Trends and Applications in Constructive Approximation
(Eds.) M.G. de **Bruin**, D.H. **Mache** & J. **Szabados**
International Series of Numerical Mathematics Vol. 1??
©2005 Birkhäuser Verlag Basel (ISBN 3-7643-7124-2)

Evolutionary Optimization of Neural Systems: The Use of Strategy Adaptation

Christian Igel Stefan Wiegand Frauke Friedrichs

Abstract

We consider the synthesis of neural networks by evolutionary algorithms, which are randomized direct optimization methods inspired by neo-Darwinian evolution theory. Evolutionary algorithms in general as well as special variants for real-valued optimization and for search in the space of graphs are introduced. We put an emphasis on strategy adaptation, a feature of evolutionary methods that allows for the control of the search strategy during the optimization process.

Three recent applications of evolutionary optimization of neural systems are presented: topology optimization of multi-layer neural networks for face detection, weight optimization of recurrent networks for solving reinforcement learning tasks, and hyperparameter tuning of support vector machines.

1 Introduction

The information processing capabilities of vertebrate brains outperform technical systems in many respects. Abstract models of neural networks (NNs) exist, which can—in principle—simulate all Turing machines and exhibit universal approximation properties (e.g., [40, 42, 44]). However, the general question of how to efficiently design an appropriate neural system for a given problem remains open and complexity theory reveals the need for using heuristics (e.g., [41]). The answer is likely to be found by investigating the three major organization principles of biological NNs: evolution, self-organization, and learning.

In the following, we consider the synthesis of NNs by evolutionary algorithms (EAs), which are randomized direct optimization methods inspired by neo-Darwinian evolution theory. We focus on strategy adaptation, a feature of evolutionary methods that enables control of the search strategy during the optimization process. Two related ways of adapting search strategies in evolutionary computation are presented. First, we describe the CMA evolution strategy [19], an efficient method for adjusting the covariance matrix of the search distribution in real-valued optimization. Second, we introduce an algorithm that adapts the probabilities of variation operators [24]. We sketch applications of these methods to the design of different types of neural systems: topology optimization of multi-layer NNs for face detection [49], weight optimization of recurrent networks for solving reinforcement learning tasks [21], and model selection for support vector machines [14].

An introduction to NNs is beyond the scope of this article, the reader is referred to the standard literature, for example the collection [1]. The articles [12, 4] provide starting points for reading about the theory of evolutionary computation. General surveys of evolutionary optimization of neural networks can be found in [29, 30, 39, 52].

2 Evolutionary Computation

Evolutionary algorithms (EAs) can be considered as a special class of global random search algorithms. Let the search problem under consideration be described by a quality function $f : \mathcal{G} \to \mathcal{F}$ to be optimized, where $\mathcal{G}$ denotes the search space (i.e., the space of candidate solutions) and $\mathcal{F}$ the (at least partially) ordered space of cost values. The general global random search scheme can be described—with slight modifications—as follows [54, 25]:

① Choose a joint probability distribution $P^{(t)}_{\mathcal{G}^\lambda}$ on $\mathcal{G}^\lambda$. Set $t \leftarrow 1$.

② Obtain λ points $\boldsymbol{g}^{(t)}_1, \dots, \boldsymbol{g}^{(t)}_\lambda$ by sampling from the distribution $P^{(t)}_{\mathcal{G}^\lambda}$. Evaluate these points using f.

③ According to a fixed (algorithm dependent) rule construct a new probability distribution $P^{(t+1)}_{\mathcal{G}^\lambda}$ on $\mathcal{G}^\lambda$.

④ Check for some appropriate stopping condition; if the algorithm has not terminated, substitute $t \leftarrow t + 1$ and return to step ②.

Random search algorithms can differ fundamentally in the way they describe (parameterize) and alter the joint distribution $P^{(t)}_{\mathcal{G}^\lambda}$, which is typically represented by a semi-parametric model.

The scheme of a canonical EA is shown in figure 1. In evolutionary computation, the iterations of the algorithm are called *generations*. The search distribution of an EA is given by the *parent population*, the *variation operators*, and the *strategy parameters*.

The parent population is a multiset of μ points $\tilde{\boldsymbol{g}}^{(t)}_1, \dots, \tilde{\boldsymbol{g}}^{(t)}_\mu \in \mathcal{G}$. Each point corresponds to the *genotype* of an *individual*. In each generation, λ *offspring* $\boldsymbol{g}^{(t)}_1, \dots, \boldsymbol{g}^{(t)}_\lambda \in \mathcal{G}$ are created by the following procedure: Individuals for reproduction are chosen from $\tilde{\boldsymbol{g}}_1^{(t)}, \ldots, \tilde{\boldsymbol{g}}_\mu^{(t)}$. This is called *mating selection* and can be deterministic or stocastic (where the sampling can be with or without replacement). The offspring's genotypes result from applying variation operators to these selected parents. Variation operators are deterministic or partially stochastic mappings from $\mathcal{G}^k$ to $\mathcal{G}^l$, $1 \leq k \leq \mu, 1 \leq l \leq \lambda$. An operator with $k = l = 1$ is called *mutation*, whereas *recombination* operators involve more than one parent and can lead to more than one offspring. Multiple operators can be applied consecutively to generate offspring. For example, an offspring $\boldsymbol{g}_i^{(t)}$ can be the product of applying recombination $o_{\text{rec}} : \mathcal{G}^2 \to \mathcal{G}$ to two randomly selected parents $\tilde{\boldsymbol{g}}_{i_1}^{(t)}$ and $\tilde{\boldsymbol{g}}_{i_2}^{(t)}$ followed by mutation $o_{\text{mut}} : \mathcal{G} \to \mathcal{G}$, that is, $\boldsymbol{g}_i^{(t)} = o_{\text{mut}}\left(o_{\text{rec}}\left(\tilde{\boldsymbol{g}}_{i_1}^{(t)}, \tilde{\boldsymbol{g}}_{i_2}^{(t)}\right)\right)$. Evolutionary algorithms allow for incorporation of *a priori* knowledge about the problem by using tailored variation operators combined with an appropriate encoding of the candidate solutions.

Let the probability that parents $\tilde{\boldsymbol{g}}_1^{(t)}, \ldots, \tilde{\boldsymbol{g}}_\mu^{(t)}$ lead to offspring $\boldsymbol{g}_1^{(t)}, \ldots, \boldsymbol{g}_\lambda^{(t)}$ be described by the conditional probability distribution

$$P_{\mathcal{G}^\lambda}\left(\boldsymbol{g}_1, \ldots, \boldsymbol{g}_\lambda \,|\, \tilde{\boldsymbol{g}}_1^{(t)}, \ldots, \tilde{\boldsymbol{g}}_\mu^{(t)}; \boldsymbol{\theta}^{(t)}\right) = P_{\mathcal{G}^\lambda}^{(t)}(\boldsymbol{g}_1, \ldots, \boldsymbol{g}_\lambda) \ . \tag{1}$$

This distribution is additionally parameterized by some *external strategy parameters* $\boldsymbol{\theta}^{(t)} \in \Theta$, which may vary over time. In some EAs, the offspring are created independently of each other based on the same distribution. In this case, the joint distribution $P_{\mathcal{G}^\lambda}^{(t)}$ can be factorized as

$$P_{\mathcal{G}^\lambda}^{(t)}(\boldsymbol{g}_1, \ldots, \boldsymbol{g}_\lambda) = P_{\mathcal{G}}^{(t)}(\boldsymbol{g}_1) \cdot \ldots \cdot P_{\mathcal{G}}^{(t)}(\boldsymbol{g}_\lambda) \ . \tag{2}$$

Evaluation of an individual corresponds to determining its fitness by assigning the corresponding cost value given by the quality function f. Evolutionary algorithms can handle optimization problems that are non-differentiable, non-continuous, multi-modal, and noisy. They are easy to parallelize by distributing the fitness evaluations of the offspring.

