Genesis of Organic Computing Systems: Coupling Evolution and Learning

Christian Igel[1] and Bernhard Sendhoff[2]

[1] Institut für Neuroinformatik, Ruhr-Universität Bochum, Germany
[2] Honda Research Institute Europe GmbH, Germany

Abstract. Organic computing calls for efficient adaptive systems in which flexibility is not traded in against stability and robustness. Such systems have to be specialized in the sense that they are biased towards solving instances from certain problem classes, namely those problems they may face in their environment. Nervous systems are perfect examples. Their specialization stems from evolution and development. In organic computing, simulated evolutionary structure optimization can create artificial neural networks for particular environments. In this chapter, trends and recent results in combining evolutionary and neural computation are reviewed. The emphasis is put on the influence of evolution and development on the structure of neural systems. It is demonstrated how neural structures can be evolved that efficiently learn solutions for problems from a particular problem class. Simple examples of systems that "learn to learn" as well as technical solutions for the design of turbomachinery components are presented.

1 Introduction

Technical systems that continuously adapt to a changing natural environment thereby acting (quasi-) autonomously have not been designed so far. There are several fundamental challenges that have to be met. First, more flexibility has to be realized on the software and possibly even on the hardware level. Second, this flexibility must not be traded in against system stability and robustness. Minimal performance must be guaranteed under all circumstances and degradation must be gradual and controlled. Third, the system must be expandable and sustainable.

Biological neural systems are able to fulfill all of these properties while their technical counterparts do not yet meet these requirements. Nevertheless, we believe that artificial neural networks (NNs) provide a computing paradigm whose potential has not yet been fully exploited. Our approach to address the above-mentioned challenges and to tap the potential of artificial neural systems is to tune them towards particular classes of problems and to particular patterns of processing.

In nature, such a specialization stems from evolution and development. Both processes design and shape structures which are ready to accommodate learning and self-organizing processes, which we see as the driving forces behind the capability of neural systems, see Fig. 1. We think that understanding the biological “design techniques” for nervous systems—evolution, development, and learning—paves the way for the design of human competitive artificial adaptive systems.

When designing adaptive systems, appropriate specialization (bias) and invariance properties are important, partially conflicting objectives. The “No-free-lunch theorems” for learning and optimization imply that it is fruitless to try to build universal adaptive systems. The systems have to be biased towards particular problem classes. This bias can be induced by evolved structures on which learning and self-organizing processes operate. In this chapter, we therefore review trends and recent results in combining evolutionary and neural computation. We will highlight synergies between the two fields beyond the typically standard examples and emphasize the influence of evolution and development on the structure of neural systems for the purpose of adaption. We demonstrate how neural structures can be evolved that efficiently learn particular problem classes. We present simple examples of systems that “learn to learn” as well as technical solutions for the design of turbomachinery components.

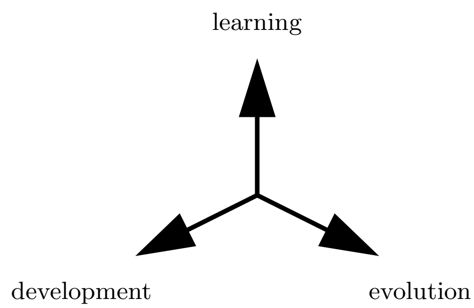

Fig. 1. Dimensions of natural design.

The next section provides some background in artificial NNs and evolutionary algorithms (EAs). We put an emphasis on theoretical limitations and perspectives of these computing paradigms. We briefly describe simple NNs based on integrate and fire neurons, introduce EAs in the framework of stochastic search, and summarize the No-free-lunch theorems for learning and optimization. In section 3, we discuss evolutionary structure optimization of neural systems and review some more recent trends in combining EAs and neural systems. Finally, we demonstrate how neural structures can be evolved that efficiently learn solutions for problems from a particular problem class.

2 Background

In this section, we provide short introductions to NNs and EAs and review “No-free-lunch” results for learning and optimization.

2.1 Neural Computation

In the following, we briefly introduce basic ideas of NNs on the basis of the simple *rate-code leaky integrator neuron* model. A detailed introduction to the broad field of neural computation is far beyond the scope of this article. A good starting point for reading is [1], recommendable introductory books on NNs for technical applications are [2–4] and on modeling nervous systems [5].

