Block Diagonal Natural Evolution Strategies

Giuseppe Cuccu and Faustino Gomez

IDSIA
USI-SUPSI
6928 Manno-Lugano, Switzerland
{giuse,tino}@idsia.ch
http://www.idsia.ch/~giuse,~tino

Abstract. The *Natural Evolution Strategies* (NES) family of search algorithms have been shown to be efficient black-box optimizers, but the most powerful version xNES does not scale to problems with more than a few hundred dimensions. And the scalable variant, SNES, potentially ignores important correlations between parameters. This paper introduces Block Diagonal NES (BD-NES), a variant of NES which uses a block diagonal covariance matrix. The resulting update equations are computationally effective on problems with much higher dimensionality than their full-covariance counterparts, while retaining faster convergence speed than methods that ignore covariance information altogether. The algorithm has been tested on the *Octopus-arm benchmark*, and the experiments section presents performance statistics showing that BD-NES achieves better performance than SNES on networks that are too large to be optimized by xNES.

1 Introduction

Natural Evolution Strategies (NES; [11]) have been shown to efficiently optimize neural network controllers for reinforcement learning tasks [2; 7; 10]. This family of algorithms searches the space of network weights by adapting a parameterized distribution (usually Gaussian) in order to optimize expected fitness by means of the natural gradient. The two main variants of NES, xNES [3] and SNES [7], make a trade-off between generality and efficiency: xNES (like CMA-ES [5]) uses a full covariance matrix, capturing all possible correlations between the weights but at a cost of $\mathcal{O}(w^3)$, where w is the number of weights. Unfortunately, xNES does not scale to the space of even modest size neural networks, with hundreds of weights. At the other extreme, SNES ignores weight correlations altogether in exchange for $\mathcal{O}(w)$ complexity, by using a diagonal covariance matrix. Even though it cannot solve non-separable problems, it seems to work well for neuroevolution, arguably because of the high number of possible solutions for any given network structure.

SNES updates its search distribution two orders of magnitude faster than xNES, but, by not taking into account epistatic linkages between network weights

C.A. Coello Coello et al. (Eds.): PPSN 2012, Part II, LNCS 7492, pp. 488–497, 2012.
© Springer-Verlag Berlin Heidelberg 2012

(e.g. arising from correlated inputs), does not make full use of strong regularity inherent in many control problems. For example, sensors positioned near each other on a robot body will likely generate correlated readings, and therefore the corresponding weights processing sensory information will probably be correlated as well.

In this paper, we introduce a new NES variant that is intermediate between SNES and xNES in that it allows for correlations between subsets of search dimensions (e.g. weights), by using a search distribution with a block-diagonal covariance matrix. By allowing correlations only between some weights, the computational complexity can be reduced significantly vis-a-vis xNES, but this requires first identifying which weights should be grouped together. In a general, unconstrained optimization setting such properties of the objective (fitness) function are not known *a priori*. However, in neuroevolution, the phenotypic structure provides a natural way to decompose the search space by grouping together those weights which belong to the same neuron (i.e. network sub-function).

This Block Diagonal NES (BD-NES) uses one full covariance matrix for each neuron, allowing correlations between all weights of a given neuron, but ignoring correlation between weights of different neurons. This approach is similar to *cooperative coevolution* [4; 6], where each neuron is represented by a separate sub-genotype, and the complete individual is constructed by concatenating the sub-genotypes.

The next section derives the new algorithm from the NES family. Section 3, presents comparative results against SNES. Section 4, discusses the results and provides some ideas on how to further improve this approach.

2 Block Diagonal Natural Evolution Strategies

BD-NES can be viewed as multiple xNES [3] algorithms running in parallel, one for each block in the covariance matrix of the search distribution. Of course, the blocks can be of different size if the relationship between problem dimensions is known in advance (i.e. whether any two dimension are separable). Here, in the context of neuroevolution and in the absence of this kind of knowledge, the division of the network weights into blocks is determined by the number of neurons in the network architecture, Ψ.