Updating the search distribution consists of two steps, *environmental selection* and sometimes *strategy adaptation* of external strategy parameters: A selection method chooses μ new parents $\tilde{\boldsymbol{g}}_1^{(t+1)}, \ldots, \tilde{\boldsymbol{g}}_\mu^{(t+1)}$ from $\tilde{\boldsymbol{g}}_1^{(t)}, \ldots, \tilde{\boldsymbol{g}}_\mu^{(t)}$ and $\boldsymbol{g}_1^{(t)}, \ldots, \boldsymbol{g}_\lambda^{(t)}$. This second selection process is called environmental selection and may be deterministic or stochastic. Either the mating or the environmental selection must be based on the objective function values of the individuals and must prefer those with better fitness—this is the driving force of the evolutionary adaptation process. An example of fitness-dependent environmental selection is choosing the μ best individuals out of $\lambda > \mu$ offspring. In addition, the EA may update external strategy parameters as discussed in the following section.

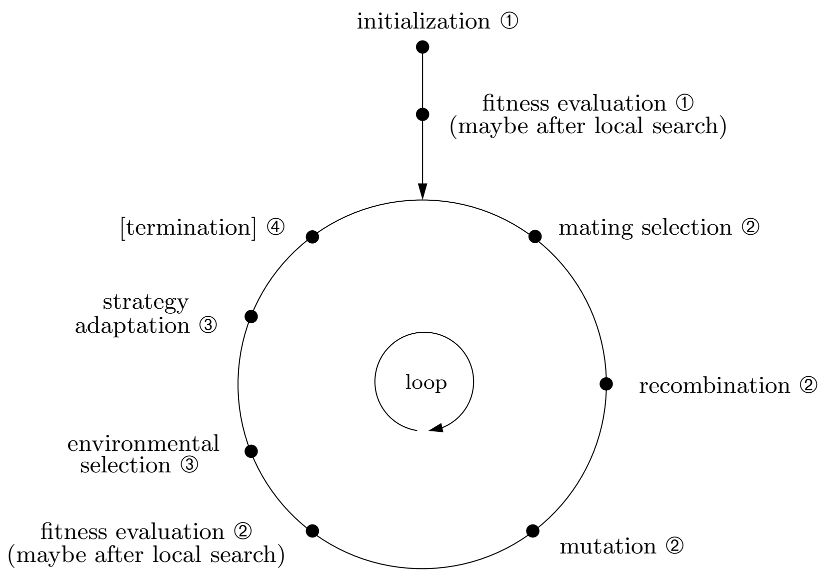

Figure 1: Basic EA loop. The numbers indicate the corresponding steps in the random search scheme. When optimizing adaptive systems, the local search usually corresponds to some learning process.

2.1 Strategy Adaptation

Strategy adaptation, that is the automatic adjustment of the search strategy during the optimization process, is a key concept to improve the performance in evolutionary computation [11, 24, 43]. It is necessary because the best search strategy for a problem is usually not known in advance and typically changes (e.g., from coarse to fine) during optimization. Examples of strategy parameters that can be controlled externally include population sizes, the probabilities that certain variation operators are applied, and parameters that determine the mutation strength.

In the following section, we describe an efficient algorithm for adjusting the covariance matrix of Gaussian mutations in EAs. Thereafter, we present a way for adaptation of the application probabilities of variation operators. Both methods are deterministic and monitor the effects of the variation operators over the generations. They are based on the same rule of thumb that recent beneficial mutations are also likely to be beneficial in the following generations.

2.1.1 The CMA Evolution Strategy

Evolution strategies (ES, [3, 32, 38]) are one of the main branches of EAs. In the following, we describe the covariance matrix adaptation ES (CMA-ES) proposed in [18, 19], which performs efficient real-valued optimization. Each individual represents an n-dimensional real-valued object variable vector. These variables are altered by two variation operators, intermediate recombination and additive Gaussian mutation. The former 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. In the CMA-ES, the complete covariance matrix of the Gaussian mutation distribution is adapted during evolution to improve the search strategy. More formally, the object parameters $\boldsymbol{g}_k^{(t)}$ of offspring $k = 1, \ldots, \lambda$ created in generation t are given by

$$\boldsymbol{g}_k^{(t)} = \langle \tilde{\boldsymbol{g}} \rangle^{(t)} + \sigma^{(t)} \boldsymbol{B}^{(t)} \boldsymbol{D}^{(t)} \boldsymbol{z}_k^{(t)} \quad , \tag{3}$$

where $\langle \tilde{\boldsymbol{g}} \rangle^{(t)} = \frac{1}{\mu} \sum_{i=1}^{\mu} \tilde{\boldsymbol{g}}_i{}^{(t)}$ is the center of mass of the parent population in generation t and the $\boldsymbol{z}_k^{(t)} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ are independent realizations of an n-dimensional normally distributed random vector with zero mean and covariance matrix equal to the identity matrix $\boldsymbol{I}$. The covariance matrix $\boldsymbol{C}^{(t)}$ of the random vectors

$$\sigma^{(t)} \boldsymbol{B}^{(t)} \boldsymbol{D}^{(t)} \boldsymbol{z}_k^{(t)} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{C}^{(t)}) \tag{4}$$

is a symmetric positive $n \times n$ matrix with

$$\boldsymbol{C'}^{(t)} = \boldsymbol{C}^{(t)} / {\sigma^{(t)}}^2 = \boldsymbol{B}^{(t)} \boldsymbol{D}^{(t)} \left(\boldsymbol{B}^{(t)} \boldsymbol{D}^{(t)} \right)^T \quad . \tag{5}$$

The columns of the orthogonal $n \times n$ matrix $\boldsymbol{B}^{(t)}$ are the normalized eigenvectors of $\boldsymbol{C'}^{(t)}$ and $\boldsymbol{D}^{(t)}$ is a $n \times n$ diagonal matrix with the square roots of the corresponding eigenvalues. Figure 2 schematically shows the transformations of $\boldsymbol{z}_k^{(t)}$ by $\boldsymbol{B}^{(t)}$ and $\boldsymbol{D}^{(t)}$.

