – and therefore a lower bound on the variance of the mutation distribution of the CMA-NeuroES appears to have no negative effect. Quite the contrary, in almost all scenarios, in particular when ongoing exploration is needed in the absence of selection pressure, bounding the variance improves the results considerably. This indicates that the learning problems have many undesired local optima.

### *5.2. CMA-NeuroES and structure optimization*

Our experiments show that sophisticated neuroevolution algorithms are not necessary for learning good control strategies for pole balancing. Still, the network structure indeed matters, which becomes obvious from the differences between structures with and without bias parameters and between architectures with different numbers of hidden neurons.

Recently, Togelius et al. [47,48] pointed out that additional meaningless input variables (especially noise) seriously impair the behavior of the CMA-NeuroES (and other methods that do not specifically identify meaningful features). Hence, indeed methods are required that evolve both the structure and the weights of neural networks for reinforcement tasks. However, the standard CMA-NeuroES – as a policy gradient method – is limited to a fixed number of object parameters. Of course, the CMA-ES can be used in a nested neuroevolution algorithm, in which the network structure is evolved in an outer loop and the CMA-ES optimizes the weights in an inner loop as in [24]. This approach is implemented in [41,47,48].

### *5.3. Pole balancing problems*

Pole balancing or inverted pendulum problems are the Drosophila of neurocontrol and RL. For this reason, they are well suited for comparing the CMA-NeuroES with other RL methods, because inverted pendulum results are widely available in the literature. Pole balancing is straight-forward to implement (and free implementations are available) and the problem difficulty can be easily tuned by adding poles, changing the lengths of poles, varying the starting conditions, making the environment partially observable, etc.

Of course, concentrating on variants of pole balancing – as done here and in other studies – for comparing algorithms introduces a strong bias. For example, because the intermediate reward signal during a balancing attempt contains no meaningful information, the task does not favor approaches with online learning. Further, our experiments show that evolving the neural network structure does not really pay off when solving pole balancing tasks. Having none or few hidden neurons (and no bias) is already a good solution. In this sense, the pole balancing problems are too simple. This holds especially for Markovian single pole balancing, where random weight guessing gives competitive results and, as argued above, the NeuroCMA-ES performs best with large initial step sizes. The field of RL must definitely move to more ambitious benchmarks and evaluating the CMA-NeuroES on these will be part of our future work.

However, we are confident that our positive evaluation of the CMA-NeuroES is not limited to the family of pole balancing problems. The CMA-NeuroES has already proven to work well on other (toy) benchmark problems from the RL literature [23] and in applications [33,41].

## 6. Conclusions

We presented and discussed the most recent version of the covariance matrix evolution strategy (CMA-ES) applied to learning the weights of neural networks for reinforcement learning (RL), an approach we termed CMA-NeuroES. The algorithm has many appealing properties, in particular the CMA-NeuroES

(1) allows for direct search in policy space and is not restricted to optimizing the policy "indirectly" by adapting state–value or state–action–value functions,
(2) is straightforward to apply and comparatively robust with respect to the tuning of meta-parameters (e.g., we found choosing the right learning rates etc. for temporal-difference learning and policy gradient methods *much* harder than adjusting the CMA-ES [22,23,21]),
(3) is based on ranking policies, which is less susceptible to noise (which arises in RL for several reasons, e.g., because of random rewards and transitions, random initialization, and noisy state observations) compared to estimating a value function or a gradient of a performance measure w.r.t. policy parameters [23],
(4) is a variable-metric algorithm learning an appropriate metric (by means of adapting the covariance matrix and thereby considering correlations between parameters) for searching for better policies,
(5) can be applied if the function approximators are non-differentiable, whereas many other methods require a differentiable structure for gradient-based optimization steps, and
(6) extracts similarly to policy gradient methods a search direction from the scalar reward signals.

We evaluated the CMA-NeuroES on five different variants of the common pole balancing problem. The results were compared to those described in a recent study that looks at several RL algorithms. In all scenarios, the CMA-ES gave the best results, except for one scenario, where it ranked second.

Because of the conceptual reasons listed above and our empirical findings in this and other studies [33,22,23,21], the CMA-NeuroES is our algorithm of choice for episodic RL tasks if episodes are short and/or intermediate reward signals cannot be exploited.

## Acknowledgments

The authors thank Marc Höner for help with the experiments, Faustino Gomez for making his implementation of the pole balancing tasks available and 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".

## Appendix A. Equations of motion

A system with $N$ poles is governed by

$$\ddot{x} = \frac{F - \mu_c \operatorname{sgn}(\dot{x}) + \sum_{i=1}^{N} \tilde{F}_i}{m_c + \sum_{i=1}^{N} \tilde{m}_i},$$

$$\ddot{\theta}_i = -\frac{3}{4\tilde{l}_i}\left(\ddot{x}\cos\theta_i + g\sin\theta_i + \frac{\mu_i \dot{\theta}_i}{m_i \tilde{l}_i}\right),$$

$$\tilde{F}_i = m_i \tilde{l}_i \dot{\theta}_i^2 \sin\theta_i + \frac{3}{4} m_i \cos\theta_i \left(\frac{\mu_i \dot{\theta}_i}{m_i \tilde{l}_i} + g\sin\theta_i\right),$$

$$\tilde{m}_i = m_i\left(1 - \frac{3}{4}\cos^2\theta_i\right)$$

for $i = 1, \ldots, N$, see [52]. Here, $x$ is the distance of the cart from the center of the track, $F$ is the force applied to the cart, and $g = 9.8$ m/sec$^2$ is the acceleration due to gravity. The mass and the coefficient of friction of the cart are denoted by $m_c$ and $\mu_c$, respectively. The variables $m_i$, $\tilde{l}_i$, $l_i$, $\theta_i$, and $\mu_i$ stand for the mass, the half of the length, the length, the angle from the vertical, and the coefficient of friction of the $i$th pole, respectively. The effective force from pole $i$ on the cart is given by $\tilde{F}_i$ and its effective mass by $\tilde{m}_i$. The sign function sgn "inherits" the unit of measurement of its argument.