Figure 1 describes the block-diagonal covariance matrix used by the search distribution. Each neuron, i has its own block, Σ_i, that captures all of the covariances between its incoming connections. Algorithm 1 presents the code for BD-NES. First, the mean vectors, $\boldsymbol{\mu}_i \in \mathbb{R}^c$, and $c \times c$ covariance matrices, $\boldsymbol{\Sigma}_i$, $i = 1..n$, are initialized, where n is the number of neurons, and c is the number of incoming connections per neuron. Each generation (the `while` loop), λ networks are constructed by sampling from each Gaussian sub-distribution to obtain ψ neuron chromosomes, $\mathbf{z}^i$, $i = 1..\psi$, (line 5) which are then concatenated into a complete genome, $\mathbf{z}$, (line 7). The genomes are then transformed into networks, and evaluated. The fitness achieved by a networks is passed to its constituent

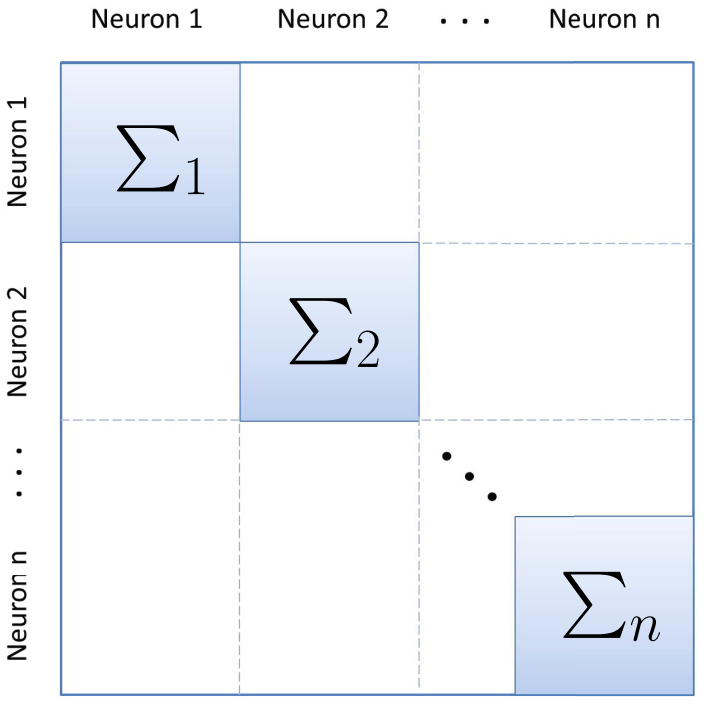


Fig. 1. Block Diagonal covariance matrix: The search distribution has a separate block in its covariance matrix for the each neuron (i.e. the covariance between neurons is zero) in the network architecture being evolved. The block size for a given neuron is the number of connections entering that neuron. To evaluate the gradient from the distribution, samples are drawn from the blocks, and then concatenated to construct the full genotype.

neuron chromosomes (line 10) and used to update the corresponding mean, and dedicated covariance block using xNES (line 14), described next.

Let $p(\mathsf{z}\,|\,\theta)$ denote the density of the Gaussian with parameters $\theta = (\boldsymbol{\mu}, \boldsymbol{\Sigma})$. Then, the expected fitness under the search distribution is

$$J(\theta) = \mathbb{E}_\theta[f(\mathsf{z})] = \int f(\mathsf{z})\; p(\mathsf{z}\,|\,\theta)\; d\mathsf{z} \;\; .$$

The gradient w.r.t. the parameters can be rewritten as

$$\begin{aligned} \nabla_\theta J(\theta) &= \nabla_\theta \int f(\mathsf{z})\; p(\mathsf{z}\,|\,\theta)\; d\mathsf{z} \\ &= \mathbb{E}_\theta \left[f(\mathsf{z})\; \nabla_\theta \log \left(p(\mathsf{z}\,|\,\theta) \right) \right] \;\; , \end{aligned}$$

(see [11] for the full derivation) from which we obtain the Monte Carlo estimate

$$\nabla_\theta J(\theta) \approx \frac{1}{\lambda} \sum_{k=1}^{\lambda} f(\mathsf{z}_k)\, \nabla_\theta \log\left(p(\mathsf{z}_k \,|\, \theta)\right) \tag{1}$$

of the search gradient. The key step then consists of replacing this gradient, pointing into the direction of (locally) steepest descent w.r.t. the given parameterization, by the natural gradient

$$\widetilde{\nabla}_\theta J = \mathbf{F}^{-1} \nabla_\theta J(\theta) \ ,$$

where $\mathbf{F} = \mathbb{E}\left[\nabla_\theta \log\left(p\left(\mathsf{z}|\theta\right)\right) \nabla_\theta \log\left(p\left(\mathsf{z}|\theta\right)\right)^\top\right]$ is the Fisher information matrix; leading to a straightforward scheme of natural gradient descent for iteratively updating the search distribution

$$\theta \leftarrow \theta - \eta \widetilde{\nabla}_\theta J = \theta - \eta \mathbf{F}^{-1} \nabla_\theta J(\theta) \ ,$$

with learning rate parameter η. The sequence of (1) sampling an offspring population, (2) computing the corresponding Monte Carlo estimate of the fitness gradient, (3) transforming it into the natural gradient, and (4) updating the search distribution, constitutes one generation of NES.