The strategy parameters, both the matrix $\boldsymbol{C'}^{(t)}$ and the so called global step-size $\sigma^{(t)}$, are updated online using the covariance matrix adaptation (CMA) method. The key idea of the CMA is to alter the mutation distribution in a deterministic way such that the probability to reproduce steps in the search space that have led to the current population is increased. This enables the algorithm to detect correlations between object variables and to become invariant under orthogonal transformations of the search space (apart from the initialization). In order to use the information from previous generations 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 for environmental selection. That is, the μ best of the λ offspring form the next parent population. After

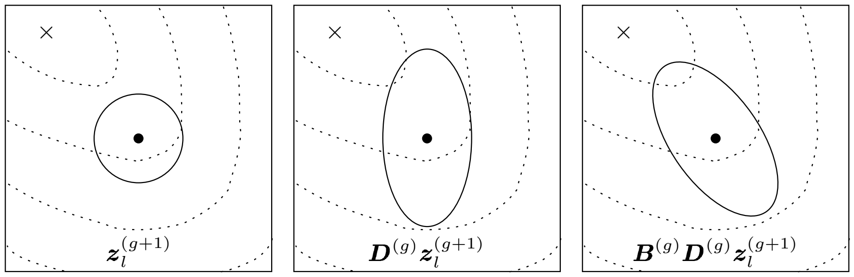
$\boldsymbol{z}_l^{(g+1)}$
$\boldsymbol{D}^{(g)}\boldsymbol{z}_l^{(g+1)}$
$\boldsymbol{B}^{(g)}\boldsymbol{D}^{(g)}\boldsymbol{z}_l^{(g+1)}$

Figure 2: The dashed lines schematically visualize an error / fitness surface (landscape) for $\mathcal{G} \subset \mathbb{R}^2$, where each line represents points of equal fitness and the $\times$ symbol marks the optimum. The dot corresponds to the center of mass of the parent population and the solid lines indicate the mutation (hyper-) ellipsoids (i.e., surfaces of equal probability density to place an offspring) of the random vectors after the different transformations. Evolution strategies that adapt only one global step size can only produce mutation ellipsoids as shown in the left plot. Algorithms that adapt n different step sizes, one for each object variable, can produce mutation ellipsoids scaled along the coordinate axes like the one shown in the center plot. Only if the complete covariance matrix is adapted, arbitrary normal distributions can be realized as shown in the right picture.

selection, the strategy parameters are updated:

$$\boldsymbol{s}^{(t+1)} = (1-c)\cdot\boldsymbol{s}^{(t)} + c_{\mathrm{u}}\cdot\underbrace{\sqrt{\mu}\,\boldsymbol{B}^{(t)}\boldsymbol{D}^{(t)}\left\langle\boldsymbol{z}\right\rangle_{\mu}^{(t)}}_{\frac{\sqrt{\mu}}{\sigma^{(t)}}\left(\langle\tilde{\boldsymbol{g}}\rangle^{(t+1)}-\langle\tilde{\boldsymbol{g}}\rangle^{(t)}\right)} \tag{6}$$

$$\boldsymbol{C'}^{(t+1)} = (1-c_{\mathrm{cov}})\cdot\boldsymbol{C'}^{(t)} + c_{\mathrm{cov}}\cdot\boldsymbol{s}^{(t+1)}\left(\boldsymbol{s}^{(t+1)}\right)^{\mathrm{T}} \quad . \tag{7}$$

Herein, $\boldsymbol{s}^{(t+1)} \in \mathbb{R}^n$ is the evolution path—a weighted sum of the centers of the population over the generations starting from $\boldsymbol{s}^{(2)} = \sqrt{\mu}\,\boldsymbol{B}^{(1)}\boldsymbol{D}^{(1)}\left\langle\boldsymbol{z}\right\rangle_{\mu}^{(1)}$ (the factor $\sqrt{\mu}$ compensates for the loss of variance due to computing the center of mass). The parameter $c \in\,]0,1]$ controls the time horizon of the adaptation of $\boldsymbol{s}$; we set $c = 1/\sqrt{n}$. The constant $c_{\mathrm{u}} = \sqrt{c(2-c)}$ normalizes the variance of $\boldsymbol{s}$ (viewed as a random variable) as $1^2 = (1-c)^2 + c_{\mathrm{u}}^2$. The expression $\left\langle\boldsymbol{z}\right\rangle_{\mu}^{(t)} = \frac{1}{\mu}\sum_{i=1}^{\mu}\boldsymbol{z}_{i:\lambda}^{(t)}$ is the average of the realizations of the random vector that led to the new parent population, where $i{:}\lambda$ denotes the index of the offspring having the ith best fitness value of all offspring in the current generation, that is $\{\{\boldsymbol{g}_{1:\lambda}^{(t)},\dots,\boldsymbol{g}_{\mu:\lambda}^{(t)}\}\} = \{\{\tilde{\boldsymbol{g}}_1^{(t+1)},\dots,\tilde{\boldsymbol{g}}_{\mu}^{(t+1)}\}\}$. The parameter $c_{\mathrm{cov}} \in [0,1[$ controls the update of $\boldsymbol{C'}^{(t)}$ and we set it to $c_{\mathrm{cov}} = 2/(n^2+n)$.

The update rule (7) shifts $\boldsymbol{C'}^{(t)}$ towards the $n \times n$ matrix $\boldsymbol{s}^{(t+1)}\left(\boldsymbol{s}^{(t+1)}\right)^T$ making mutation steps in the direction of $\boldsymbol{s}^{(t+1)}$ more likely. The vector $\boldsymbol{s}$ does not only represent the last (adaptive) step of the parent population, but a time average over all previous adaptive steps. The influence of previous steps decays exponentially, where the decay rate is controlled by c.

The adaptation of the global step-size parameter σ is done separately on a shorter timescale (a single parameter can be estimated based on less samples compared to the complete covariance matrix). We keep track of a second evolution path $\boldsymbol{s}_\sigma$ without the scaling by $\boldsymbol{D}$:

$$\boldsymbol{s}_\sigma^{(t+1)} = (1 - c_\sigma) \cdot \boldsymbol{s}_\sigma^{(t)} + c_{\mathrm{u}_\sigma} \cdot \underbrace{\sqrt{\mu}\,\boldsymbol{B}^{(t)}\,\langle \boldsymbol{z} \rangle_\mu^{(t+1)}}_{\boldsymbol{B}^{(t)}\left(\boldsymbol{D}^{(t)}\right)^{-1}\left(\boldsymbol{B}^{(t)}\right)^{-1}\frac{\sqrt{\mu}}{\sigma^{(t)}}\left(\langle \boldsymbol{g} \rangle_\mu^{(t+1)} - \langle \boldsymbol{g} \rangle_\mu^{(t)}\right)} \tag{8}$$

$$\sigma^{(t+1)} = \sigma^{(t)} \cdot \exp\left(\frac{\|\boldsymbol{s}_\sigma^{(t+1)}\| - \hat{\boldsymbol{\chi}}_n}{d \cdot \hat{\boldsymbol{\chi}}_n}\right) \quad , \tag{9}$$

where $\hat{\boldsymbol{\chi}}_n$ is the expected length of a n-dimensional, normally distributed random vector with covariance matrix $\boldsymbol{I}$. The damping parameter $d \geq 1$ decouples the adaptation rate from the strength of the variation. We set $d = \sqrt{n}$ and start from $\boldsymbol{s}_\sigma^{(2)} = \sqrt{\mu}\,\boldsymbol{B}^{(1)}\,\langle \boldsymbol{z} \rangle_\mu^{(1)}$. The parameter $c_\sigma \in]0, 1]$ controls the update of $\boldsymbol{s}_\sigma$. Here, we use $c_\sigma = c$. Setting $c_{\mathrm{u}_\sigma} = \sqrt{c_\sigma(2 - c_\sigma)}$ normalizes the variance of $\boldsymbol{s}_\sigma$.

The evolution path $\boldsymbol{s}_\sigma$ is the sum of normally distributed random variables. Because of the normalization, its expected length would be $\hat{\boldsymbol{\chi}}_n$ if there were no selection. Hence, the rule (9) basically increases the global step-size if the steps leading to selected individuals have been larger than expected and decreases the step size in the opposite case.

The CMA-ES needs only small population sizes. These are chosen according to the heuristic $\lambda = 4 + \lfloor 3 \ln n \rfloor$ and $\mu = \lfloor \lambda/4 \rfloor$. Note that almost all the parameters of the algorithm can be set to the default values given in [18, 19]. The initializations of $\boldsymbol{C}'$ and σ allow for incorporation of prior knowledge about the scaling of the search space. In the following, we set $\boldsymbol{C'}^{(1)} = \boldsymbol{I}$ and choose $\sigma^{(1)}$ dependent on the problem.

