Neuroevolution for Reinforcement Learning Using Evolution Strategies

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

Abstract- We apply the CMA-ES, an evolution strategy which efficiently adapts the covariance matrix of the mutation distribution, to the optimization of the weights of neural networks for solving reinforcement learning problems. It turns out that the topology of the networks considerably influences the time to find a suitable control strategy. Still, our results with fixed network topologies are significantly better than those reported for the best evolutionary method so far, which adapts both the weights and the structure of the networks.

1 Introduction

In this paper, we apply an evolution strategy (ES) to the adaptation of the weights of neural networks (NNs) for reinforcement learning (RL) tasks. In supervised learning, gradient-based optimization of NN parameters usually turns out to be considerably faster than sole evolutionary optimization (Stagge, 2001; Igel et al., 2001; Mandischer, 2002), although it might be more prone to getting stuck in local minima. In RL, however, where feedback about the actions of the agent to be optimized is sparse and/or delayed and standard supervised learning methods cannot be applied directly, evolutionary algorithms have proven to be powerful and competitive approaches (Moriarty et al., 1999). The recent success of evolved NNs in game playing (Chellapilla and Fogel, 1999) underlines the potential of the combination of NNs and evolutionary computation for RL and artificial intelligence.

When using an NN for representing directly the policy of an agent, e.g., a control strategy, the NN weights *parameterize* the space of policies the network can realize. This parameterization is usually complex, there are strong correlations, i.e., 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. We propose to use the CMA-ES (Hansen and Ostermeier, 2001) for evolving the weights of NNs for RL tasks. The CMA-ES efficiently adapts the covariance matrix of the mutation distribution and can thereby account for correlations between the parameters.

In the next section, we briefly give some background information about RL and direct search methods, describe the pole balancing benchmark problems used to evaluate our approach, and summarize related work. Then, in section 3, the CMA-ES and the network structures used in our study are introduced. Section 4 is devoted to the experiments and their results. The article ends with a discussion.

2 Evolving Neural Controllers

2.1 Reinforcement Learning and Direct Search

In the standard reinforcement learning (RL) scenario, an agent interacts with its environment at discrete time steps t. It perceives the environment to be in state $s_t \in \mathcal{S}$ and chooses a behavior a_t from the set of actions $\mathcal{A}$ according to its policy $\pi : \mathcal{S} \to \mathcal{A}$. After the execution of action a_t, the environment makes a possibly stochastic transition to a perceived state s_{t+1} and thereby emits a possibly stochastic numerical reward $r_{t+1} \in \mathbb{R}$. The objective of the agent is to adapt its policy such that the expected cumulative future reward is maximized.

There are basically two distinct (model-free) approaches to solve RL problems, namely, methods that search in the space of policies (Moriarty et al., 1999) and methods that search in the space of value functions (Sutton and Barto, 1998). In the latter, which is exemplified by temporal-difference learning algorithms, a state-value function $V : \mathcal{S} \to \mathbb{R}$ or a state-action-value function $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ for judging states or state-action pairs, respectively, is learned. The policy π is then defined on top of this function. When $\mathcal{S}$ or $\mathcal{A}$ is too large or generalization from experiences to new states and actions is desired, function approximators like neural networks (NNs) are used to model Q, V, or π.

"Although not often thought of in this way, genetic algorithms are, in a sense, inherently a reinforcement technique" (Whitley et al., 1993). The potential advantages of direct search methods like evolutionary algorithms compared to standard RL methods are that they (1) allow for direct search in the policy space, whereas most RL methods are restricted to optimizing the policy "indirectly" by adapting state-value or state-action-value functions, (2) are often easier to apply and are more robust with respect to the tuning of the meta-parameters (e.g., we found choosing the right learning rates etc. for temporal-difference learning harder than adjusting the CMA-ES), (3) can be applied if the function approximators are non-differentiable, whereas standard methods involve gradient-based optimization steps, and (4) can also

0-7803-7804-0 /03/$17.00 © 2003 IEEE

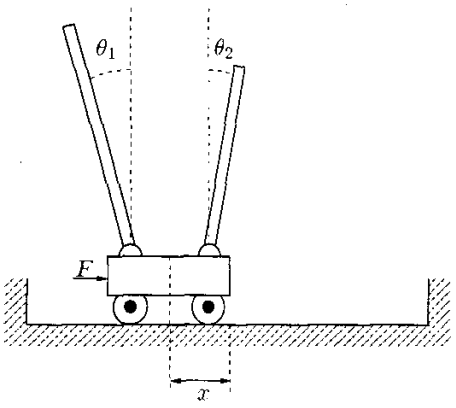

Figure 1: Double pole balancing problem. The parameters x, θ_1, and θ_2 are the offset of the cart from the center of the track and the angles from the vertical of the long and short pole, respectively.

optimize the underlying structure of the function approximators (e.g., the topology of NNs, see Yao, 1999).