In our benchmark experiments, we use the most common choices for the parameters of the systems and the most frequently used numerical solution method. In all experiments, $m_c = 1$ kg, $m_1 = 0.1$ kg, $\tilde{l}_1 = 0.5 l_1$, $l_1 = 1$ m, and the width of the track is 4.8 m. In the double pole experiments, $\tilde{l}_2 = 0.1\tilde{l}_1$, $m_2 = 0.1 m_1$. The coefficients of friction are $\mu_c = 5 \cdot 10^{-4}$ Ns/m and $\mu_1 = \mu_2 = 2 \cdot 10^{-6}$ Nms. The dynamical system is numerically solved using fourth-order Runge–Kutta integration with step size $\tau = 0.01$ s.

## Appendix B. Detailed results

**Table B.1**
Average number of episodes required to find a successful control strategy for the fully observable single pole for all hyperparameter configurations.

| $h$ | $s$ | $b$ | $\sigma_{\text{init}}$ | $\sigma_{\text{min}}$ | Evaluations | $h$ | $s$ | $b$ | $\sigma_{\text{init}}$ | $\sigma_{\text{min}}$ | Evaluations |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | no | no | 0.1 | 0 | 10,000.00 | 1 | no | no | 0.1 | 0.05 | 10,000.00 |
| 1 | yes | no | 0.1 | 0 | 4828.38 | 1 | yes | no | 0.1 | 0.05 | 915.08 |
| 1 | no | yes | 0.1 | 0 | 9640.94 | 1 | no | yes | 0.1 | 0.05 | 1854.78 |
| 1 | yes | yes | 0.1 | 0 | 5845.08 | 1 | yes | yes | 0.1 | 0.05 | 1302.62 |
| 1 | no | no | 1 | 0 | 10,000.00 | 1 | no | no | 1 | 0.5 | 10,000.00 |
| 1 | yes | no | 1 | 0 | 375.08 | 1 | yes | no | 1 | 0.5 | 189.10 |
| 1 | no | yes | 1 | 0 | 5063.02 | 1 | no | yes | 1 | 0.5 | 454.46 |
| 1 | yes | yes | 1 | 0 | 273.48 | 1 | yes | yes | 1 | 0.5 | 277.02 |
| 1 | no | no | 10 | 0 | 10,000.00 | 1 | no | no | 10 | 5 | 10,000.00 |
| 1 | yes | no | 10 | 0 | 470.10 | 1 | yes | no | 10 | 5 | 104.70 |
| 1 | no | yes | 10 | 0 | 1487.08 | 1 | no | yes | 10 | 5 | 302.88 |
| 1 | yes | yes | 10 | 0 | 551.76 | 1 | yes | yes | 10 | 5 | 195.56 |
| 1 | no | no | 25 | 0 | 10,000.00 | 1 | no | no | 25 | 12.5 | 10,000.00 |
| 1 | yes | no | 25 | 0 | 266.98 | 1 | yes | no | 25 | 12.5 | 173.82 |
| 1 | no | yes | 25 | 0 | 2779.70 | 1 | no | yes | 25 | 12.5 | 399.34 |
| 1 | yes | yes | 25 | 0 | 348.52 | 1 | yes | yes | 25 | 12.5 | 197.28 |

(*continued on next page*)