2.1.2 Adaptation of Operator Probabilities

The search strategy is mainly determined by the variation operators and the probabilities of their application. In the CMA-ES, there is only one mutation and one recombination operator, which are both always applied, and strategy adaptation corresponds to adjusting the parameters of the mutation operator. Now we consider the case when there are several mutation operators and the strategy parameters to adapt are the probabilities of application of these operators.

The algorithm we propose combines concepts from [10] and from the CMA-ES. Let Ω denote a set of mutation operators. Each time an offspring $\boldsymbol{g}_i^{(t)}$ is created from a parent, first the number $v_i^{(t)}$ of variations is determined, then $v_i^{(t)}$ operators are randomly chosen from Ω and applied successively. Let $p_o^{(t)}$ be the probability that $o \in \Omega$ is chosen at generation t. Further, let $O_o^{(t)}$ contain all offspring produced in generation t by an application of the operator o. The case when an offspring is produced by applying more than one operator is treated as if the offspring has been generated several times, once by each of the operators involved. The operator probabilities are updated every τ generations. This period is called an adaptation cycle. The average performance achieved by an operator o over an adaptation cycle is measured by

$$q_o^{(t,\tau)} = \sum_{i=0}^{\tau-1} \sum_{\boldsymbol{g} \in O_o^{(t-i)}} \max(0, f(\boldsymbol{g}) - f(\mathrm{parent}(\boldsymbol{g}))) \Big/ \sum_{i=0}^{\tau-1} \left|O_o^{(t-i)}\right| \ , \tag{10}$$

where parent($\boldsymbol{g}$) denotes the parent of offspring $\boldsymbol{g}$ (and we assume a fitness maximization task). The operator probabilities $p_o^{(t+1)}$ are adjusted every $\tau = 4$ generations according to

$$s_o^{(t+1)} = \begin{cases} c_\Omega \cdot q_o^{(t,\tau)}/q_{\mathrm{all}}^{(t,\tau)} + \ (1-c_\Omega) \cdot s_o^{(t)} & \text{if } q_{\mathrm{all}}^{(t,\tau)} > 0 \\ c_\Omega/|\Omega| + \ (1-c_\Omega) \cdot s_o^{(t)} & \text{otherwise} \end{cases} \tag{11}$$

and

$$p_o^{(t+1)} = p_{\min} \ + \ (1 - |\Omega| \cdot p_{\min}) s_o^{(t+1)} \Big/ \sum_{o' \in \Omega} s_{o'}^{(t+1)} \ . \tag{12}$$

The factor $q_{\mathrm{all}}^{(t,\tau)} = \sum_{o' \in \Omega} q_{o'}^{(t,\tau)}$ is used for normalization and $s_o^{(t+1)}$ stores the weighted average of the quality of the operator o, where the influence of previous adaptation cycles decreases exponentially. The rate of this decay is controlled by $c_\Omega \in (0,1]$, which is set to $c_\Omega = 0.3$ in our experiments. The operator probability $p_o^{(t+1)}$ is computed from the weighted average $s_o^{(t+1)}$, such that all operator probabilities sum to one and are bounded from below by $p_{\min} < 1/|\Omega|$. Initially, $s_o^{(0)} = p_o^{(0)}$ for all $o \in \Omega$. Note that s_o has a similar function as the evolution paths in the CMA-ES. A more detailed description and an empirical evaluation of the operator adaptation algorithm is given in [24].

3 Evolutionary Optimization of Neural Networks

Learning of an adaptive (e.g., neural) system can be defined as goal-directed, data-driven changing of its behavior. The major components of an adaptive system can be described by a triple $(\mathcal{S}, \mathcal{A}, \mathcal{D})$, where $\mathcal{S}$ stands for the structure or architecture of the adaptive system, $\mathcal{A}$ is a learning algorithm that operates on $\mathcal{S}$ and adapts flexible parameters of the system, and $\mathcal{D}$ denotes the sample data. Examples of learning algorithms for technical NNs include gradient-based heuristics, see section 3.1, or quadratic program solvers, see section 3.3. Such "classical" optimization methods are usually considerably faster than pure evolutionary optimization of NN parameters [22, 26, 45], although they might be more prone to getting stuck in local minima. However, there are cases where "classical" optimization methods are not applicable, for example when the neural model or the objective function is non-differentiable as in section 3.2. Still, the main application of evolutionary optimization in the field of neurocomputing is adapting the structures of neural systems, that is, optimizing those parts that are not altered by the learning algorithm. Both in biological and technical neural systems the structure is crucial for the learning behavior—the evolved structures of brains are an important reason for their incredible learning performance: "development of intelligence requires a balance between innate structure and the ability to learn" [2]. Hence, it appears to be obvious to apply evolutionary methods for adapting the structure of neural systems for technical applications, a task for which generally no efficient "classical" methods exist. A prototypical example of evolutionary optimization of a neural architecture on which a learning algorithm operates is given in the following section 3.1, where the topology and the weights of multi-layer perceptron network are optimized. Adopting the extended definition of structure as that part of the adaptive system that cannot be optimized by the learning algorithm itself, section 3.3 presents the optimization of the "structure" of support vector machines.

3.1 Evolving Neural Networks for Face Detection

Feed forward NNs have proven to be powerful tools in pattern recognition [53]. For example, they can be used to decide whether images of a fixed size contain a complete frontal upright face or not, see Fig. 3 (left). In the following, we discuss the evolutionary optimization of a feed forward multi-layer perceptron for such a face detection task. The optimization process has two objectives: improving the classification accuracy and the speed of processing.

3.1.1 Feed Forward Multi-Layer Perceptrons

The structure of a multi-layer perceptron (MLP, a good introduction is given in [5]) is given by a connected directed graph $G = (\mathcal{V}, \mathcal{E})$ with vertices $\mathcal{V}$ and edges $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$. We identify the vertices with neurons and edges with connecting synapses. If G is acyclic it describes a feed forward MLP. The n_{out} nodes without successors are called the output neurons. The $n_{\text{in}} + 1$ nodes without predecessors are the input neurons and an additional bias unit. The n_{hidden} nodes with at least one successor and at least one predecessor are called hidden neurons. Each feed forward MLP represents a static function that maps an input $\boldsymbol{x} \in \mathbb{R}^{n_{\text{in}}}$ to an output value $\boldsymbol{y} \in \mathbb{R}^{n_{\text{out}}}$. Traversing the graph from the input neurons, we compute

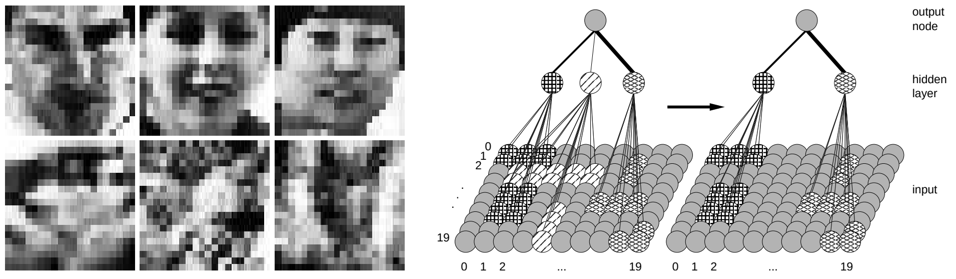

Figure 3: Left, the input data to the face detection network are preprocessed 20×20 pixel grayscale images showing either frontal, upright faces or nonface examples. Right, scheme of the *delete-node* operator. The linewidths indicate the magnitude of the corresponding weight values.

the activation z_i of each neuron i. The activation of the input nodes is equal to the corresponding component of the input pattern $\boldsymbol{x}$, the activation of the additional node without predecessors (the bias node) is constantly equal to 1. For the other nodes the activation is given by

$$z_i = g\left(\sum_{j \in \mathrm{pred}(i)} w_{ij} z_j\right) \quad , \tag{13}$$

where $\mathrm{pred}(i) \subset \mathcal{V}$ is the set of predecessors, w_{ij} is the weightening factor of the connection from neuron j to neuron i, and g is a non-linear transfer function, here the sigmoidal $g(u) = u/(|u|+1)$. The activations of the output nodes correspond to the components of the output $\boldsymbol{y}$ of the network.