2.2 Pole Balancing Problems

Pole balancing problems (also known as inverted pendulum problems) are standard benchmark tasks for the design of controllers for unstable systems, see Wieland (1991) for early and Stanley and Miikkulainen (2002) for recent references. In particular, they have become a standard for the evaluation of evolutionary algorithms that adapt NNs for control.

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. A balancing attempt fails if either the angle from the vertical of any pole exceeds a certain threshold or the cart leaves the track. Control of carts with more than one pole becomes possible when the poles have different lengths. Figure 1 illustrates the task and, for completeness, the corresponding equations of motion are given in appendix A. The problem of designing a controller for the cart can be viewed as a RL task, where the actions are the applied forces and the perceived state corresponds to the information about the system provided to the controller. The only evaluative feedback the controller receives is a negative reinforcement signal when balancing fails.

In this article, we consider three basically 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 θ_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 θ_2 and the angular velocity $\dot{\theta}_2$ of the second pole. Both tasks are Markov decision processes in the sense that the controller perceives the complete information to predict the system, i.e., the whole real system state. The third scenario lacks the Markov property: Again, two poles have to be balanced but this time without velocity information. In this *double pole without velocities* task, the controller gets only x, θ_1, and θ_2 as inputs. In order to distinguish between system states, e.g., whether a pole is moving up or down, the controller needs the capability to exploit history information. This can be achieved by recurrent NNs.

2.3 Previous Work

We give a brief overview of evolutionary methods applied to the optimization of NNs for inverted pendulum problems. In the described studies 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. Usually the search spaces, i.e., the possible NN architectures, differ. This weakens the comparison.

Wieland (1991) considered the evolutionary optimization of recurrent NNs for different pole balancing problems. He adapted the weights of fully recurrent networks by a genetic algorithm with crossover and mutation, where each network parameter was encoded by eight bits and the output was interpreted as a control signal between -10 N and 10 N. We abbreviate this method by NE for conventional neuroevolution (Stanley and Miikkulainen, 2002).

GENITOR for the optimization of NN parameters as described by Whitley et al. (1993) is a steady-state, real-coded genetic algorithm. An offspring is created either by crossover or mutation, where the crossover probability is adaptive. In this article, we refer to the GENITOR experiments conducted by Moriarty and Miikulainen (1996), in which the weights of fully-connected feedforward NNs with five hidden units and shortcut connections were adapted. The single output in the range $[0, 1]$ was interpreted as the parameter of a Bernoulli distribution having two outcomes, full-push left or right. The action was chosen randomly according to this distribution.

When using cellular encoding (CE, Gruau et al., 1996), both the weights and the structure are subject to evolution. Each individual corresponds to an expression of a formal language. These expressions describe growth processes for NNs, i.e., the NNs are encoded *indirectly*. The parse trees of the expressions undergo mutation and crossover in the style of genetic programming. When applied to pole balancing, Gruau et al. (1996) used continuous control signals.

Saravanan and Fogel (1995) applied evolutionary programming (EP) to parameter optimization of feedforward NNs for the double pole balancing task. The networks had 10 hidden units and the control signal was continuous.

In *Symbiotic, Adaptive Neuro-Evolution* (SANE), single neurons (and sometimes additionally "blueprints" for network topologies) are subject to evolution, not complete networks (Moriarty and Miikulainen, 1996). In each generation, a fixed number of hidden neurons, which carry information about the existence and strength of connections to the input (where the bias is interpreted as a connection to an input unit with constant activation, i.e., it can be pruned) or output units, are randomly chosen out of a neuron population to form networks. These NNs are evaluated on the given task. The fitness of an individual neuron is determined based on how well on average the networks that used this neuron performed. When applied to pole-balancing, the evolved networks had two outputs, one for pushing left and one for pushing right. The output with the higher activation determined the action, either a force of -10 N or 10 N. For a comparison of SANE with reinforcement learning approaches on pole balancing tasks the reader is referred to Moriarty and Miikulainen (1996).