**Table B.1** (*continued*)

| $h$ | $s$ | $b$ | $\sigma_{\text{init}}$ | $\sigma_{\text{min}}$ | Evaluations | $h$ | $s$ | $b$ | $\sigma_{\text{init}}$ | $\sigma_{\text{min}}$ | Evaluations |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | no | no | 50 | 0 | 10,000.00 | 1 | no | no | 50 | 25 | 10,000.00 |
| 1 | yes | no | 50 | 0 | 199.30 | 1 | yes | no | 50 | 25 | 181.56 |
| 1 | no | yes | 50 | 0 | 2216.18 | 1 | no | yes | 50 | 25 | 450.38 |
| 1 | yes | yes | 50 | 0 | 166.04 | 1 | yes | yes | 50 | 25 | 166.62 |
| 2 | no | no | 0.1 | 0 | 9489.06 | 2 | no | no | 0.1 | 0.05 | 1837.62 |
| 2 | yes | no | 0.1 | 0 | 4509.50 | 2 | yes | no | 0.1 | 0.05 | 1383.82 |
| 2 | no | yes | 0.1 | 0 | 9695.90 | 2 | no | yes | 0.1 | 0.05 | 3168.28 |
| 2 | yes | yes | 0.1 | 0 | 6220.40 | 2 | yes | yes | 0.1 | 0.05 | 2026.08 |
| 2 | no | no | 1 | 0 | 575.50 | 2 | no | no | 1 | 0.5 | 434.48 |
| 2 | yes | no | 1 | 0 | 197.94 | 2 | yes | no | 1 | 0.5 | 204.94 |
| 2 | no | yes | 1 | 0 | 1776.52 | 2 | no | yes | 1 | 0.5 | 585.56 |
| 2 | yes | yes | 1 | 0 | 487.92 | 2 | yes | yes | 1 | 0.5 | 195.56 |
| 2 | no | no | 10 | 0 | 942.66 | 2 | no | no | 10 | 5 | 426.90 |
| 2 | yes | no | 10 | 0 | 98.20 | 2 | yes | no | 10 | 5 | 98.20 |
| 2 | no | yes | 10 | 0 | 513.80 | 2 | no | yes | 10 | 5 | 302.90 |
| 2 | yes | yes | 10 | 0 | 200.70 | 2 | yes | yes | 10 | 5 | 200.70 |
| 2 | no | no | 25 | 0 | 967.76 | 2 | no | no | 25 | 12.5 | 624.10 |
| 2 | yes | no | 25 | 0 | 102.74 | 2 | yes | no | 25 | 12.5 | 102.74 |
| 2 | no | yes | 25 | 0 | 777.98 | 2 | no | yes | 25 | 12.5 | 377.48 |
| 2 | yes | yes | 25 | 0 | 234.98 | 2 | yes | yes | 25 | 12.5 | 227.82 |
| 2 | no | no | 50 | 0 | 2130.84 | 2 | no | no | 50 | 25 | 902.74 |
| 2 | yes | no | 50 | 0 | 90.52 | 2 | yes | no | 50 | 25 | 90.52 |
| 2 | no | yes | 50 | 0 | 885.44 | 2 | no | yes | 50 | 25 | 355.2 |
| 2 | yes | yes | 50 | 0 | 370.5 | 2 | yes | yes | 50 | 25 | 192.76 |
| 4 | no | no | 0.1 | 0 | 9009.24 | 4 | no | no | 0.1 | 0.05 | 3239.40 |
| 4 | yes | no | 0.1 | 0 | 3509.70 | 4 | yes | no | 0.1 | 0.05 | 2139.38 |
| 4 | no | yes | 0.1 | 0 | 8677.60 | 4 | no | yes | 0.1 | 0.05 | 4799.40 |
| 4 | yes | yes | 0.1 | 0 | 4494.96 | 4 | yes | yes | 0.1 | 0.05 | 2884.16 |
| 4 | no | no | 1 | 0 | 505.62 | 4 | no | no | 1 | 0.5 | 343.10 |
| 4 | yes | no | 1 | 0 | 237.54 | 4 | yes | no | 1 | 0.5 | 237.54 |
| 4 | no | yes | 1 | 0 | 638.66 | 4 | no | yes | 1 | 0.5 | 599.40 |
| 4 | yes | yes | 1 | 0 | 334.96 | 4 | yes | yes | 1 | 0.5 | 328.60 |
| 4 | no | no | 10 | 0 | 150.28 | 4 | no | no | 10 | 5 | 150.28 |
| 4 | yes | no | 10 | 0 | 107.70 | 4 | yes | no | 10 | 5 | 107.70 |
| 4 | no | yes | 10 | 0 | 281.14 | 4 | no | yes | 10 | 5 | 281.14 |
| 4 | yes | yes | 10 | 0 | 342.86 | 4 | yes | yes | 10 | 5 | 319.98 |
| 4 | no | no | 25 | 0 | 151.00 | 4 | no | no | 25 | 12.5 | 151.00 |
| 4 | yes | no | 25 | 0 | 118.18 | 4 | yes | no | 25 | 12.5 | 118.18 |
| 4 | no | yes | 25 | 0 | 314.36 | 4 | no | yes | 25 | 12.5 | 315.70 |
| 4 | yes | yes | 25 | 0 | 306.70 | 4 | yes | yes | 25 | 12.5 | 307.62 |
| 4 | no | no | 50 | 0 | 151.34 | 4 | no | no | 50 | 25 | 151.34 |
| 4 | yes | no | 50 | 0 | 125.46 | 4 | yes | no | 50 | 25 | 125.46 |
| 4 | no | yes | 50 | 0 | 316.60 | 4 | no | yes | 50 | 25 | 315.64 |
| 4 | yes | yes | 50 | 0 | 272.94 | 4 | yes | yes | 50 | 25 | 272.94 |
| 8 | no | no | 0.1 | 0 | 8166.40 | 8 | no | no | 0.1 | 0.05 | 5842.00 |
| 8 | yes | no | 0.1 | 0 | 4259.08 | 8 | yes | no | 0.1 | 0.05 | 3509.08 |
| 8 | no | yes | 0.1 | 0 | 8302.12 | 8 | no | yes | 0.1 | 0.05 | 7610.10 |
| 8 | yes | yes | 0.1 | 0 | 5537.64 | 8 | yes | yes | 0.1 | 0.05 | 5001.86 |
| 8 | no | no | 1 | 0 | 366.90 | 8 | no | no | 1 | 0.5 | 366.90 |
| 8 | yes | no | 1 | 0 | 310.92 | 8 | yes | no | 1 | 0.5 | 310.92 |
| 8 | no | yes | 1 | 0 | 576.54 | 8 | no | yes | 1 | 0.5 | 574.16 |
| 8 | yes | yes | 1 | 0 | 421.72 | 8 | yes | yes | 1 | 0.5 | 421.72 |
| 8 | no | no | 10 | 0 | 137.06 | 8 | no | no | 10 | 5 | 137.06 |
| 8 | yes | no | 10 | 0 | 138.4 | 8 | yes | no | 10 | 5 | 138.40 |
| 8 | no | yes | 10 | 0 | 386.46 | 8 | no | yes | 10 | 5 | 386.46 |
| 8 | yes | yes | 10 | 0 | 282.98 | 8 | yes | yes | 10 | 5 | 282.98 |
| 8 | no | no | 25 | 0 | 173.24 | 8 | no | no | 25 | 12.5 | 173.24 |
| 8 | yes | no | 25 | 0 | 140.58 | 8 | yes | no | 25 | 12.5 | 140.58 |
| 8 | no | yes | 25 | 0 | 326.90 | 8 | no | yes | 25 | 12.5 | 326.90 |
| 8 | yes | yes | 25 | 0 | 326.60 | 8 | yes | yes | 25 | 12.5 | 326.60 |
| 8 | no | no | 50 | 0 | 200.34 | 8 | no | no | 50 | 25 | 200.34 |
| 8 | yes | no | 50 | 0 | 144.62 | 8 | yes | no | 50 | 25 | 144.62 |
| 8 | no | yes | 50 | 0 | 352.30 | 8 | no | yes | 50 | 25 | 352.30 |
| 8 | yes | yes | 50 | 0 | 316.24 | 8 | yes | yes | 50 | 25 | 316.24 |