Neural systems can be described on different levels of abstraction. Many models, including those usually adopted for technical applications, can be derived from the leaky integrator neuron. This model is based on the assumption that the basic units of computation in nervous systems are single neurons. A model neuron i is situated in time t and its state is described by the *membrane potential* $u_i(t)$ governed by the differential equation

$$\tau_i \frac{\partial u_i(t)}{\partial t} = -u_i(t) + \sum_j w_{ij} \sigma_j[u_j(t)] + \sum_k w'_{ik} s_k(t) + \theta_i$$

with time constant τ_i. The neuron computes a weighted linear sum of the inputs it receives (see [6] for a review of more detailed models of single neurons). The first sum runs over all neurons j providing input to i, the second over all external inputs $s_k(t)$, which are gathered in the vector $\boldsymbol{s}(t)$, to the system. The weights w_{ij} and w'_{ik} describe the strengths of the synaptic connections. In the absence of input the membrane potential relaxes to the resting level θ_i. It is assumed that the communication in a network of these units is realized only through spikes traveling between the neurons. A neuron emits a spike when its membrane potential exceeds a certain threshold. Real spikes are discrete events, but in the model a rate code describing the average spiking frequency is assumed to capture the essence of the signals. This rate can be viewed as an ensemble average across a population of neurons with the same properties, or as the frequency of spikes of a single neuron in some time interval. The activation function σ_i, which is usually sigmoidal (i.e., nondecreasing and bounded), maps the membrane potential u_i to the corresponding spiking frequency. Forward Euler approximation, $\partial u_i(t)/\partial t \approx (u_i(t + \Delta t) - u_i(t))/\Delta t$, with $\Delta t = \tau_i = 1$, leads to the basic discrete-time equation $u_i(t + 1) = \sum_j w_{ij} \sigma_j[u_j(t)] + \sum_k w'_{ik} s_k(t) + \theta_i$.

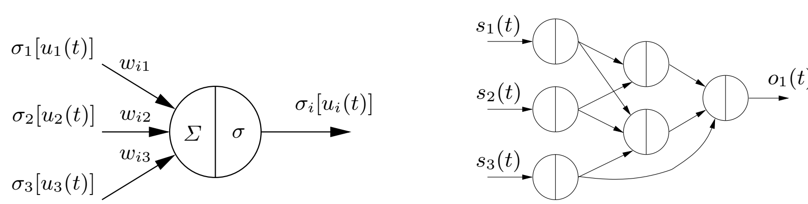

Fig. 2. Simple computational model of a single neuron, left, and a neural network graph, right.

The structure or architecture of the NN can be described by a graph in which the nodes correspond to the neurons and there is an edge from i to j if neuron j gets input from neuron i. If the network graph contains no cycles, we speak of a feed-forward NN. If we number the neurons such that node i only receives input from units j with $j < i$ and update the neurons in increasing order, the discrete-time network equation can be written as a static function $u_i(\boldsymbol{s}) = \sum_{j<i} w_{ij}\sigma_j[u_j] + \sum_k w'_{ik}s_k + \theta_i$. Often, some of the neurons are dedicated *output neurons* whose spike rates are gathered in the vector $\boldsymbol{o}(t)$ and the neural system can be viewed as a functional mapping input sequences $\boldsymbol{s}(t)$ to output sequences $\boldsymbol{o}(t)$. In case of a feed-forward NN, the mapping reduces to a function assigning an output $\boldsymbol{o}$ to an input $\boldsymbol{s}$, see Fig. 2.

Models of NNs based on leaky integrator neurons, in principle, exhibit universal approximation and computation properties under mild assumptions (e.g., see [7–9]). However, the general question of how to design an appropriate neural system efficiently for a given task remains open and complexity theory reveals the need for using heuristics (e.g., see [10])—here these heuristics are the major organization principles of biological NNs, evolution, development, and learning.