The *Enforced Sub-Population* (ESP) approach is based on SANE, but instead of one population of neurons there is a sup-population for each non-input unit of the NN structure created for fitness evaluation. The application of ESP to double pole balancing with and without velocities is described by Gomez and Miikulainen (1999). For the tasks with complete state information, a fixed network topology with 10 hidden units was used, whereas for evolving controllers without velocity information the appropriate number of hidden units was determined by a restart strategy.

NeuroEvolution of Augmenting Topologies (NEAT, Stanley and Miikkulainen, 2002) is a sophisticated evolutionary algorithm that evolves both the structure and the weights of NNs using crossover and mutation. The algorithm keeps track of the history of genetic information ("innovations"). This history record is exploited by a tailored crossover operator. In NEAT, niching (or *speciation*) is achieved by *explicit fitness sharing* in order to protect structural innovations. The algorithm has several parameters (Stanley and Miikkulainen, 2002, section 4.1). However, as it evolves the structure of the NNs starting from small initial topologies, there is no need for expert knowledge for choosing a suitable NN architecture.

Selected results of the studies described above, in particular the average number of balancing attempts (evaluations) needed to find a suitable control strategy, are shown in Tables 1, 2, and 3.

It is important to note that in all of these and our experiments (1) no discretization of the state and action spaces is used (except in the experiments where there are only two actions), as opposed, e.g., to the solutions of pole balancing problems in the seminal papers of Michie and Chambers (1968) and Barto et al. (1983), and (2) the function approximators considered here are not linear in the parameters (weights), because otherwise the evolutionary approach would have to be compared to fast temporal-difference learning methods for linear models, see, e.g., Xu et al. (2002) for an application of least-squares methods to single pole balancing.

method	evaluations	
	centered	random
GENITOR	1846	2578
SANE	535	1691

Table 1: Number of attempts to find an appropriate control strategy for the single pole balancing task starting with all state values zero and from a random initial state, respectively, averaged of 50 trials (taken from Moriarty and Miikulainen, 1996).

3 Neuroevolution Using an Evolution Strategy

In this section, we describe the two ingredients of our approach to solving the inverted pendulum problems: the CMA-ES and the NN architectures.

3.1 CMA Evolution Strategy

We use an elaborated evolution strategy (ES, see Beyer and Schwefel, 2002, for an introduction), namely the $(\mu/\mu, \lambda)$-CMA-ES (Hansen and Ostermeier 1997, 2001), which has proven to perform efficient optimization with small population sizes. Each individual represents an n-dimensional real-valued object variable vector. These variables are altered by recombination and mutation. We use global intermediate recombination, which corresponds to computing the center of mass of the μ individuals in the parent population. Mutation is realized by adding a normally distributed random vector with zero mean, where the complete covariance matrix is adapted during evolution to improve the search strategy. More formally, the object parameters $\boldsymbol{x}_k^{(g+1)}$ of offspring $k = 1, \ldots, \lambda$ created in generation g are given by

$$\boldsymbol{x}_k^{(g+1)} = \langle \boldsymbol{x} \rangle^{(g)} + N_k^{(g)}\left(\mathbf{0}, {\sigma^{(g)}}^2 \boldsymbol{C}^{(g)}\right) ,$$

where $\langle \boldsymbol{x} \rangle^{(g)}$ denotes the center of mass of the population in generation g and the $N_k^{(g)}\left(\mathbf{0}, {\sigma^{(g)}}^2 \boldsymbol{C}^{(g)}\right)$ are independent realizations of an n-dimensional normally distributed random vector with zero mean and covariance matrix ${\sigma^{(g)}}^2 \boldsymbol{C}^{(g)}$. The strategy parameters, both the matrix $\boldsymbol{C}^{(g)}$ and the so called global step-size $\sigma^{(g)}$, are updated online using the covariance matrix adaptation (CMA) method.

method	evaluations	population size
NE (Wieland, 1991)	≈ 307200	2048
EP (Saravanan and Fogel, 1995)	≈ 80000	100
SANE (Moriarty and Miikulainen, 1996)	12600	200
ESP (Gomez and Miikulainen, 1999)	3800	200
NEAT (Stanley and Miikkulainen, 2002)	3600	150

Table 2: Average number of balancing attempts needed for solving the double pole balancing task with velocities. Additionally, the population sizes used in the different neuroevolution methods are given.

method	evaluations	generalization	population size
CE (Gruau et al., 1996)	840000	300	16384
ESP (Gomez and Miikulainen, 1999)	169466	289	1000
NEAT (Stanley and Miikkulainen, 2002)	33184	286	1000

Table 3: Average number of required balancing attempts and population sizes for the double pole balancing task without velocities. The generalization refers to the number of successful balancing attempts starting from 625 different initial positions, see section 4 for details.