Learning of a MLP means adapting the weights w_{ij} based on some sample input-output patterns $\{(\boldsymbol{x}_1, \boldsymbol{y}_1), \dots, (\boldsymbol{x}_\ell, \boldsymbol{t}_\ell)\}$. This is usually done by gradient-based minimization of the (squared) differences between the targets $\boldsymbol{t}_i$ and the corresponding outputs $\boldsymbol{y}_i$ of the NN given the input $\boldsymbol{x}_i$. The goal is not to learn the training patterns by heart, but to find a statistical model for the underlying relationship between input and output data. Such a model will generalize well, that is, will make good predictions for cases other than the training patterns. A critical issue is therefore to avoid overfitting during the learning process: The NN should just fit the signal and not the noise. This is usually achieved by restricting the effective complexity of the network, for example by regularization of the learning process.

3.1.2 Evolving Neural Face Detectors

Real-time face recognition requires both fast and accurate face detection methods. Recognition speed may be crucial, for example when processing a huge amount of data from video streams. As stated in a recent survey "The advantage of using neural networks for face detection is the feasibility of training a system to capture the complex class conditional density of face patterns. However, one drawback is that the network architecture has to be extensively tuned (number of layers, number of nodes, learning rates, etc.) to get exceptional performance" [51]. In the following, we show how evolutionary computation can help to overcome this drawback. We optimize the weights and the structure of an already existing neural face classifier, which is part of a complex face detection system similar to the one described in [35]. We consider an implementation in which the speed of classification scales approximately linearly with the number of hidden neurons. Therefore our goal is to reduce the number of neurons of a NN without loss of classification accuracy, whereas we tolerate an increase in the number of connections.

We apply an EA combined with gradient-based learning as schematically shown in Fig. 1. Each individual encodes a NN. In every generation, each parent creates one offspring, which inherits its parent's genotype. The offspring's genotype is then altered by elemental variation operators. These are chosen randomly from a set Ω of 8 different operators and are applied sequentially. The process of choosing and applying an operator is repeated $1 + \kappa$ times, where κ is an individual realization of a Poisson distributed random number with mean 1. There are 5 basic operators: *add-connection*, *delete-connection*, *add-node*, *delete-node*, and *jog-weights*. Their effects on the network graph represented by the genotype becomes obvious from their names, probably except for the operator *jog-weights*. The latter adds Gaussian noise to the weights in order to push the weight configuration out of local minima. The elemental deletion operators are based on the *magnitude based pruning* heuristic (see [33]), which assigns a higher probability to the deletion of small weights. As an example, the *delete-node* operator is schematically depicted in Fig. 3 (right). In addition to the 5 basic operators, there are 3 task-specific mutations inspired by the concept of "receptive fields" (RFs). These RF-operators *add-RF-connection*, *delete-RF-connection*, and *add-RF-node* behave as their basic counterparts, but act on groups of connections. These groups are defined by rectangular regions of the input image, see the input layer in Fig. 3 (right). The RF-operators consider the topology of the image plane by taking into account that "isolated" processing of pixels is rarely useful for object detection. Not all operators might be necessary at all stages of the evolutionary process and questions such as when fine-tuning becomes more important than operating on receptive fields cannot be answered in advance. Hence, the application probabilities of the 8 variation operators are adapted using the method described in section 2.1.2.

An inner loop of learning is embedded, just before fitness evaluation in order to fine-tune all weights. An iterative learning algorithm is used, namely the improve Rprop algorithm [23, 34]. Learning is done for a fixed number of iterations[1]

[1]A method that automatically adjusts the length of the learning period similar to the evolutionary strategy adaptation described in section 2.1 has been proposed in [20].

and is stopped earlier when the generalization performance of the network, which is measured using a validation data set, decreases. The weight configuration with the smallest error on training and validation sample data found during network learning is regarded as the outcome of the learning process and is stored in the genome of the corresponding individual (this principle is often called Lamarckian inheritance). Based on this weight configuration the fitness of the individual, a weighted sum of the classification accuracy and the number of neurons, is calculated.

3.1.3 Experimental Evaluation

We initialized the 25 individuals in the population of our EA with the expert-designed architecture proposed in [35]. This network has been tailored to the face detection task and has become the standard reference for neural network based face detection, see [51]. In the following, all results are given relative to the properties of the initial architecture.

Our EA successfully tackled the problem of reducing the number of hidden neurons of the face detection network without loss of detection accuracy [49]. The numbers of hidden neurons of the evolved networks are reduced by 27-35 %. This means, we could improve the speed of classification whether an image region corresponds to a face or not by approximately 30 %. By speeding up classification, the rate of complete scans of video-stream images of face recognition systems can be increased leading to a more accurate recognition. A generalization performance test on an external data set, which is independent from all data used for optimization, demonstrates that most of our considerably smaller networks perform at least as good as the expert-designed architecture. Some of the evolved classifiers even show an improvement of the classification accuracy by more than 10 %.

3.2 Adapting the Weights of Networks for Reinforcement Learning Tasks

Reinforcement (RL, [47]) learning is an important, biologically plausible learning paradigm. In RL the feedback about the performance of an adaptive system may be sparse, unspecific, and delayed. Evolutionary algorithms have proven to be powerful and competitive approaches compared to standard RL methods [28]. The recent success of evolved NNs in game playing [7] underlines the potential of the combination of NNs and evolutionary computation for RL. In the following, we describe an application of the CMA-ES to the adaptation of the weights of a NN for solving a RL task. In this scenario, no gradient information is available to adapt the NN parameters.

3.2.1 Reinforcement Learning

In the standard RL scenario, an agent interacts with its environment at discrete time steps t. It perceives the environment to be in state $\boldsymbol{s}_t \in S$ and chooses a behavior a_t from the set of actions A according to its policy $\pi : S \rightarrow A$. After the execution of action a_t, the environment makes a possibly stochastic transition to a perceived state $\boldsymbol{s}_{t+1}$ 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 discounted cumulative future reward* $R_t = \sum_{t'=t+1}^{\infty} \gamma^{t'-t-1} r_{t'}$ with discount rate $\gamma \in]0,1]$ is maximized. Two different (model-free) approaches to solve RL problems can be distinguished. The most common methods such as temporal-difference learning algorithms adapt value functions [47]. Usually, they learn a state-value function $V : S \rightarrow \mathbb{R}$ or a state-action-value function $Q : S \times A \rightarrow \mathbb{R}$ for judging states or state-action pairs, respectively. The policy π is then defined on top of this function. The second approach is to search directly in the space of policies [28]. However, the gradient $\partial R_t \,/\, \partial \pi_t$ can usually not be computed (this problem can be circumvented be *actor-critic* architectures, see [47]). When S or A is too large or generalization from experiences to new states and actions is desired, function approximators like NNs are used to model Q, V, or π.