Supervised learning of an NN means adapting the weights w_{ij}, w'_{ik} such that given some input $\boldsymbol{s}(t)$ the output neurons show a predefined behavior $\boldsymbol{y}(t)$, which is described by sample (training) input-output sequences. A feed-forward NN learns a static function h based on sample input-output patterns $\{(\boldsymbol{s}_1, \boldsymbol{y}_1), \dots, (\boldsymbol{s}_\ell, \boldsymbol{y}_\ell)\}$. This is usually done by gradient-based minimization of the (squared) differences between the targets $\boldsymbol{y}_i$ and the corresponding outputs $\boldsymbol{o}_i$ of the NN given the input $\boldsymbol{s}_i$. The ultimate goal is not to simply memorize the training patterns, but to find a statistical model for the underlying relationship between input and output data. Such a model will generalize well, that is, it will make good predictions for cases other than the training patterns. Therefore, a critical issue is 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 [11].

In the context of feed-forward NNs, generalization can for example be formalized in the framework of statistical learning theory as follows. Let the goal be learning a function from some input space S to some output space Y and let $h : S \to Y$ be the function realized by the NN. Based on some input-output patterns drawn identically independently distributed according to the distribution P on $S \times Y$, the goal of generalization is to minimize $\int_{S\times Y} P(\boldsymbol{s}, \boldsymbol{y}) L(h(\boldsymbol{s}), \boldsymbol{y}) \mathrm{d}\boldsymbol{s}\, \mathrm{d}\boldsymbol{y}$, where $L : Y \times Y \to \mathbb{R}_0^+$ denotes a loss function. The distribution P is usually unknown and defines the learning problem at hand. The value $L(a, b)$ quantifies the cost or regret of predicting a instead of b and returns zero if its arguments are equal. For example when learning a one-dimensional real-valued function, $S = Y = \mathbb{R}$ and $L(a, b) = (a - b)^2$ is a typical choice.

In this chapter, we focus on the architecture of feed-forward neural networks, however, most of our findings and discussions apply equally well to recurrent neural systems, which also have been used successfully in applications in the past (in particular for time series prediction tasks, e.g., [12, 13]).

2.2 Evolutionary Computation

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

① Choose a joint probability distribution $P^{(1)}_{\mathcal{G}^\lambda}$ 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 whether some stopping condition is reached; 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 (parametrize) 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 Fig. 3. 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 $\tilde{\boldsymbol{g}}^{(t)}_1, \dots, \tilde{\boldsymbol{g}}^{(t)}_\mu$. This is called *mating selection* and can be deterministic or stochastic (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 $\mathcal{G}^l$, $1 \le k \le \mu, 1 \le l \le \lambda$. 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 $\boldsymbol{g}^{(t)}_i$ can be the product of applying recombination $o_{\text{rec}} : \mathcal{G}^2 \to \mathcal{G}$ to two randomly selected parents $\tilde{\boldsymbol{g}}^{(t)}_{i_1}$ and $\tilde{\boldsymbol{g}}^{(t)}_{i_2}$ followed by mutation $o_{\text{mut}} : \mathcal{G} \to \mathcal{G}$, that is, $\boldsymbol{g}^{(t)}_i = o_{\text{mut}}\left(o_{\text{rec}}\left(\tilde{\boldsymbol{g}}^{(t)}_{i_1}, \tilde{\boldsymbol{g}}^{(t)}_{i_2}\right)\right)$. 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 $P^{(t)}_{\mathcal{G}^\lambda}(\boldsymbol{g}^{(t)}_1, \dots, \boldsymbol{g}^{(t)}_\lambda) = P_{\mathcal{G}^\lambda}\left(\boldsymbol{g}^{(t)}_1, \dots, \boldsymbol{g}^{(t)}_\lambda \,|\, \tilde{\boldsymbol{g}}^{(t)}_1, \dots, \tilde{\boldsymbol{g}}^{(t)}_\mu; \boldsymbol{\theta}^{(t)}\right)$ be the probability that parents $\tilde{\boldsymbol{g}}^{(t)}_1, \dots, \tilde{\boldsymbol{g}}^{(t)}_\mu$ create offspring $\boldsymbol{g}^{(t)}_1, \dots, \boldsymbol{g}^{(t)}_\lambda$. This distribution is additionally parametrized by some *external strategy parameters* $\boldsymbol{\theta}^{(t)}$, 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 $P^{(t)}_{\mathcal{G}^\lambda}$ can be factorized as $P^{(t)}_{\mathcal{G}^\lambda}(\boldsymbol{g}^{(t)}_1, \dots, \boldsymbol{g}^{(t)}_\lambda) = P^{(t)}_{\mathcal{G}}(\boldsymbol{g}^{(t)}_1) \cdot \ldots \cdot P^{(t)}_{\mathcal{G}}(\boldsymbol{g}^{(t)}_\lambda)$.