Sometimes it may become necessary to enforce a lower bound $\sigma_{\min}$ on the variance of the CMA-ES. In our implementation, we ensure that $\sigma^{(g)} \cdot \lambda_{\min}^{(g)} \geq \sigma_{\min}$, where $\lambda_{\min}^{(g)}$ is the smallest eigenvalue of $\boldsymbol{C}^{(g)}$, i.e., a minimum variance of the shortest principal axis of the mutation ellipsoid can be guaranteed. When not reported otherwise, we set $\sigma^{(0)} = 1$, $\sigma_{\min} = 0$, and $\boldsymbol{C}^{(0)} = \mathrm{diag}(1, \ldots, 1)$.

The CMA implements important concepts for strategy parameter adaptation, in particular *derandomization*: the mutation distribution is altered in a deterministic way such that the probability to reproduce steps in the search space that have led to the actual population is increased. Thereby, the algorithm detects correlations between object variables and becomes invariant under orthogonal transformations of the search space (apart from the initialization). Another important concept is *cumulation*: in order to use the information from previous generations more efficiently, the search path of the population over a number of past generations is taken into account.

In the CMA-ES, rank-based (μ, λ)-selection is used, i.e., the μ best of the λ offspring form the next parent population. The population sizes are chosen according to the heuristic $\lambda = 4 + \lfloor 3 \ln n \rfloor$ and $\mu = \lfloor \lambda/4 \rfloor$ given by Hansen and Ostermeier (2001).

A detailed description of the CMA-ES is beyond the scope of this article, the reader is referred to Hansen and Ostermeier (2001).

3.2 Network Structures

For the Markovian tasks, standard feedforward NNs with a single hidden layer and no shortcut connections were used. As the inverted pendulum problem is somehow symmetric, we tried architectures with and without bias (threshold) parameters. It turned out that the use of bias parameters had a considerably negative effect on the performance, see section 4.2. Hence, there were no bias parameters in most experiments.

Recurrent neural networks (RNNs) were used in the double pole balancing problem without velocity information. In this task, the environment is only partially observable. However, RNNs store history information which can be fused with the actual input signals to represent *belief states* on which the policy can be defined.

We tried different classes of recurrent networks, the best results gave the following simple family of architectures. Let $x_i(t)$ for $0 < i \leq n_{\text{inputs}}$ be the activation of the n_{inputs} input units of a network with n_{neurons} neurons at time step t. 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 $n_{\text{inputs}} < i \leq n_{\text{neurons}}$, where w_{ij} are the weights and $f(a) = a/(|a| + 1)$ is a sigmoidal transfer function.

All NNs have a single output neuron and the input signals provided to the networks are appropriately scaled.

4 Experimental Evaluation

4.1 Experiments

The experiments have been designed in order to be comparable to the results reported by Moriarty and Miikulainen (1996), Gomez and Miikulainen (1999), and Stanley and Miikkulainen (2002). For the equations of the dynamical system that describe the inverted pendulum problems see appendix A.

Single pole. In our single pole experiments, full state information $(x, \dot{x}, \theta_1, \dot{\theta}_1)$ is provided. The goal is to evolve a

controller that can balance the pole for 10^5 time steps starting either from a random position with $-0.2\,\text{rad} < \theta_1 < 0.2\,\text{rad}$, $-2.4\,\text{m} < x < 2.4\,\text{m}$, $-1\,\text{m/s} < \dot{x} < 1\,\text{m/s}$, and $-1.5\,\text{rad/s} < \dot{\theta}_1 < 1.5\,\text{rad/s}$ or with all state variables set to zero (see Moriarty and Miikulainen, 1996). Each discrete time step corresponds to 0.02 s. When the angle exceeds $\pm 12°$ the balancing attempt is regarded as a failure. Networks with 8 hidden neurons and without bias parameters are used. The continuous output is probabilistically mapped to either -10 N or 10 N, see the description of the the GENITOR algorithm in section 2.3.

Double pole with velocities. The double pole experiments are based on the descriptions by Gomez and Miikulainen (1999) and Stanley and Miikkulainen (2002). Full state information $(x, \dot{x}, \theta_1, \dot{\theta}_1, \theta_2, \dot{\theta}_2)$ is provided, the initial state of the long pole is $\theta_1 = 1°$. We varied the number of hidden neurons $n_{\text{hidden}} \in \{4, 6, 8, \ldots, 16\}$ and tested each architecture with and without bias parameters. The angles of the poles have to be in the range $[-36°, 36°]$ for 10^5 time steps, where each step corresponds to 0.02 s.