**Table B.2**
Average number of episodes required to find a successful control strategy for the double pole with velocities task for all configurations.

| $h$ | $s$ | $b$ | $\sigma_{\text{init}}$ | $\sigma_{\text{min}}$ | Evaluations | $h$ | $s$ | $b$ | $\sigma_{\text{init}}$ | $\sigma_{\text{min}}$ | Evaluations |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | no | no | 0.1 | 0 | 10,000.00 | 1 | no | no | 0.1 | 0.05 | 10,000.00 |
| 1 | yes | no | 0.1 | 0 | 6850.80 | 1 | yes | no | 0.1 | 0.05 | 2660.02 |
| 1 | no | yes | 0.1 | 0 | 10,000.00 | 1 | no | yes | 0.1 | 0.05 | 3452.70 |
| 1 | yes | yes | 0.1 | 0 | 7158.88 | 1 | yes | yes | 0.1 | 0.05 | 3062.98 |
| 1 | no | no | 1 | 0 | 10,000.00 | 1 | no | no | 1 | 0.5 | 10,000.00 |
| 1 | yes | no | 1 | 0 | 2019.40 | 1 | yes | no | 1 | 0.5 | 1184.60 |
| 1 | no | yes | 1 | 0 | 7728.62 | 1 | no | yes | 1 | 0.5 | 1786.82 |
| 1 | yes | yes | 1 | 0 | 2691.76 | 1 | yes | yes | 1 | 0.5 | 1090.36 |
| 1 | no | no | 10 | 0 | 10,000.00 | 1 | no | no | 10 | 5 | 10,000.00 |
| 1 | yes | no | 10 | 0 | 1232.76 | 1 | yes | no | 10 | 5 | 1087.92 |
| 1 | no | yes | 10 | 0 | 6493.36 | 1 | no | yes | 10 | 5 | 1227.24 |
| 1 | yes | yes | 10 | 0 | 2982.46 | 1 | yes | yes | 10 | 5 | 1245.28 |
| 1 | no | no | 25 | 0 | 10,000.00 | 1 | no | no | 25 | 12.5 | 10,000.00 |
| 1 | yes | no | 25 | 0 | 1431.94 | 1 | yes | no | 25 | 12.5 | 1289.02 |
| 1 | no | yes | 25 | 0 | 5013.10 | 1 | no | yes | 25 | 12.5 | 1262.40 |
| 1 | yes | yes | 25 | 0 | 2527.38 | 1 | yes | yes | 25 | 12.5 | 1247.18 |
| 1 | no | no | 50 | 0 | 10,000.00 | 1 | no | no | 50 | 25 | 10,000.00 |
| 1 | yes | no | 50 | 0 | 2456.80 | 1 | yes | no | 50 | 25 | 1897.66 |
| 1 | no | yes | 50 | 0 | 5940.80 | 1 | no | yes | 50 | 25 | 1342.88 |
| 1 | yes | yes | 50 | 0 | 3417.16 | 1 | yes | yes | 50 | 25 | 1580.76 |
| 2 | no | no | 0.1 | 0 | 9563.16 | 2 | no | no | 0.1 | 0.05 | 3513.48 |
| 2 | yes | no | 0.1 | 0 | 6047.82 | 2 | yes | no | 0.1 | 0.05 | 3136.26 |
| 2 | no | yes | 0.1 | 0 | 9401.14 | 2 | no | yes | 0.1 | 0.05 | 5081.20 |
| 2 | yes | yes | 0.1 | 0 | 6723.30 | 2 | yes | yes | 0.1 | 0.05 | 3849.96 |
| 2 | no | no | 1 | 0 | 2925.06 | 2 | no | no | 1 | 0.5 | 1186.08 |
| 2 | yes | no | 1 | 0 | 1169.24 | 2 | yes | no | 1 | 0.5 | 860.76 |
| 2 | no | yes | 1 | 0 | 5074.48 | 2 | no | yes | 1 | 0.5 | 1688.66 |
| 2 | yes | yes | 1 | 0 | 2170.10 | 2 | yes | yes | 1 | 0.5 | 1388.80 |
| 2 | no | no | 10 | 0 | 2249.08 | 2 | no | no | 10 | 5 | 884.96 |
| 2 | yes | no | 10 | 0 | 635.40 | 2 | yes | no | 10 | 5 | 584.84 |
| 2 | no | yes | 10 | 0 | 3290.10 | 2 | no | yes | 10 | 5 | 1654.32 |
| 2 | yes | yes | 10 | 0 | 1452.04 | 2 | yes | yes | 10 | 5 | 1367.58 |
| 2 | no | no | 25 | 0 | 1522.62 | 2 | no | no | 25 | 12.5 | 892.12 |
| 2 | yes | no | 25 | 0 | 1507.36 | 2 | yes | no | 25 | 12.5 | 793.44 |
| 2 | no | yes | 25 | 0 | 2787.02 | 2 | no | yes | 25 | 12.5 | 1478.48 |
| 2 | yes | yes | 25 | 0 | 2229.42 | 2 | yes | yes | 25 | 12.5 | 1480.44 |
| 2 | no | no | 50 | 0 | 2336.40 | 2 | no | no | 50 | 25 | 933.66 |
| 2 | yes | no | 50 | 0 | 1257.04 | 2 | yes | no | 50 | 25 | 925.08 |
| 2 | no | yes | 50 | 0 | 2075.28 | 2 | no | yes | 50 | 25 | 1251.88 |
| 2 | yes | yes | 50 | 0 | 2726.80 | 2 | yes | yes | 50 | 25 | 1627.92 |
| 4 | no | no | 0.1 | 0 | 8849.28 | 4 | no | no | 0.1 | 0.05 | 6715.94 |
| 4 | yes | no | 0.1 | 0 | 4994.86 | 4 | yes | no | 0.1 | 0.05 | 4115.16 |
| 4 | no | yes | 0.1 | 0 | 8900.14 | 4 | no | yes | 0.1 | 0.05 | 8574.42 |
| 4 | yes | yes | 0.1 | 0 | 6964.08 | 4 | yes | yes | 0.1 | 0.05 | 5924.36 |
| 4 | no | no | 1 | 0 | 1383.24 | 4 | no | no | 1 | 0.5 | 1060.70 |
| 4 | yes | no | 1 | 0 | 732.70 | 4 | yes | no | 1 | 0.5 | 723.26 |
| 4 | no | yes | 1 | 0 | 3299.18 | 4 | no | yes | 1 | 0.5 | 2257.82 |
| 4 | yes | yes | 1 | 0 | 1599.24 | 4 | yes | yes | 1 | 0.5 | 1505.52 |
| 4 | no | no | 10 | 0 | 767.78 | 4 | no | no | 10 | 5 | 775.0 |
| 4 | yes | no | 10 | 0 | 661.76 | 4 | yes | no | 10 | 5 | 658.84 |
| 4 | no | yes | 10 | 0 | 1808.52 | 4 | no | yes | 10 | 5 | 1573.64 |
| 4 | yes | yes | 10 | 0 | 1660.78 | 4 | yes | yes | 10 | 5 | 1618.88 |
| 4 | no | no | 25 | 0 | 624.08 | 4 | no | no | 25 | 12.5 | 630.66 |
| 4 | yes | no | 25 | 0 | 741.64 | 4 | yes | no | 25 | 12.5 | 910.36 |
| 4 | no | yes | 25 | 0 | 2312.18 | 4 | no | yes | 25 | 12.5 | 1912.42 |
| 4 | yes | yes | 25 | 0 | 1747.10 | 4 | yes | yes | 25 | 12.5 | 1499.92 |
| 4 | no | no | 50 | 0 | 857.12 | 4 | no | no | 50 | 25 | 752.02 |
| 4 | yes | no | 50 | 0 | 667.50 | 4 | yes | no | 50 | 25 | 668.98 |
| 4 | no | yes | 50 | 0 | 2031.04 | 4 | no | yes | 50 | 25 | 1755.52 |
| 4 | yes | yes | 50 | 0 | 1941.06 | 4 | yes | yes | 50 | 25 | 1775.32 |
| 8 | no | no | 0.1 | 0 | 8048.44 | 8 | no | no | 0.1 | 0.05 | 7866.22 |
| 8 | yes | no | 0.1 | 0 | 6249.64 | 8 | yes | no | 0.1 | 0.05 | 6078.66 |
| 8 | no | yes | 0.1 | 0 | 9514.60 | 8 | no | yes | 0.1 | 0.05 | 9514.60 |
| 8 | yes | yes | 0.1 | 0 | 7486.38 | 8 | yes | yes | 0.1 | 0.05 | 7492.06 |