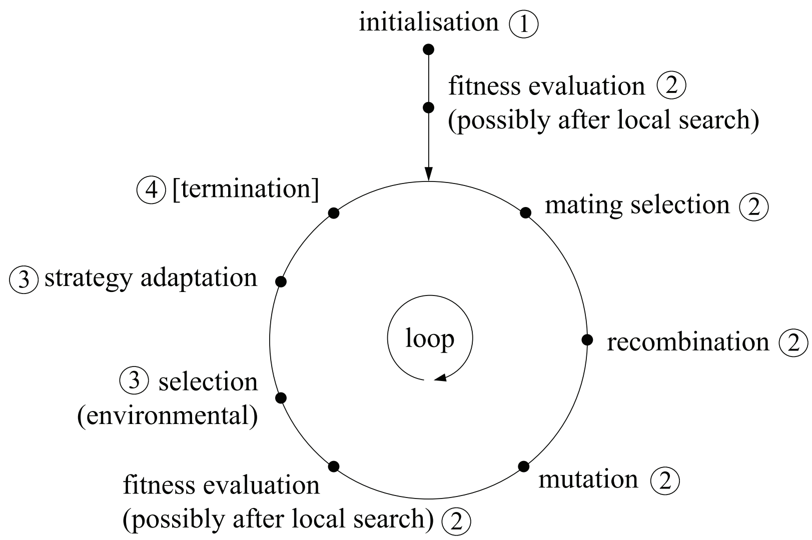

Fig. 3. Basic EA loop. The numbers indicate the corresponding steps in the random search scheme. When optimizing adaptive systems, the local search usually corresponds to some learning process.

Evaluation of an individual corresponds to determining its fitness by assigning the corresponding cost value given by the quality function f. Evolutionary algorithms can—in principle—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. In single-objective optimization, we usually have $\mathcal{Y} \subset \mathbb{R}$, whereas in multi-objective optimization, see section 3.2, vector-valued functions (e.g., $\mathcal{Y} \subset \mathbb{R}^k, k > 1$) are considered. In co-evolution (see section 3.2), individuals interact in a way so that they affect one another's adaptations. Therefore, the fitness values are not determined for each individual in isolation, but in the context of the current population (i.e., $f : \mathcal{G}^\lambda \to \mathcal{Y}^\lambda$ or even $f : \mathcal{G}^{\lambda+\mu} \to \mathcal{Y}^{\lambda+\mu}$ if the parents are also involved in the fitness calculation). The interaction of the individuals may be competitive or cooperative. As the fitness function is not fixed, co-evolution allows for a "bootstrapping" of the evolutionary process and "open ended" evolution.

Updating the search distribution corresponds to *environmental selection* and sometimes additional *strategy adaptation* of external strategy parameters $\boldsymbol{\theta}^{(t+1)}$. The later is extensively discussed in the context of optimization of NNs in [14, 15]. 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.

It is often argued that evolutionary optimization is theoretically not well understood—ignoring the tremendous progress in EA theory during the last years. Although there are only a few results for general settings (e.g., convergence [16]), there exist rigorous expected runtime analyzes of simplified algorithms on restricted, but important classes of optimization problems, see [17, 18] and references therein. The article [19] provides good starting points for reading about EA theory.