Double pole without velocities. In this problem, only x, θ_1, and θ_2 are given as inputs, i.e., the system state is only partially observable. A very special fitness function combined with an additional termination criterion defines the goal of the task, which was introduced by Gruau et al. (1996) and was also used by Gomez and Miikulainen (1999) and Stanley and Miikkulainen (2002).

The fitness function is the weighted sum of two components $0.1 f_1 + 0.9 f_2$ defined over 1000 time steps (each time step corresponding to 0.02 s), given by

$$f_1 = t/1000\ ,$$

$$f_2 = \begin{cases} 0 & \text{if } t < 100 \\ \dfrac{0.75}{\sum_{t'=t-99}^{t} (|x| + |\dot{x}| + |\theta_1| + |\dot{\theta}_2|)} & \text{otherwise} \end{cases}$$

Here t denotes the number of steps the controller is able to balance the poles starting from a fixed initial position (all states zero except $\theta_1 = 4.5°$, angles $\theta_1, \theta_2 \in [-36°, 36°]$ are feasible). The first addend rewards successful balancing. The component f_2 penalizes oscillations. The idea behind this second term is to exclude the control strategy to balance the poles by just moving the cart quickly back and forth from the set of solutions (see Gruau et al., 1996; Gomez and Miikulainen, 1999; Stanley and Miikkulainen, 2002).

A trial is stopped and regarded as successful when the fittest individual in the population passes two tests. First, it has to balance the poles from the 4.5° initialization for 10^5 time steps. Second, it has to control the poles for 1000 steps starting from at least 200 out of 625 different initial positions, namely for all $(x, \dot{x}, \theta_1, \dot{\theta}_1, \theta_2, \dot{\theta}_2) \in \{(k_1 \cdot 4.32\,\text{m} - 2.16\,\text{m}, k_2 \cdot 2.70\,\text{m/s} - 1.35\,\text{m/s}, k_3 \cdot 7.2° - 3.6°, k_4 \cdot 17.2°/\text{s} - 8.6°/\text{s}, 0°, 0°/\text{s}) \mid k_1, k_2, k_3, k_4 \in \{0.05, 0.25, 0.5, 0.75, 0.95\}\}$. Note that there is no selection pressure towards evolving a solution that passes this second test apart from f_1 and f_2.

As generally the weights of RNNs are more "sensitive" to changes compared to feedforward architectures, we set the initial global step-size $\sigma^{(0)} = 0.1$ in this task.

4.2 Results

For each scenario, 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. In the following, all statistical test results refer to t-tests.

Single pole. Starting from the upright position, on average 283 (sd = 138) balancing attempts were sufficient to find a control strategy. Starting from random initializations, 967 (sd = 1148) attempts were needed. These results are highly significantly better ($p < 0.001$) than the best ones so far reported by Moriarty and Miikulainen (1996), see Table 1.

Double pole with velocities. The results for the Markovian double pole balancing are given in Table 4. The best results, only 895 evaluations on average, were achieved with NNs having 6 hidden neurons and no biases. This is more than four times better than the best result so far, see Table 2.

It is striking that the architectures without bias parameters clearly outperform those with bias parameters and the same number of hidden neurons. This is not simply due to the increase of the degrees of freedom when adding thresholds, because larger NNs without biases gave better results than smaller ones with biases. A reason for these differences in performance may be the inherent symmetry of the inverted pendulum task, which is broken by bias terms.

On the other hand, the ES approach is quite robust against the choice of the number of hidden neurons n_{hidden}, as long as there are not too few of them. For $n_{\text{hidden}} \in \{8, \ldots, 16\}$ the averages were in the ranges 967–1119 and 2146–2494 for NNs without and with thresholds, respectively. For 4 and 6 hidden neurons, the NNs with thresholds can get stuck in local minima and a few trials fail. This was also observed one time for an architecture with 4 hidden units without bias parameters. We repeated the three sets of experiments where failures occurred with a lower bound $\sigma_{\min} = \sigma^{(0)}/2 = 0.5$ on the variance of the mutation distribution in order to ensure a minimum exploration and thereby prevent the ES from premature "convergence". As shown in Table 4, this led to better results, in particular in all trials a solution was found.