(*continued on next page*)

**Table B.2** (*continued*)

| $h$ | $s$ | $b$ | $\sigma_{\text{init}}$ | $\sigma_{\text{min}}$ | Evaluations | $h$ | $s$ | $b$ | $\sigma_{\text{init}}$ | $\sigma_{\text{min}}$ | Evaluations |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 8 | no | no | 1 | 0 | 1104.84 | 8 | no | no | 1 | 0.5 | 1094.58 |
| 8 | yes | no | 1 | 0 | 931.32 | 8 | yes | no | 1 | 0.5 | 931.32 |
| 8 | no | yes | 1 | 0 | 2425.66 | 8 | no | yes | 1 | 0.5 | 2416.74 |
| 8 | yes | yes | 1 | 0 | 1731.22 | 8 | yes | yes | 1 | 0.5 | 1729.08 |
| 8 | no | no | 10 | 0 | 856.12 | 8 | no | no | 10 | 5 | 856.12 |
| 8 | yes | no | 10 | 0 | 753.54 | 8 | yes | no | 10 | 5 | 757.26 |
| 8 | no | yes | 10 | 0 | 2301.90 | 8 | no | yes | 10 | 5 | 2277.9 |
| 8 | yes | yes | 10 | 0 | 1899.72 | 8 | yes | yes | 10 | 5 | 1865.46 |
| 8 | no | no | 25 | 0 | 873.76 | 8 | no | no | 25 | 12.5 | 873.76 |
| 8 | yes | no | 25 | 0 | 794.80 | 8 | yes | no | 25 | 12.5 | 805.28 |
| 8 | no | yes | 25 | 0 | 2228.40 | 8 | no | yes | 25 | 12.5 | 2170.78 |
| 8 | yes | yes | 25 | 0 | 2444.38 | 8 | yes | yes | 25 | 12.5 | 2247.14 |
| 8 | no | no | 50 | 0 | 969.32 | 8 | no | no | 50 | 25 | 960.48 |
| 8 | yes | no | 50 | 0 | 820.96 | 8 | yes | no | 50 | 25 | 820.96 |
| 8 | no | yes | 50 | 0 | 2373.04 | 8 | no | yes | 50 | 25 | 2287.92 |
| 8 | yes | yes | 50 | 0 | 2263.80 | 8 | yes | yes | 50 | 25 | 2241.62 |

**Table B.3**
Average number of episodes required to find a successful control strategy for the non-Markovian single pole balancing task for all configurations.