The potential advantages of direct search methods like EAs compared to standard RL methods are that they

1. allow for direct search in the policy space,
2. are often easier to apply and are more robust with respect to the tuning of the meta-parameters (learning rates, etc.),
3. can be applied if the function approximators are non-differentiable, and
4. can also optimize the underlying structure of the function approximators.

In the following, we describe an application of the CMA-ES to the adaptation of the weights of an NN that directly represents a policy π.

3.2.2 Evolving Networks for Pole Balancing

Pole balancing problems (also known as inverted pendulum problems) are standard benchmark tasks for EAs that adapt NNs for control, see [50] for early and [46] for recent references. In our example, the task is to balance two 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. Figure 4 illustrates the task (the corresponding equations of motion are given, e.g., in [21, 50]). 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.

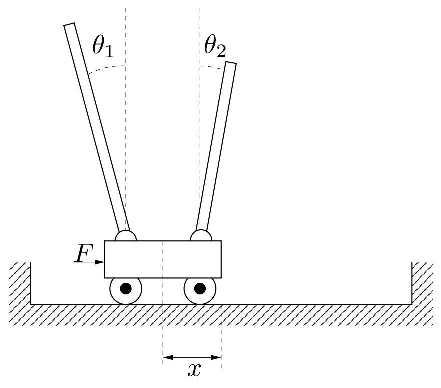

Figure 4: Double pole balancing problem. The parameters x, θ_1, and θ_2 are the offset of the cart from the center of the track and the angles from the vertical of the long and short pole, respectively.

We consider the *double pole without velocities* scenario, where the controller gets only x, θ_1, and θ_2 as inputs and the output is a force between $-10\,N$ and $10\,N$. The environment is only partially observable as information about the velocity of the cart and the angle velocities are needed for a successful control strategy. 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 neural networks (RNNs). In our example, we use the following simple RNN architecture. Let $z_i(t)$ for $0 < i \leq n_{\text{in}}$ be the activation of the n_{in} input units at time step t of a network with a total of n_{neurons} neurons. The activation of the other neurons is given by

$$z_i(t) = g\left(\sum_{j=1}^{n_{\text{in}}} w_{ij} z_j(t) + \sum_{j=n_{\text{in}}+1}^{n_{\text{neurons}}} w_{ij} z_j(t-1)\right) \tag{14}$$

for $n_{\text{in}} < i \leq n_{\text{neurons}}$, where the w_{ij} are the weights and g is a non-linear transfer function, see section 3.1.1. All RNNs in this study have a single output neuron and the input signals provided to the networks are appropriately scaled.

The weights of the RNNs are adapted by the CMA-ES [21]. We use the same fitness function as in [17], see also [16, 21, 46]. This quality measure consists of two additive terms. The first addend is proportional to the number of time steps the controller manages to balance the poles starting from a fixed initial position (i.e., we do not test for generalization). The second term penalizes oscillations in order to exclude the control strategy to balance the poles by just moving the cart quickly back and forth from the set of solutions. A trial is stopped and regarded as successful when the fittest individual in the population balances the poles for

10^5 time steps.

method	evaluations	population size
ESP	6213	100
NEAT	6326	100
CMA-ES, $n_{\text{hidden}} = 3$	3521	13
CMA-ES, $n_{\text{hidden}} = 5$	4856	13
CMA-ES, $n_{\text{hidden}} = 7$	5029	16

Table 1: Number of balancing attempts needed to find an appropriate control strategy averaged over 75 (ESP and NEAT) and 50 (CMA-ES) trials, respectively. Additionally, the population sizes used in the experiments are given.

The results of our experiments using RNN architectures with different numbers of neurons are shown in table 1. They are compared to the best performing methods for evolving RNNs for this task so far, the *Enforced Sub-Population* (ESP, [16]) and the *NeuroEvolution of Augmenting Topologies* (NEAT, [46]) algorithm. In contrast to our approach, NEAT does not only adapt the weights of NNs, but also their structure—this weakens the comparison. Note that the ESP and NEAT results are better than those reported in [16, 21, 46], because they refer to new trials with optimized settings of the algorithms, in particular with reduced population sizes (Kenneth O. Stanley, private communication). However, even for $n_{\text{hidden}} = 7$ the CMA-ES performs statistically significantly better (t-test, $p < .05$) than the ESP method. The smaller the network size, the better are the results obtained by the CMA-ES.

Our experiments show that that standard RNN architectures combined with the CMA-ES are sufficient for achieving good results on pole balancing tasks when the architecture of the NNs are fixed. The efficient adaptation of the search strategy by the CMA-ES allows for fast optimization of the network weights by detecting correlations between object variables without requiring large population sizes. Still, the network structure matters, which becomes obvious from the differences between architectures with different numbers of hidden neurons. Hence, methods are required that evolve both the structure and the weights of NNs for RL tasks—although for nearly all initializations our approach outperformed the existing algorithms that additionally adapt the topology (more results and details are given in [21]).

3.3 Evolutionary Tuning of Support Vector Machines

Support vector machines (SVMs, e.g., [9, 37, 48]), which can be viewed as special regularization networks (at least there is a very close relation, see [13]), have become a standard method in machine learning. The main idea of SVMs for binary classification is to map the input vectors to a feature space and to classify the transformed data by a linear function that promises good generalization. The transformation is done implicitly by a kernel, which computes an inner product in the feature space. Constructing the linear decision function is a convex optimization problem that can be solved using quadratic programming. Thus, the problem of designing an appropriate model is elegantly separated into two stages: First, the kernel function (and its parameters) has to be chosen. In the general case of non-separable data, one also has to select a regularization parameter, which controls the trade-off between minimizing the training error and the complexity of the decision function. Second, the corresponding convex optimization problem has to be solved.

The first stage can be identified with choosing the structure of the adaptive systems, and the second stage with adapting its parameters by learning. The latter can be solved efficiently by "classical" algorithms, the former is a difficult multimodal optimization task requiring the use of heuristics. Often a parameterized family of kernel functions is considered and the kernel adaptation reduces to finding an appropriate parameter vector for the given problem. These parameters together with the regularization parameter are called the hyperparameters of the SVM.

In practice the hyperparameters are usually determined by grid search. That is, the hyperparameters are varied with a fixed step-size through a wide range of values and the performance of every combination is assessed using some performance measure. Because of the computational complexity, grid search is only suitable for the adjustment of very few parameters. Perhaps the most elaborate alternative techniques for choosing multiple hyperparameters are gradient descent methods [6, 8, 15]. However, these approaches have some severe drawbacks, for example: The kernel function has to be differentiable, which excludes for example string kernels. The score function for assessing the performance of the hyperparameters (or at least an accurate approximation of this function) also has to be differentiable with respect to kernel and regularization parameters, which excludes reasonable measures such as the number of support vectors.

In the following, we present the application of the CMA-ES for optimizing SVM hyperparameters—a new approach that does not suffer from the limitations described above [14]. Beforehand, we give a concise formal description of SVMs.