CMA-ES	n_{hidden}	bias	n_{weights}	evaluations	failures
(3/3, 13)	4	no	28	**884**/ 3142	0/1
(3/3, 13)	4	yes	33	2929/25853	0/12
(3/3, 13)	6	no	42	**895**	0
(3/3, 13)	6	yes	49	2672/13464	0/5
(4/4, 16)	8	no	56	1119	0
(4/4, 16)	8	yes	65	2493	0
(4/4, 16)	10	no	70	1003	0
(4/4, 16)	10	yes	81	2494	0
(4/4, 16)	12	no	84	1143	0
(4/4, 16)	12	yes	97	2216	0
(4/4, 16)	14	no	98	1021	0
(4/4, 16)	14	yes	113	2391	0
(4/4, 16)	16	no	112	967	0
(4/4, 16)	16	yes	129	2146	0

Table 4: Results for the double pole balancing task with velocity information. Given are the population sizes, the number n_{hidden} of hidden neurons, whether bias parameters were used or not, the degrees of freedom n_{weights}, the average number of pole balancing attempts to find an appropriate control strategy, and the number of times no such control strategy was found after 10^5 attempts. The results refer to $\sigma_{\min} = 0$; when two values are given the first one refers to $\sigma_{\min} = \sigma^{(0)}/2 = 0.5$ and the second to $\sigma_{\min} = 0$.

In order to judge the statistical significance of our results, we compared the performance of the architectures with 10 hidden nodes *including* bias (mean = 2494, sd = 1515) to the best result reported by Stanley and Miikkulainen (mean = 3600, sd = 2704). Even this moderate CMA-ES result is a significant improvement ($p < 0.05$), although SANE, ESP, and NEAT can in principle switch of the threshold parameters during evolution.

Double pole without velocities. In the evolution of the recurrent networks, our pilot experiments showed a high rate of failures, i.e., trials where no network was found that passed both tests. Closer examination of these trials revealed the reason: In most cases, the CMA-ES found networks that could balance the poles starting from the $\theta_1 = 4.5°$ position for 10^5 time steps, but failed the final generalization test. This is due to a "feature" of the CMA-ES: After the fitness function has been minimized, i.e., the poles can be balanced with few oscillations for 1000 time steps starting from $\theta_1 = 4.5°$, there is no selection pressure towards better generalizing RNNs and therefore the CMA-ES reduces its global step-size in order to stabilize the evolved solution. This is exemplarily shown in Figure 2.

Hence, we ensured ongoing exploration by introducing a minimum variance by setting $\sigma_{\min} = \sigma^{(0)}/2 = 0.05$. The corresponding results for the CMA are shown in Table 5. For 3, 5, and 7 hidden neurons, the average number of balancing attempts needed to find an appropriate control strategy was 6061 (sd = 4025), 8786 (sd = 6150), and 7801 (sd = 5841), respectively. That is, the differences in the number of evaluations between NEAT and the CMA-ES are highly statistically significant ($p < 0.001$). With 3 hidden units without biases, our method is more than five times faster than NEAT. Nevertheless, changes of the topology of the RNNs considerably influence the performance: the RNNs with 3 hidden units are about 25 % faster than the networks with 5 hidden neurons.

For 9 and 11 hidden neurons, some trials still do not find solutions. However, even for $n_{\text{hidden}} = 9$ (sd = 26516, $p < 0.1$) the results are significantly better compared to NEAT; for $n_{\text{hidden}} = 11$ (sd = 31946, $p > 0.1$) the improvement is not significant.

The average number of successful generalization trials is smaller in the CMA experiments compared to the other methods, see Table 3. However, this measure is not directly related to the evolutionary process, because there is no direct selection pressure—apart from the 200 trials threshold that determines the end of a trial—towards solutions that pass many generalization tests.

5 Discussion

Using standard neural network architectures and the CMA evolution strategy for adapting the weights led to better results for solving pole balancing reinforcement tasks than reported so far. The efficient adaptation of the search strategy implemented in the CMA-ES does not only enable faster optimization by detecting correlations between object variables, it also allows for small population sizes.

In the inverted pendulum problems, bias parameters seem to have a negative effect on the adaptation of the neural network, most likely because of the symmetry of the tasks. Further, fine tuning of the weights seems to be not