In order to render the algorithm invariant under monotonic (rank preserving) transformations of the fitness values, *fitness shaping* [11] is used to normalize the fitness into rank-based *utilities* $u_k \in \mathbb{R}$, $k \in \{1, \ldots, \lambda\}$. The individuals are ordered by fitness, with $\mathsf{z}_{1:\lambda}$ and $\mathsf{z}_{\lambda:\lambda}$ denoting the most and least fit offspring, respectively. The distribution parameters are then updated using the "fitness-shaped" gradient:

$$\nabla_\theta J = \sum_{k=1}^{\lambda} u_k \cdot \nabla_{(\theta)} \log\left(p(\mathsf{z}_{k:\lambda} \,|\, \theta)\right) \ . \tag{2}$$

Typically, the utility values are either non-negative numbers that sum to one, or a shifted variant with zero mean.

Using the same exponential local coordinates as in [3], the update equations for the sub-distributions are:

$$\boldsymbol{\mu}^i_{new} \leftarrow \boldsymbol{\mu}^i + \eta_\mu \cdot \sum_{k=1}^{\lambda} u_k \cdot \mathsf{z}^i_k$$

$$A^i_{new} \leftarrow A^i \cdot \exp\left(\frac{\eta_A}{2} \cdot \sum_{k=1}^{\lambda} u_k \cdot \left(\mathsf{z}^i_k \mathsf{z}^{i^\top}_k - I\right)\right)$$

where A^i is the upper triangular matrix resulting from the Cholesky decomposition of covariance block Σ^i, $\Sigma^i = A^{i\top} A^i$.

This approach assumes weights of different neurons not to be correlated, but given the high number of feasible solutions in continuous control problems such constraint does not usually limit the search.

Algorithm 1. BD-NES(Ψ)

```
Initialize (μ_1, Σ_1) ... (μ_n, Σ_n)
while not solved do
    for k ← 1 to λ do
        for i ← 1 to ψ do
            z_k^i ~ N(μ_i, Σ_i)
        end
        z_k ← Concatenate(z_k^1 ... z_k^ψ)
        fit_k ← Evaluate(z_k)
        for j ← 1 to ψ do
            fit_k^i ← fit_k
        end
    end
    for i ← 1 to ψ do
        (μ_i, Σ_i) ← UpdateXNES(μ_i, Σ_i, (z_1^i ... z_λ^i))
    end
end
```

Computational Complexity

SNES and xNES can be thought of as special cases of BD-NES. Let P be a partition of the weights consisting of b blocks of the same size s, and w be the total number of weights. SNES considers all weights to be uncorrelated, so $s = 1$, and $b = w$, whereas in xNES all of the weights are considered to be correlated: $b = 1$ and $s = w$, producing the full covariance matrix.

The dominant operation in the NES update step in terms of computational complexity is the covariance matrix inversion. The computational cost under this framework is proportional to the cost of inverting each matrix block, times the number of blocks being inverted, $\mathcal{O}(bs^3)$. For BD-NES, *with neurons defining the covariance blocks*, $b = \psi$ and $s = c$, where ψ is the number of neurons in the network, and c is the number of connections per neuron (i.e. the node degree).

For single layer feed-forward networks, c depends only on the number of input/output units specified by the problem domain, and not on the number of neurons, so the complexity becomes $\mathcal{O}(\psi)$; the same as SNES, with an hidden constant depending on the number of input units in the network.

For fully-connected recurrent neural networks, c grows with the number of neurons $c \sim \psi$ (each additional neuron adds a connection to every other neuron), thus the complexity becomes $\mathcal{O}(\psi \times \psi^3) = \mathcal{O}(\psi^4)$, which is between SNES and xNES, as in such networks $w \sim \psi^2$, making the complexity of SNES $\mathcal{O}(\psi^2)$ and that of xNES $\mathcal{O}(\psi^6)$. The complexity improves further if we assume that for large networks the connectivity is sparse, with neurons having a fixed average number of recurrent connections, k, as is the case in real-world complex networks, which exhibit the small world property [1]. In this case, BD-NES reduces to $\mathcal{O}(\psi)$, since $c = k$ is constant.