3.3.1 Support Vector Machines

We consider L_1-norm soft margin SVMs for the discrimination of two classes. Let $(\boldsymbol{x}_i, t_i), 1 \leq i \leq \ell$, be the training examples, where $t_i \in \{-1, 1\}$ is the label associated with input pattern $\boldsymbol{x}_i \in \mathbb{R}^{n_{\text{in}}}$. The mapping $\phi : \mathbb{R}^{n_{\text{in}}} \to F$ of the input vectors to the feature space F is implicitly done by a kernel $K : \mathbb{R}^{n_{\text{in}}} \times \mathbb{R}^{n_{\text{in}}} \to \mathbb{R}$. The kernel computes an inner product in the feature space, that is $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j) \rangle$. The linear function for classification in the feature space is chosen according to a bound on the generalization error. This bound takes a target margin and the margin slack vector into account (cf. [9, 37]). The latter corresponds to the amounts by which individual training patterns fail to meet the target margin.

This leads to the SVM decision function

$$h(\boldsymbol{x}) = \operatorname{sign}\left(\sum_{i=1}^{\ell} t_i \alpha_i^* K(\boldsymbol{x}_i, \boldsymbol{x}) + b\right) \ , \tag{15}$$

where we define $\operatorname{sign}(x) = -1$ if $x < 0$ and $\operatorname{sign}(x) = 1$ otherwise. The coefficients α_i^* are the solution of the following quadratic optimization problem:

$$\begin{aligned} \text{maximize} \quad & W(\boldsymbol{\alpha}) = \sum_{i=1}^{\ell} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{\ell} t_i t_j \alpha_i, \alpha_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) \qquad (16) \\ \text{subject to} \quad & \sum_{i=1}^{\ell} \alpha_i t_i = 0 \\ & 0 \le \alpha_i \le C, \quad i = 1, \dots, \ell . \end{aligned}$$

The optimal value for b can then be computed based on the solution $\boldsymbol{\alpha}^*$. The vectors $\boldsymbol{x}_i$ with $\alpha_i > 0$ are called support vectors. The regularization parameter C controls the trade-off between maximizing the target margin and minimizing the L_1-norm of the margin slack vector of the training data.

3.3.2 Evolving SVM Hyperparameters

We consider general Gaussian kernels

$$K_{\boldsymbol{A}}(\boldsymbol{x}, \boldsymbol{x}') = e^{-(\boldsymbol{x}-\boldsymbol{x}')^T \boldsymbol{A} (\boldsymbol{x}-\boldsymbol{x}')} \ , \tag{17}$$

where $\boldsymbol{x}, \boldsymbol{x}' \in \mathbb{R}^{n_{\text{in}}}$ and $\boldsymbol{A}$ is a symmetric positive definite $n_{\text{in}} \times n_{\text{in}}$ matrix. We allow arbitrary symmetric positive definite matrices $\boldsymbol{A}$; this means the input space can be scaled and rotated. The individuals in the ES encode C and the kernel parameters. When encoding $\boldsymbol{A}$, we have to ensure that after variation the genotype still corresponds to a feasible (i.e., symmetric, positive definite) matrix. We make use of the fact that for any symmetric and positive definite $n \times n$ matrix $\boldsymbol{A}$ there exists an orthogonal $n \times n$ matrix $\boldsymbol{T}$ and a diagonal $n \times n$ matrix $\boldsymbol{D}$ with positive entries such that $\boldsymbol{A} = \boldsymbol{T}^T \boldsymbol{D} \boldsymbol{T}$ and

$$\boldsymbol{T} = \prod_{i=1}^{n-1} \prod_{j=i+1}^{n} \boldsymbol{R}(\alpha_{i,j}) \ , \tag{18}$$

as proven in [36]. The $n \times n$ matrices $\boldsymbol{R}(\alpha_{i,j})$ are elementary rotation matrices. These are equal to the unit matrix except for $[\boldsymbol{R}(\alpha_{i,j})]_{ii} = [\boldsymbol{R}(\alpha_{i,j})]_{jj} = \cos\alpha_{ij}$ and $[\boldsymbol{R}(\alpha_{i,j})]_{ji} = -[\boldsymbol{R}(\alpha_{i,j})]_{ij} = \sin\alpha_{ij}$.

When using the evolution strategy, each genotype encodes the $n_{\text{in}} + ({n_{\text{in}}}^2 - n_{\text{in}})/2 + 1$ parameters

$$(C', d_1, \dots, d_{n_{\text{in}}}, \alpha_{1,2}, \alpha_{1,3}, \dots, \alpha_{1,n_{\text{in}}}, \alpha_{2,3}, \alpha_{2,4}, \dots, \alpha_{2,n_{\text{in}}}, \dots, \alpha_{n_{\text{in}}-1,n_{\text{in}}}) \ . \tag{19}$$

data		accuracy on test set	# SV
Breast-Cancer	grid-search	74.51	113.52
	evolutionary tuned	$75.38 \pm 0.42^\star$	$112.70 \pm 0.68^\star$
Diabetes	grid-search	76.67	247.83
	evolutionary tuned	76.73 ± 0.32	$235.73 \pm 3.43^\star$
Heart	grid-search	84.79	106.33
	evolutionary tuned	$85.14 \pm 0.33^\star$	$75.51 \pm 1.5^\star$
Thyroid	grid-search	95.83	16.36
	evolutionary tuned	$96.01 \pm 0.05^\star$	$15.42 \pm 0.18^\star$

Table 2: Results averaged over 20 trials ± standard deviations. The first column specifies the medical benchmark. The second column indicates whether the results refer to the initial grid search values or to the evolutionary optimized kernels. The percentages of correctly classified patterns on the test sets (averaged over 100 different partitions into training and test data) are given as well as the average numbers of support vectors (# SV). Results that are statistically significantly better compared to grid-search are marked with $\star$ (two-sided t-test, $p < .05$).

We encode $\boldsymbol{A}$ according to (18) and set $\boldsymbol{D} = \operatorname{diag}(|d_1|, \ldots, |d_n|)$ and $C = |C'|$.

We evaluated our approach on common medical benchmark problems [14], namely *Breast-Cancer*, *Diabetes*, *Heart*, and *Thyroid* preprocessed and partitioned as proposed in [31]. These are binary classification problems where the task is to predict whether a patient suffers from a certain disease or not. There are 100 partitions of each dataset into disjoint training and test sets. In [27], appropriate SVM hyperparameters C and a for Gaussian kernels with $\boldsymbol{A} = a\boldsymbol{I}$ are determined using a two-stage grid search. For each hyperparameter combination, five SVMs are constructed using the training sets of the first five data partitions and the average of the classification rates on the corresponding five test sets determines the score value of this parameter vector. The hyperparameter vector with the best score is selected and its performance is finally measured by calculating the score function using all 100 partitions.

We initialized our populations with the values found in [27] and used the same score function to determine the fitness. The results are shown in table 2. Except for one case, the scaled and rotated kernels led to significantly ($p < .05$) better results. There is another remarkable advantage of the scaled and rotated kernels: The number of support vectors—and therefore the execution time and storage complexity of the classifier—decreases. For a detailed description and additional results see [14].

4 Conclusions

Finding an appropriate neural system for a given task usually requires the solution of difficult optimization problems. These include adapting the hyperparameters of support vector machines or finding the right topology of a multi-layer perceptron network. Evolutionary algorithms (EAs) are particularly well suited for such kinds of tasks, especially when higher order optimization methods cannot be applied. We demonstrated three successful applications of evolutionary computation to the optimization of neural systems, where the EAs made use of deterministic strategy adaptation to improve the search performance.

Of course, it is appealing to use evolutionary methods for the design of neural networks, because this resembles the natural evolution of nervous systems. However, as our examples have shown, evolutionary methods should be regarded as the state-of-the-art choice for many neural network optimization tasks not because of the biological metaphor but because they are highly competitive approaches often superior to alternative methods.

References

[1] M. A. Arbib, editor. *The Handbook of Brain Theory and Neural Networks.* MIT Press, 2 edition, 2002.