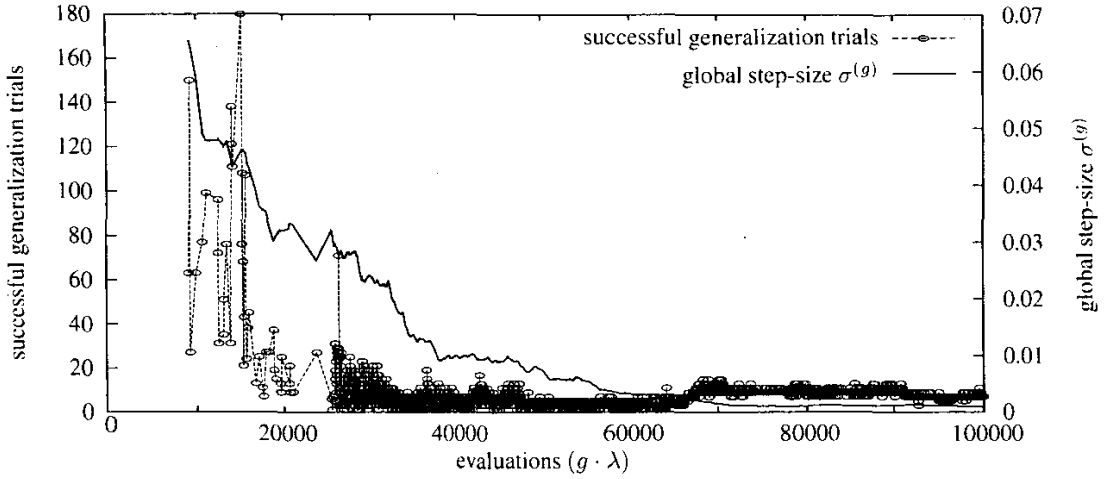

Figure 2: Trajectories of the generalization performance and of the global step-size for a typical unsuccessful trial in the double pole balancing task without velocities and $\sigma_{\min} = 0$. For every generation where the best individual in the population can balance the poles for 10^5 steps starting from the initial $4.5°$ position, the corresponding number of successful generalization trials and the global step-size $\sigma^{(g)}$ are shown. The global step-size decreases over the generations. It takes about 9000 evaluations until a network can balance for 10^5 time steps. Then some kind of random walk starts, where networks with different generalization performance are explored, but the 10^5 steps criterion is not met in all generations. Finally, the algorithm searches only locally around some badly generalizing networks, which do not pass the final test. However, these networks can balance for 10^5 time steps starting from the $4.5°$ position—with respect to the main fitness function, the ES has "stabilized".

CMA-ES	n_{hidden}	n_{weights}	evaluations	generalization	first hit	failures
(3/3, 13)	3	28	6061	250	3521	0
(3/3, 13)	5	54	8786	243	4856	0
(4/4, 16)	7	88	7801	248	5029	0
(4/4, 16)	9	130	21556	227	6597	3
(4/4, 19)	11	180	25254	226	7290	7

Table 5: Results for the double pole balancing task without velocities, generalization refers to the average number of successful balancing attempts of the final NN starting from 625 initial positions and the first hit refers to the average number of balancing attempts needed to find a controller that can balance the poles for 1000 time steps starting from $\theta_1 = 4.5°$.

very important—this may be typical for many reinforcement tasks (e.g., compared to standard regression tasks)—and therefore a lower bound on the variance of the mutation distribution of the CMA-ES appears to have no negative effect. Quite the contrary, in some scenarios—in particular when ongoing exploration is needed in the absence of selection pressure—bounding the variance was necessary to achieve good results.

When evolving the weights of NNs with a fixed number of hidden neurons, our experiments show that sophisticated neuroevolution algorithms like ESP or SANE are not necessary for good results on pole balancing tasks.

Still, the network structure really matters, which becomes obvious from the differences between structures with and without bias parameters and between architectures with different numbers of hidden neurons. Hence, indeed methods are required that evolve both the structure and the weights of neural networks for reinforcement tasks—although for nearly all initializations our approach outperformed the existing algorithms that additionally adapt the topology. However, the standard CMA-ES is limited to a fixed number of object parameters and it is left to future research to find a method that combines (not just trivially by nesting) the power of the CMA for adapting the real valued weights with structure optimization.

Acknowledgments

The implementation of the benchmark problems is directly based on the freely available source code by Richard S. Sutton and Charles W. Anderson for the single pole task and by Faustino Gomez, Kenneth O. Stanley, and Risto Miikulainen for the double pole tasks.