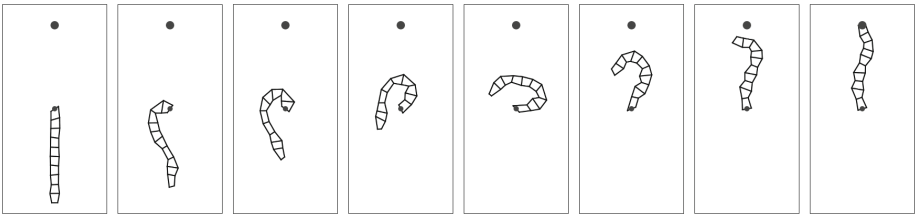


Fig. 2. Octopus-Arm Acrobot Task. A flexible arm consisting of n compartments, each with 3 controllable muscles, must be lifted from its initial downward-pointing position (left), against gravity, to touch a goal location (black dot) with its tip. The behavior shown was evolved through BD-NES.

3 Experiments

BD-NES was tested on a version of the octopus arm control benchmark. This environment was chosen because it requires networks with thousands of weights and therefore cannot be solved using modern evolutionary strategies like xNES and CMA-ES that use a full covariance matrix for the search distribution.

3.1 Octopus-Arm Acrobot Task

The Octopus Arm [12; 13] (see figure 2) consists of p compartments floating in a 2D water environment. Each compartment has a constant volume and contains three controllable muscles (dorsal, transverse and ventral). The state of a compartment is described by the x, y-coordinates of two of its corners plus their corresponding x and y velocities. Together with the arm base rotation, the arm has $8p + 2$ state variables and $3p + 2$ control variables.

In the standard setup, the goal of the task is to reach a target position with the tip of the arm, starting from three different initial positions, by contracting the appropriate muscles at each 1sec step of simulated time. It turns out that it is very easy to get close to the target from two of the initial positions. Therefore, we devised a version, shown in figure 2, where the arm initially hangs down (in stable equilibrium due to gravity), and must be lifted to touch the target above, on the opposite side of the environment, with its tip. The task is termed the *octopus arm acrobot* due to its similarity with the classic acrobot swing-up task [9].

Also, instead of the standard 8 "meta"–actions that simplify control by contracting groups of muscles simultaneously (e.g. all dorsal, all ventral, etc.), the controllers must instead contract each individual muscle independently.

3.2 Network Architecture

Networks were evolved to control a $n = 10$ compartment arm using fully-connected recurrent neural networks having 32 neurons, one for each muscle (see figure 3).

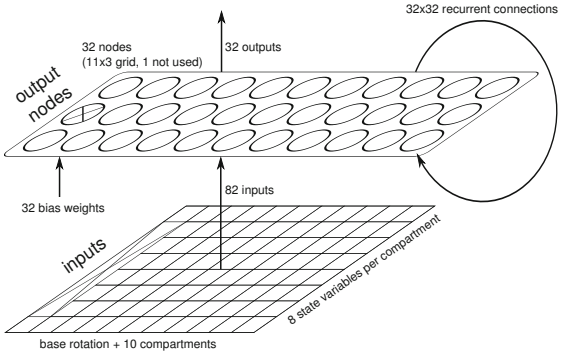


Fig. 3. Network architecture. The octopus arm is controlled by a single layer recurrent neural network with 82 inputs and 32 neurons (outputs), one for each muscle in the arm.

The networks have a 32×82 input weight matrix, a 32×32 recurrent weight matrix and bias vector of length 32, for a total of $3,680$ weights.

The size of a full covariance matrix to search in $3,680$ dimensions is $3,680^2 = 13,542,400$ entries. Yet each of the 32 neurons has only $82+32+1 = 115$ incoming connections, so the covariance blocks in BD-NES that have to be inverted have only $115^2 = 13,225$ entries, three orders of magnitude fewer than for the full covariance matrix.

3.3 Setup

BD-NES was compared with SNES. To provide a baseline, random weight guessing (RWG; [8]) was also used, where the network weights are chosen at random (i.d.d.) from a uniform distribution. This approach gives us an idea of how difficult the task is to solve by simply guessing a good set of weights.