| $h$ | $b$ | $\sigma_{\text{init}}$ | $\sigma_{\text{min}}$ | Evaluations | $h$ | $b$ | $\sigma_{\text{init}}$ | $\sigma_{\text{min}}$ | Evaluations |
|---|---|---|---|---|---|---|---|---|---|
| 1 | no | 0.1 | 0 | 570.34 | 1 | no | 0.1 | 0.05 | 396.00 |
| 1 | yes | 0.1 | 0 | 9481.74 | 1 | yes | 0.1 | 0.05 | 3599.06 |
| 1 | no | 1 | 0 | 413.30 | 1 | no | 1 | 0.5 | 192.46 |
| 1 | yes | 1 | 0 | 9613.96 | 1 | yes | 1 | 0.5 | 5587.86 |
| 2 | no | 0.1 | 0 | 368.04 | 2 | no | 0.1 | 0.05 | 368.04 |
| 2 | yes | 0.1 | 0 | 9657.04 | 2 | yes | 0.1 | 0.05 | 5515.36 |
| 2 | no | 1 | 0 | 448.20 | 2 | no | 1 | 0.5 | 338.42 |
| 2 | yes | 1 | 0 | 10,000.00 | 2 | yes | 1 | 0.5 | 8977.28 |
| 4 | no | 0.1 | 0 | 527.78 | 4 | no | 0.1 | 0.05 | 527.78 |
| 4 | yes | 0.1 | 0 | 8675.30 | 4 | yes | 0.1 | 0.05 | 7892.24 |
| 4 | no | 1 | 0 | 758.22 | 4 | no | 1 | 0.5 | 803.64 |
| 4 | yes | 1 | 0 | 10,000.00 | 4 | yes | 1 | 0.5 | 10,000.00 |
| 8 | no | 0.1 | 0 | 795.98 | 8 | no | 0.1 | 0.05 | 795.98 |
| 8 | yes | 0.1 | 0 | 9280.78 | 8 | yes | 0.1 | 0.05 | 9223.82 |
| 8 | no | 1 | 0 | 3319.82 | 8 | no | 1 | 0.5 | 3415.90 |
| 8 | yes | 1 | 0 | 9946.20 | 8 | yes | 1 | 0.5 | 10,000.00 |

**Table B.4**
Average number of episodes required to find a successful control strategy for the non-Markovian double pole standard performance function for all configurations.

| $h$ | $b$ | $\sigma_{\text{init}}$ | $\sigma_{\text{min}}$ | Evaluations | $h$ | $b$ | $\sigma_{\text{init}}$ | $\sigma_{\text{min}}$ | Evaluations |
|---|---|---|---|---|---|---|---|---|---|
| 1 | no | 0.1 | 0 | 6031.80 | 1 | no | 0.1 | 0.05 | 2209.00 |
| 1 | yes | 0.1 | 0 | 9105.12 | 1 | yes | 0.1 | 0.05 | 2977.92 |
| 1 | no | 1 | 0 | 984.60 | 1 | no | 1 | 0.5 | 859.60 |
| 1 | yes | 1 | 0 | 6906.16 | 1 | yes | 1 | 0.5 | 3660.18 |
| 2 | no | 0.1 | 0 | 4750.72 | 2 | no | 0.1 | 0.05 | 2536.32 |
| 2 | yes | 0.1 | 0 | 8752.78 | 2 | yes | 0.1 | 0.05 | 5790.64 |
| 2 | no | 1 | 0 | 1920.24 | 2 | no | 1 | 0.5 | 1640.24 |
| 2 | yes | 1 | 0 | 6772.8 | 2 | yes | 1 | 0.5 | 4822.66 |
| 4 | no | 0.1 | 0 | 3877.90 | 4 | no | 0.1 | 0.05 | 3375.00 |
| 4 | yes | 0.1 | 0 | 9149.90 | 4 | yes | 0.1 | 0.05 | 8760.30 |
| 4 | no | 1 | 0 | 2645.80 | 4 | no | 1 | 0.5 | 2756.90 |
| 4 | yes | 1 | 0 | 7495.10 | 4 | yes | 1 | 0.5 | 7179.20 |
| 8 | no | 0.1 | 0 | 5692.92 | 8 | no | 0.1 | 0.05 | 5608.68 |
| 8 | yes | 0.1 | 0 | 9826.00 | 8 | yes | 0.1 | 0.05 | 9933.68 |
| 8 | no | 1 | 0 | 5696.76 | 8 | no | 1 | 0.5 | 5356.20 |
| 8 | yes | 1 | 0 | 9809.60 | 8 | yes | 1 | 0.5 | 9830.48 |

**Table B.5**
Average number of episodes required to find a successful control strategy for the non-Markovian double pole with damping performance function for all configurations.

| $h$ | $b$ | $\sigma_{\text{init}}$ | $\sigma_{\text{min}}$ | Evaluations | $h$ | $b$ | $\sigma_{\text{init}}$ | $\sigma_{\text{min}}$ | Evaluations |
|---|---|---|---|---|---|---|---|---|---|
| 1 | no | 0.1 | 0 | 7687.00 | 1 | no | 0.1 | 0.05 | 2386.00 |
| 1 | yes | 0.1 | 0 | 8959.70 | 1 | yes | 0.1 | 0.05 | 3432.02 |
| 1 | no | 1 | 0 | 2170.20 | 1 | no | 1 | 0.5 | 1141.00 |
| 1 | yes | 1 | 0 | 7711.28 | 1 | yes | 1 | 0.5 | 4348.36 |
| 2 | no | 0.1 | 0 | 5102.88 | 2 | no | 0.1 | 0.05 | 2649.44 |
| 2 | yes | 0.1 | 0 | 9227.38 | 2 | yes | 0.1 | 0.05 | 5183.42 |
| 2 | no | 1 | 0 | 2022.32 | 2 | no | 1 | 0.5 | 1907.68 |
| 2 | yes | 1 | 0 | 7177.02 | 2 | yes | 1 | 0.5 | 5184.48 |
| 4 | no | 0.1 | 0 | 3827.60 | 4 | no | 0.1 | 0.05 | 3379.20 |
| 4 | yes | 0.1 | 0 | 9182.70 | 4 | yes | 0.1 | 0.05 | 8839.60 |
| 4 | no | 1 | 0 | 2951.80 | 4 | no | 1 | 0.5 | 3242.90 |
| 4 | yes | 1 | 0 | 8513.60 | 4 | yes | 1 | 0.5 | 8326.00 |
| 8 | no | 0.1 | 0 | 4111.56 | 8 | no | 0.1 | 0.05 | 4104.36 |
| 8 | yes | 0.1 | 0 | 9766.92 | 8 | yes | 0.1 | 0.05 | 9813.76 |
| 8 | no | 1 | 0 | 5277.32 | 8 | no | 1 | 0.5 | 5583.92 |
| 8 | yes | 1 | 0 | 9565.16 | 8 | yes | 1 | 0.5 | 9529.16 |