2.3 The Need for Specialization: No-Free-Lunch

It is not only intuitive, but also provable that it is in general not possible to design an universal adaptive system that outperforms other systems across all possible problems. This is formally expressed by the No-free-lunch (NFL) theorems going back to the work of Wolpert and Macready [20, 21]. Coarsely speaking, the NFL theorems for learning state that if there is no assumption how the past (training data) is related to the future (test data), prediction is impossible. In other words, if there is no a priori restriction on the possible phenomena that are expected, it is impossible to generalize and thus no algorithm is superior to another. Even worse, any consistent algorithm (i.e., any algorithm converging to the Bayes optimal classifier almost surely when the number of training patterns, drawn independently from the distribution describing the problem, approaches infinity) can have arbitrarily poor behavior when given a finite, incomplete training set [20, 22, 23].

These results carry over to general search and optimization algorithms. The NFL theorem for optimization formalizes that averaged over the set $\mathcal{F}$ of all possible objective functions defined between a finite search space $\mathcal{X}$ and a finite set $\mathcal{Y}$ of cost values all optimization algorithms have the same performance. It is assumed that the algorithms never visit a search point twice and that the performance measure just depends on the objective function values of the visited search points [18, 21, 24–27]. More general, the following holds for any probability distribution P over $\mathcal{F}$. If and only if $\mathcal{F} = \bigcup_i \mathcal{F}_i$, every $\mathcal{F}_i$ is closed under permutation, and $f, g \in F_i$ implies that f and g have the same probability $P(f) = P(g)$ to be the objective function, then all optimization algorithms have the same performance averaged over $\mathcal{F}$ w.r.t. P [26]. Closure under permutation of a set $\mathcal{F}_i$ means that for every bijective function $\pi : \mathcal{X} \to \mathcal{X}$ it holds that $f \in \mathcal{F}_i$ implies $f \circ \pi \in \mathcal{F}_i$. These assumptions for an NFL result to hold are rather strict, and fortunately problem classes relevant in practice are likely to violate these assumptions [25, 26].

Nonetheless, only if we consider restricted problem classes, in the sense that the assumptions of the NFL theorems are not fulfilled, we can design efficient adaptive systems. It is important to make this bias towards a problem class explicit in the design process. In nature, such a bias stems from the evolved structures on which learning and self-organizing processes operate. In organic computing, simulated evolutionary structure optimization can create systems biased towards relevant problem classes.

3 Evolutionary Computation and Neural Systems

Artificial evolutionary and artificial neural systems have a long history, which in many respects resembles each other. In their beginnings, they were both met with considerable skepticism from both the biological as well as the technological world. For the one community their simplifications and abstractions meant throwing over board years of carefully accumulated details on how biological systems operate, develop, learn, and evolve. For the other community the new type of distributed, stochastic, and nonlinear processing was equally hard to accept. During their maturation both fields met a couple of times, but not as often as one might expect bearing in mind that their philosophy to extract principles of biological information processing and apply them to technical systems is so similar.

Although not directly aimed at the formation of neural systems, the design of intelligent automata was among the earliest applications of EAs and may be traced back to the 50s, see [28]. However, it took another 30 years until first papers were published describing explicitly the application of EAs to NNs and in this context—albeit more hesitantly—to learning [29, 30]. The subject quickly received considerable interest and several papers were published in the early nineties concentrating on both the optimization of the network architecture and its weights. Although nowadays NNs and EAs are used frequently and successfully together in a variety of applications, the real breakthrough, that is, the evolution of neural systems showing *qualitatively* new behavior, has not been reached yet. The complexity barrier might have been pushed along but it has not been broken down. Nevertheless, many important questions on the architecture, (e.g., modularity) the nature of learning, (e.g., nature vs. nurture) and the development of neural systems (e.g., interaction between levels of adaptation) have been raised and important results have been obtained.

There are still only few works on connecting current research on brain science with evolutionary computation, however, first promising attempts have been made, as we will see in section 3.2. Here, on a more general note, we would like to argue that combining evolutionary development with brain science is more than just optimizing models of biological neural systems. The brain is a result of the past as much as of the present. That means that learning (the present) can only operate on an appropriate structure (the past). The current structure reflects its history as much as its functionality. The flexibility and adaptability of the brain is based on its structural organization which is the result of its ontogenetic development. The brain is not one design but many designs; it is like a cathedral where many different parts have been added and removed over the centuries. However, not all designs are capable of such continuous changes and the fact that the brain is, is deeply rooted in its structural organization.