The population size λ is proportional to the number of weights, w, being evolved; set here to $\lambda = 50$. The learning rates are $\eta_{\boldsymbol{\mu}} = \frac{\log(w)+3}{5\sqrt{w}}$ and $\eta_{\boldsymbol{\sigma}} = \frac{\eta_{\boldsymbol{\mu}}}{2}$. Each run was limited to $10,000$ fitness evaluations. The fitness was computed as:

$$\left[1 - \frac{t}{T}\frac{d}{D}, 0\right], \tag{3}$$

where t is the number of time steps before the arm touches the goal, T (set to 100) is the maximum number of time steps in a trial, d is the final distance of the arm tip to the goal, and D is the initial distance of the arm tip to the goal. This fitness measure is different to the one used in [12], because minimizing the integrated distance of the arm tip to the goal causes greedy behaviors. In the viscous fluid environment of the octopus arm, a greedy strategy using the shortest length trajectory does not lead to the fastest movement: the arm needs to be contracted first, then rotated, and finally stretched upwards towards the

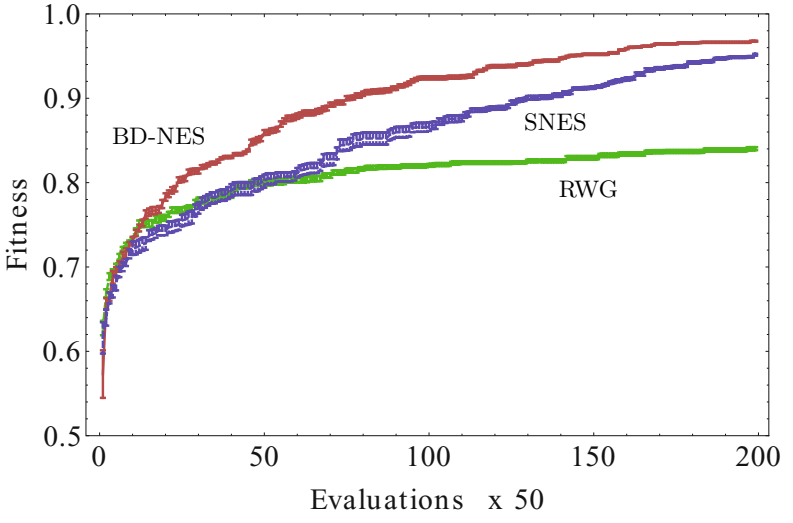


Fig. 4. Performance on octopus-arm acrobot. BD-NES (top, blue) and SNES (bottom, red) performance on the octopus arm benchmark. Curves are averages over 20 runs, with error bars showing the corresponding variance.

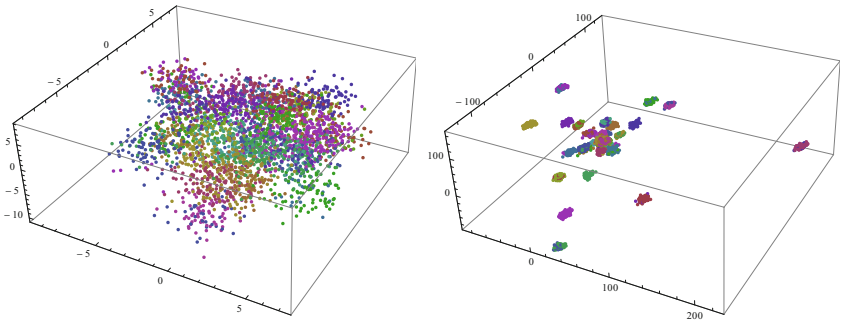
goal. The fitness function favors behaviors that reach the goal within a small number of time steps.

3.4 Results

Figure 4 shows the fitness of the best network found so far for the three methods, averaged over 20 runs (bars indicate variance). BD-NES reaches a fitness equal to the final fitness of SNES at around 7000 evaluations (30% fewer evaluations), and does so in 15% less cpu-time[1] (55.4min for BD-NES, 65.4min for SNES).

Figure 5 shows the neuron chromosomes at the end of a typical run of (a) SNES and (b) BD-NES, projected into 3D via principal component analysis. Each point denotes a neuron sampled from one of final sub-distributions. In all SNES runs the neuron distributions overlap (note scale), suggesting that the neurons are functionally more similar than the neurons comprising a network BD-NES, where neuron clusters are more distinct. While similarity between neurons is to be expected given the similar function that muscles in adjacent compartments must perform, different parts of the arm must perform slightly different tasks (e.g. the muscles controlling the rotation at the base), so that the specialization occurring in BD-NES could explain the better performance.