## References


[1] J.S. Albus, A new approach to manipulator control: The cerebellar model articulation controller (CMAC), Trans. ASME J. Dyn. Syst. Meas. Control 97 (1975) 220–227.
[2] D.V. Arnold, Noisy Optimization with Evolution Strategies, Kluwer Academic Publishers, 2002.
[3] A. Auger, Convergence results for the $(1, \lambda)$-SA-ES using the theory of $\varphi$-irreducible Markov chains, Theoret. Comput. Sci. 334 (1) (2005) 35–69.
[4] A. Auger, N. Hansen, A restart CMA evolution strategy with increasing population size, in: Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2005), IEEE Press, 2005, pp. 1769–1776.
[5] A.G. Barto, R.S. Sutton, C.W. Anderson, Neuronlike elements that can solve difficult learning control problems, IEEE Trans. Syst. Man Cybernet. 13 (5) (1983) 835–846.
[6] H.-G. Beyer, The Theory of Evolution Strategies, Nat. Comput. Ser., Springer-Verlag, 2001.
[7] H.-G. Beyer, Evolution strategies, Scholarpedia 2 (8) (2007) 1965.
[8] H.-G. Beyer, H.-P. Schwefel, Evolution strategies: A comprehensive introduction, Nat. Comput. 1 (1) (2002) 3–52.
[9] K. Chellapilla, D.B. Fogel, Evolution neural networks, games, and intelligence, Proc. IEEE 87 (9) (1999) 1471–1496.
[10] I.R. Fiete, M.S. Fee, H.S. Seung, Model of birdsong learning based on gradient estimation by dynamic perturbation of neural conductances, J. Neurophysiology 98 (4) (2007) 2038.
[11] D. Floreano, P. Dürr, C. Mattiussi, Neuroevolution: From architectures to learning, Evol. Intelligence 1 (1) (2008) 47–62.
[12] F. Gomez, R. Miikkulainen, Solving non-markovian tasks with neuroevolution, in: T. Dean (Ed.), Proceeding of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI), Morgan Kaufmann, 1999, pp. 1356–1361.
[13] F. Gomez, J. Schmidhuber, R. Miikkulainen, Efficient non-linear control through neuroevolution, in: Proc. European Conference on Machine Learning (ECML 2006), in: Lecture Notes in Comput. Sci., vol. 4212, Springer-Verlag, 2006, pp. 654–662.
[14] F. Gomez, J. Schmidhuber, R. Miikkulainen, Accelerated neural evolution through cooperatively coevolved synapses, J. Mach. Learn. Res. 9 (2008) 937–965.
[15] F. Gruau, D. Whitley, L. Pyeatt, A comparison between cellular encoding and direct encoding for genetic neural networks, in: J.R. Koza, D.E. Goldberg, D.B. Fogel, R.L. Riol (Eds.), Genetic Programming 1996: Proceedings of the First Annual Conference, MIT Press, 1996, pp. 81–89.
[16] N. Hansen, http://www.bionik.tu-berlin.de/user/niko/cec2005.html, 2006.
[17] N. Hansen, The CMA evolution strategy: A comparing review, in: J.A. Lozano, P. Larranaga, I. Inza, E. Bengoetxea (Eds.), Towards a New Evolutionary Computation. Advances on Estimation of Distribution Algorithms, Springer-Verlag, 2006, pp. 75–102.
[18] N. Hansen, S.D. Müller, P. Koumoutsakos, Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES), Evol. Comput. 11 (1) (2003) 1–18.
[19] N. Hansen, A. Ostermeier, Completely derandomized self-adaptation in evolution strategies, Evol. Comput. 9 (2) (2001) 159–195.
[20] V. Heidrich-Meisner, C. Igel, Learning behavioral policies using extrinsic perturbations on the level of synapses, in: Frontiers in Computational Neuroscience, Conference Abstract: Bernstein Symposium 2008, doi:10.3389/conf.neuro.10.2008.01.060.
[21] V. Heidrich-Meisner, C. Igel, Evolution strategies for direct policy search, in: G. Rudolph (Ed.), Parallel Problem Solving from Nature (PPSN X), in: Lecture Notes in Comput. Sci., vol. 5199, Springer-Verlag, 2008, pp. 428–437.
[22] V. Heidrich-Meisner, C. Igel, Similarities and differences between policy gradient methods and evolution strategies, in: M. Verleysen (Ed.), 16th European Symposium on Artificial Neural Networks (ESANN), D-Side Publications, Evere, Belgium, 2008, pp. 149–154.
[23] V. Heidrich-Meisner, C. Igel, Variable metric reinforcement learning methods applied to the noisy mountain car problem, in: S. Girgin, et al. (Eds.), European Workshop on Reinforcement Learning (EWRL 2008), in: Lecture Notes in Artificial Intelligence, vol. 5323, Springer-Verlag, 2008, pp. 136–150.
[24] C. Igel, Neuroevolution for reinforcement learning using evolution strategies, in: Congress on Evolutionary Computation (CEC 2003), vol. 4, IEEE Press, 2003, pp. 2588–2595.
[25] C. Igel, T. Glasmachers, V. Heidrich-Meisner, Shark, J. Mach. Learn. Res. 9 (2008) 993–996.
[26] C. Igel, W. Erlhagen, D. Jancke, Optimization of dynamic neural field, Neurocomputing 36 (1–4) (2001) 225–233.
[27] J. Jägersküpper, How the $(1 + 1)$ ES using isotropic mutations minimizes positive definite quadratic forms, Theoret. Comput. Sci. 36 (1) (2006) 38–56.
[28] S. Kakade, A natural policy gradient, in: T.G. Dietterich, S. Becker, Z. Ghahramani (Eds.), Advances in Neural Information Processing Systems (NIPS14), MIT Press, 2002.
[29] M. Mandischer, A comparison of evolution strategies and backpropagation for neural network training, Neurocomputing 42 (1–4) (2002) 87–117.
[30] D. Michie, R.A. Chambers, BOXES: An experiment in adaptive control, in: Ella Dale, Donald Michi (Eds.), Machine Intelligence 2, Oliver & Boyd, 1968, pp. 137–152 (chap. 9).