A Equations of Motion

A system with N poles is governed by

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

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

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

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

for $i = 1, \ldots, N$, see Wieland (1991). Here, x is the distance of the cart from the center of the track, F is the force applied to the cart, and $g = 9.8\,\mathrm{m/sec^2}$ is the acceleration due to gravity. The mass and the coefficient of friction of the cart are denoted by m_c and μ_c, respectively. The variables m_i, l_i, θ_i, and μ_i stand for the mass, the half of 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 signum 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, which both depend on the task. In all experiments, $m_c = 1\,\mathrm{kg}$, $m_1 = 0.1\,\mathrm{kg}$, $l_1 = 0.5\,m$, and the width of the track is 4.8 m. In the double pole experiments, $l_2 = 0.1 l_1$, $m_2 = 0.1 m_1$, $\mu_c = 5 \cdot 10^{-4}\,\mathrm{Ns/m}$ and $\mu_1 = \mu_2 = 2 \cdot 10^{-6}\,\mathrm{Nms}$. The dynamical system is numerically solved using fourth-order Runge-Kutta integration with step size $\tau = 0.01\,\mathrm{s}$. In the single pole balancing experiments, the equations of motion are simplified, no friction is considered (e.g., see Whitley et al., 1993) and the dynamical system is numerically solved using Euler's method with time step $\tau = 0.02\,\mathrm{s}$.

Bibliography

Barto, A. G., R. S. Sutton, and C. W. Anderson (1983). Neuronlike elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics 13*(5), 835–846.

Beyer, H.-G. and H.-P. Schwefel (2002). Evolution strategies: A comprehensive introduction. *Natural Computing 1*(1), 3–52.

Chellapilla, K. and D. B. Fogel (1999). Evolution, neural networks, games, and intelligence. *Proceedings of the IEEE 87*(9), 1471–1496.

Gomez, F. J. and R. Miikulainen (1999). Solving non-markovian tasks with neuroevolution. In T. Dean (Ed.), *Proceeding of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI)*, Stockholm, Sweden, pp. 1356–1361. Morgan Kaufmann.

Gruau, F., D. Whitley, and L. Pyeatt (1996). A comparison between cellular encoding and direct encoding for genetic neural networks. In J. R. Koza, D. E. Goldberg, D. B. Fogel, and R. L. Riolo (Eds.), *Genetic Programming 1996: Proceedings of the First Annual Conference*, Stanford University, CA, USA, pp. 81–89. MIT Press.

Hansen, N. and A. Ostermeier (1997). Convergence properties of evolution strategies with the derandomized covariance matrix adaptation: The $(\mu/\mu, \lambda)$-CMA-ES. In *5th European Congress on Intelligent Techniques and Soft Computing (EUFIT'97)*, pp. 650–654. Aachen, Germany: Verlag Mainz, Wissenschaftsverlag.

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

Igel, C., W. Erlhagen, and D. Jancke (2001). Optimization of dynamic neural fields. *Neurocomputing 36*(1-4), 225–233.

Mandischer, M. (2002). A comparison of evolution strategies and backpropagation for neural network training. *Neurocomputing 42*(1–4), 87–117.

Michie, D. and R. A. Chambers (1968). BOXES: An experiment in adaptive control. In E. Dale and D. Michie (Eds.), *Machine Intelligence 2*, Chapter 9, pp. 137–152. Edinburgh, UK: Oliver & Boyd.

Moriarty, D. E. and R. Miikulainen (1996). Efficient reinforcement learning through symbiotic evolution. *Machine Learning 22*, 11–32.

Moriarty, D. E., A. C. Schultz, and J. J. Grefenstette (1999). Evolutionary Algorithms for Reinforcement Learning. *Journal of Artificial Intelligence Research 11*, 199–229.

Saravanan, N. and D. B. Fogel (1995). Evolving neural control systems. *IEEE Expert 10*(3), 23–27.

Stagge, P. (2001). *Strukturoptimierung rückgekoppelter neuronaler Netze*. Konzepte neuronaler Informationsverarbeitung. Stuttgart: ibidem-Verlag.

Stanley, K. O. and R. Miikkulainen (2002). Evolving neural networks through augmenting topologies. *Evolutionary Computation 10*(2), 99–127.

Sutton, R. S. and A. G. Barto (1998). *Reinforcement Learning: An Introduction*. MIT Press.

Whitley, D., S. Dominic, R. Das, and C. W. Anderson (1993). Genetic reinforcement learning for neurocontrol problems. *Machine Learning 13*(2–3), 259–284.

Wieland, A. (1991). Evolving controls for unstable systems. In *Proceedings of the International Joint Conference on Neural Networks*, Volume II, Seattle, WA, USA, pp. 667–673. IEEE Press.

Xu, X., H. He, and D. Hu (2002). Efficient reinforcement learning using recursive least-squares methods. *Journal of Artificial Intelligence Research 16*, 259–292.

Yao, X. (1999). Evolving artificial neural networks. *Proceedings of the IEEE 87*(9), 1423–1447.