[1] Reference machine: intel i7 640M at 3.33GHz and 4GB of ram DDR3 at 1066MHz. Mathematica implementation of search algorithm using the Java implementation of the octopus arm available at: `http://www.cs.mcgill.ca/~idprecup/workshops/ICML06/octopus.html`.



(a) (b)

Fig. 5. Neuron specialization. The plots show the 115-dimensional neuron chromosomes in the final population of a typical run, projected into 3D by PCA, for (a) SNES and (b) BD-NES. For SNES, The neuron clusters overlap and are concentrated in a small region of the search space. For BD-NES, neuron distributions form distinct clusters, that are more spread out.

4 Discussion

BD-NES is a novel algorithm of the NES family allowing for partial correlation information to be retained while limiting the computational overhead. The experiments show that block diagonal covariance matrix adaptation can scale up to over 3000 dimensions, and search more efficiently than its diagonal-covariance counterpart, SNES.

For problems where the number of inputs is very large (e.g. video input), decomposing the network at the level of neurons will not work. In this case, neurons can use receptive fields that receive only part of the full input, as is done in convolutional networks. Or, blocks can be built based on inputs rather than neurons, each block representing the covariance matrix for the weights of all connections from a particular input to all neurons. Future work will start by applying the method to a vision version of the task used here, where the network does not receive the state of the arm, but instead sees the arm configuration from a 3rd-person perspective, and must solve the task using high-dimensional images as input.

Acknowledgments. This research was supported by Swiss National Science Foundation grant #137736: "Advanced Cooperative NeuroEvolution for Autonomous Control". Thanks to Jan Koutník for inspiration and support.

References

[1] Albert, R., Barabási, A.-L.: Statistical mechanics of complex networks. Reviews of Modern Physics 74 (January 2002)

[2] Cuccu, G., Luciw, M., Schmidhuber, J., Gomez, F.: Intrinsically motivated evolutionary search for vision-based reinforcement learning. In: Proceedings of the 2011 IEEE Conference on Development and Learning and Epigenetic Robotics IEEE-ICDL-EPIROB. IEEE (2011)

[3] Glasmachers, T., Schaul, T., Sun, Y., Wierstra, D., Schmidhuber, J.: Exponential natural evolution strategies. In: Genetic and Evolutionary Computation Conference, GECCO, Portland, OR (2010)

[4] Gomez, F., Miikkulainen, R.: Incremental evolution of complex general behavior. Adaptive Behavior 5(3-4), 317 (1997)

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

[6] Potter, M.A., De Jong, K.A.: Evolving neural networks with collaborative species. In: Proceedings of the 1995 Summer Computer Simulation Conference (1995)

[7] Schaul, T., Glasmachers, T., Schmidhuber, J.: High dimensions and heavy tails for natural evolution strategies. In: Genetic and Evolutionary Computation Conference, GECCO (2011)

[8] Schmidhuber, J., Hochreiter, S., Bengio, Y.: Evaluating benchmark problems by random guessing. In: Kremer, S.C., Kolen, J.F. (eds.) A Field Guide to Dynamical Recurrent Neural Networks. IEEE Press (2001)

[9] Spong, M.W.: Swing up control of the acrobot. In: Proceedings of the 1994 IEEE Conference on Robotics and Automation, San Diego, CA, vol. 46, pp. 2356–2361 (1994)

[10] Sun, Y., Wierstra, D., Schaul, T., Schmidhuber, J.: Stochastic search using the natural gradient. In: International Conference on Machine Learning, ICML (2009)

[11] Wierstra, D., Schaul, T., Peters, J., Schmidhuber, J.: Natural evolution strategies. In: Proceedings of the Congress on Evolutionary Computation, CEC 2008, Hongkong. IEEE Press (2008)

[12] Woolley, B.G., Stanley, K.O.: Evolving a Single Scalable Controller for an Octopus Arm with a Variable Number of Segments. In: Schaefer, R., Cotta, C., Kołodziej, J., Rudolph, G. (eds.) PPSN XI. LNCS, vol. 6239, pp. 270–279. Springer, Heidelberg (2010)

[13] Yekutieli, Y., Sagiv-Zohar, R., Aharonov, R., Engel, Y., Hochner, B., Flash, T.: A dynamic model of the octopus arm. I. Biomechanics of the octopus reaching movement. Journal of Neurophysiology 94(2), 1443–1458 (2005)