[31] D.E. Moriarty, R. Miikkulainen, Efficient reinforcement learning through symbiotic evolution, Mach. Learn. 22 (1996) 11–32.
[32] D.E. Moriarty, A.C. Schultz, J.J. Grefenstette, Evolutionary algorithms for reinforcement learning, J. Artificial Intelligence Res. 11 (1999) 199–229.
[33] A. Pellecchia, C. Igel, J. Edelbrunner, G. Schöner, Making driver modeling attractive, IEEE Intell. Syst. 20 (2) (2005) 8–12.
[34] J. Peters, S. Schaal, Natural actor-critic, Neurocomputing 71 (7–9) (2008) 1180–1190.
[35] I. Rechenberg, Evolutionsstrategie: Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution, Frommann-Holzboog, 1973.
[36] M. Riedmiller, J. Peters, 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), 2007, pp. 254–261.
[37] J.C. Santamaria, R.S. Sutton, A. Ram, Experiments with reinforcement learning in problems with continuous state and action spaces, Adapt. Behav. 6 (2) (1997) 163.
[38] N. Saravanan, D.B. Fogel, Evolving neural control systems, IEEE Expert 10 (3) (1995) 23–27.
[39] J. Schmidhuber, S. Hochreiter, Guessing can outperform many long time lag algorithms, Technical Report IDSIA-19-96, Istituto Dalle Molle Di Studi Sull Intelligenza Artificiale (IDSIA), 1996.
[40] H.-P. Schwefel, Evolution and Optimum Seeking, Sixth-Generation Comput. Tech. Ser., John Wiley & Sons, 1995.
[41] Nils T. Siebel, Gerald Sommer, Evolutionary reinforcement learning of artificial neural networks, Internat. J. Hybrid Intell. Syst. 4 (3) (2007) 171–183.
[42] P. Stagge, Strukturoptimierung rückgekoppelter neuronaler Netze. Konzepte neuronaler Informationsverarbeitung, ibidem-Verlag, Stuttgart, 2001.
[43] K.O. Stanley, R. Miikkulainen, Evolving neural networks through augmenting topologies, Evol. Comput. 10 (2) (2002) 99–127.
[44] R.S. Sutton, A.G. Barto, Reinforcement Learning: An Introduction, MIT Press, 1998.
[45] R.S. Sutton, D. McAllester, S. Singh, Y. Mansour, Policy gradient methods for reinforcement learning with function approximation, in: Adv. Neural Inf. Process. Syst., vol. 12, MIT Press, 2000, pp. 1057–1063.
[46] T. Suttorp, N. Hansen, C. Igel, Efficient covariance matrix update for variable metric evolution strategies, Mach. Learn. 75 (2) (2009) 167–197.
[47] J. Togelius, F. Gomez, J. Schmidhuber, Learning what to ignore: Memetic climbing in topology and weight space, in: Proceedings of the IEEE World Congress on Computational Intelligence (WCCI 2008), IEEE Press, 2008.
[48] J. Togelius, T. Schaul, J. Schmidhuber, F. Gomez, Countering poisonous inputs with memetic neuroevolution, in: G. Rudolph (Ed.), Parallel Problem Solving from Nature (PPSN X), in: Lecture Notes in Comput. Sci., vol. 5199, Springer-Verlag, 2008, pp. 610–619.
[49] C.J.C.H. Watkins, P. Dayan, Q-learning, Mach. Learn. 8 (1992) 279–292.
[50] S. Whiteson, P. Stone, Evolutionary function approximation for reinforcement learning, J. Mach. Learn. Res. 7 (2006) 877–917.
[51] D. Whitley, S. Dominic, R. Das, C.W. Anderson, Genetic reinforcement learning for neurocontrol problems, Mach. Learn. 13 (2–3) (1993) 259–284.
[52] A. Wieland, Evolving controls for unstable systems, in: Proceedings of the International Joint Conference on Neural Networks, vol. 2, IEEE Press, 1991, pp. 667–673.
[53] D. Wierstra, A. Foerster, J. Peters, J. Schmidhuber, Solving deep memory POMDPs with recurrent policy gradients, in: International Conference on Artificial Neural Networks (ICANN 2007), in: Lecture Notes in Comput. Sci., vol. 4668, Springer-Verlag, 2007, p. 697.
[54] X. Yao, Evolving artificial neural networks, Proc. IEEE 87 (9) (1999) 1423–1447.