[2] M. A. Arbib. Towards a neurally-inspired computer architecture. *Natural Computing*, 2(1):1–46,, 2003.

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

[4] H.-G. Beyer, H.-P. Schwefel, and I. Wegener. How to analyse evolutionary algorithms. *Theoretical Computer Science*, 287:101–130, 2002.

[5] C. M. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, 1995.

[6] O. Chapelle, V. Vapnik, O. Bousquet, and S. Mukherjee. Choosing multiple parameters for support vector machines. *Machine Learning*, 46(1):131–159, 2002.

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

[8] K.-M. Chung, W.-C. Kao, C.-L. Sun, and C.-J. Lin. Radius margin bounds for support vector machines with RBF kernel. *Neural Computation*, 15(11):2643–2681, 2003.

[9] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines and other kernel-based learning methods.* Cambridge University Press, 2000.

[10] L. Davis. Adapting operator probabilities in genetic algorithms. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms, ICGA'89*, pages 61–69, Fairfax, VA, USA, 1989. Morgan Kaufmann.

[11] A. E. Eiben, R. Hinterding, and Z. Michalewicz. Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 3(2):124–141, 1999.

[12] A. E. Eiben and G. Rudolph. Theory of evolutionary algorithms: A bird's eye view. *Theoretical Computer Science*, 229(1):3–9, 1999.

[13] T. Evgeniou, M. Pontil, and T. Poggio. Regularization networks and support vector machines. *Advances in Computational Mathematics*, 13:1–50, 2000.

[14] F. Friedrichs and C. Igel. Evolutionary tuning of multiple SVM parameters. In M. Verleysen, editor, *12th European Symposium on Artificial Neural Networks (ESANN 2004)*, pages 519–524. Evere, Belgium: d-side publications, 2004.

[15] C. Gold and P. Sollich. Model selection for support vector machine classification. *Neurocomputing*, 55(1-2):221–249, 2003.

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

[17] F. Gruau, D. Whitley, and L. Pyeatt. 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, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 81–89, Stanford University, CA, USA, 1996. MIT Press.

[18] N. Hansen and A. Ostermeier. 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)*, pages 650–654. Aachen, Germany: Verlag Mainz, Wissenschaftsverlag, 1997.

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

[20] M. Hüsken and C. Igel. Balancing learning and evolution. In W. B. Langdon, E. Cantu-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A. C. Schultz, J. F. Miller, E. Burke, and N. Jonoska, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2002)*, pages 391–398. Morgan Kaufmann, 2002.

[21] C. Igel. Neuroevolution for reinforcement learning using evolution strategies. In R. Sarker, R. Reynolds, H. Abbass, K. C. Tan, B. McKay, D. Essam, and T. Gedeon, editors, *Congress on Evolutionary Computation (CEC 2003)*, volume 4, pages 2588–2595. IEEE Press, 2003.

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

[23] C. Igel and M. Hüsken. Empirical evaluation of the improved Rprop learning algorithm. *Neurocomputing*, 50(C):105–123, 2003.

[24] C. Igel and M. Kreutz. Operator adaptation in evolutionary computation and its application to structure optimization of neural networks. *Neurocomputing*, 55(1-2):347–361, 2003.

[25] C. Igel and M. Toussaint. Neutrality and self-adaptation. *Natural Computing*, 2(2):117–132, 2003.

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

[27] P. Meinicke, T. Twellmann, and H. Ritter. Discriminative densities from maximum contrast estimation. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*, Cambridge, MA, 2002. MIT Press.

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

[29] S. Nolfi. Evolution and learning in neural networks. In M. A. Arbib, editor, *The Handbook of Brain Theory and Neural Networks*, pages 415–418. MIT Press, 2 edition, 2002.

[30] M. Patel, V. Honavar, and K. Balakrishnan, editors. *Advances in the Evolutionary Synthesis of Intelligent Agents*. MIT Press, 2001.

[31] G. Rätsch, T. Onoda, and K.-R. Müller. Soft margins for adaboost. *Machine Learning*, 42(3):287–32, 2001.

[32] I. Rechenberg. *Evolutionsstrategie '94*. Werkstatt Bionik und Evolutionstechnik. Frommann-Holzboog, Stuttgart, 1994.

[33] R. D. Reed and R. J. Marks II. *Neural Smithing*. MIT Press, 1999.

[34] M. Riedmiller. Advanced supervised learning in multi-layer perceptrons – From backpropagation to adaptive learning algorithms. *Computer Standards and Interfaces*, 16(5):265–278, 1994.

[35] H. A. Rowley, S. Baluja, and T. Kanade. Neural network-based face detection. *IEEE Transactions on Pattern Analysis and Machine Intelligenc*, 20(1):23–38, 1998.

[36] G. Rudolph. On correlated mutations in evolution strategies. In R. Männer and B. Manderick, editors, *Parallel Problem Solving from Nature 2 (PPSN II)*, pages 105–114. Elsevier, 1992.

[37] B. Schölkopf and A. J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, 2002.

[38] H.-P. Schwefel. *Evolution and Optimum Seeking*. Sixth-Generation Computer Technology Series. John Wiley & Sons, 1995.

[39] B. A. Sendhoff. *Evolution of Structures – Optimization of Artificial Neural Structures for Information Processing*. Shaker Verlag, Aachen, 1998.

[40] H. T. Siegelmann and E. D. Sontag. On the computational power of neural nets. *Journal of Computer and System Sciences*, 50(1):132–150, 1995.

[41] J. Šíma. Training a single sigmoidal neuron is hard. *Neural Computation*, 14:2709–2728, 2002.

[42] J. Šíma and P. Orponen. General-purpose computation with neural networks: A survey of complexity theoretix results. *Neural Computation*, 15(12):2727–2778, 2003.

[43] J. E. Smith and T. C. Fogarty. Operator and parameter adaptation in genetic algorithms. *Soft Computing*, 1(2):81–87, 1997.

[44] E. D. Sontag. Recurrent neural networks: Some systems-theoretic aspects. In M. Karny, K. Warwick, and V. Kurkova, editors, *Dealing with Complexity: A Neural Network Approach*, pages 1–12. Springer-Verlag, 1997.

[45] P. Stagge. *Strukturoptimierung rückgekoppelter neuronaler Netze*. Konzepte neuronaler Informationsverarbeitung. ibidem-Verlag, Stuttgart, 2001.

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

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

[48] V. N. Vapnik. *The Nature of Statistical Learning Theory.* Springer-Verlag, 1995.

[49] S. Wiegand, C. Igel, and U. Handmann. Evolutionary optimization of neural networks for face detection. In M. Verleysen, editor, *12th European Symposium on Artificial Neural Networks (ESANN 2004)*, pages 39–144. Evere, Belgium: d-side publications, 2004.

[50] A. Wieland. Evolving controls for unstable systems. In *Proceedings of the International Joint Conference on Neural Networks*, volume II, pages 667–673. IEEE Press, 1991.

[51] M.-H. Yang, D. J. Kriegman, and N. Ahuja. Detecting faces in images: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(1):34–58, 2002.

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

[53] G. P. Zhang. Neural Networks for Classification: A Survey. *IEEE Transactions on System, Man, and Cybernetics – Part C*, 30(4), 2000.

[54] A. A. Zhigljavsky. *Theory of global random search.* Kluwer Academic Publishers, 1991.

Institut für Neuroinformatik
Ruhr-Universität Bochum
D-44780 Bochum, Germany
Email address: christian.igel@neuroinformatik.rub.de