In this section, we discussed some selected aspects of combining neural and evolutionary computing. More comprehensive surveys, all having slightly different focuses, can be found in [31–34].

3.1 Structure Optimization of Adaptive Systems

Although NNs are successfully applied to support evolutionary computation (see section 4.2), the most prominent combination of EAs and NNs is evolutionary optimization of neural systems.

In general, 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 sample data driving the adaptation. We define the *structure* as those parts of the system that cannot be changed by the learning/self-adaptation algorithm. Given an adaptation rule $\mathcal{A}$, the structure $\mathcal{S}$ determines

- the set of solutions that can be realized,
- how solution changes given new stimuli/signals/data, partial failure, noise, etc.,
- the neighborhood of solutions (i.e., distances in solution space),
- bias (specialization) and invariance properties.

Learning of an adaptive system can be defined as goal-directed, data-driven changing of its behavior. Examples of learning algorithms for technical NNs include gradient-based heuristics (see section 2.1) or quadratic program solvers. Such "classical" optimization methods are usually considerably faster than pure evolutionary optimization of these parameters, 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 (e.g., see section 3.2). Then EAs for real-valued optimization provide a means for adjusting the NN parameters. 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" [35]. 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 the search for an appropriate topology for a multi-layer perceptron NN, see [36–38] for some real-world applications. Here, the search space ultimately consists of graphs, see section 2.1. When using EAs to design NN graphs, the key questions are how to encode the topologies and how to define variation operators that act on this representation. In terms of section 2.2, operators and representation both determine the search distribution and thereby the neighborhood of NNs in the search space. Often an intermediate space, the phenotype space $\mathcal{P}$, is introduced in order to facilitate the analysis of the problem and of the optimization process itself. The fitness function can then be written as $f = f' \circ \phi$, where $\phi : \mathcal{G} \to \mathcal{P}$ and $f' : \mathcal{P} \to \mathcal{Y}$. The definition of the phenotype space is to a certain degree arbitrary. This freedom in the definition of the phenotype space equally exists in evolutionary biology [39] and is not restricted to EAs. The probability that a certain phenotype $p \in \mathcal{P}$ is created from a population of phenotypes strongly depends on the representation and the variation operators. When the genotype-phenotype mapping ϕ is not injective, we speak of neutrality, which may considerably influence the evolutionary process (see [40] for an example in the context of NNs). We assume that $\mathcal{P}$ is equipped with an extrinsic (i.e., independent of the evolutionary process) metric or at least a consistent neighborhood measure, which *may* be defined in relation to the function of the individual. In the case of NNs, the phenotype space is often simply the space of all possible connection matrices of the networks. Representations for evolutionary structure optimization of NNs have often been classified in "direct" and "indirect" encodings. Coarsely speaking, a direct encoding or representation is one where (intrinsic) neighbourhood relations in the genotype space (induced by $P_{\mathcal{G}^\lambda}$) broadly correspond to extrinsic distances of the corresponding phenotypes. Note that such a classification only makes sense once a phenotype space with an extrinsic distance measure has been defined and that it is only valid for this particular definition (this point has frequently been overlooked because of the implicit agreement on the definition of the phenotype space, e.g., the graph space equipped with a graph editing distance). This does not imply that both spaces are identical. In an indirect encoding the genotype usually encodes a rule, a program or a mapping to build, grow or develop the phenotype. Such encoding foster the design of large, modular systems. Examples can be found in [41–43, 33, 44].

3.2 Trends in Combining EAs and Neural Computation

In the following, we review some more recent trends in combining neural and evolutionary computing. Needless to say that such a collection is a subjective, biased selection.

Multi-objective Optimization of Neural Networks Designing a neural system usually requires optimization of several, often conflicting objectives. This includes coping with the bias-variance dilemma or trading off speed of classification vs. accuracy in real-time applications. Although the design of neural systems is obviously a multi-objective problem, it is usually tackled by aggregating the objectives into a scalar function and applying standard methods to the resulting single-objective task. However, this approach will in general not find all desired solutions [45]. Furthermore, the aggregation weights have to be chosen correctly in order to obtain the desired result. In practice, it is more convenient to make the trade-offs between the objectives explicit (e.g., visualize them) after the design process and select from a diverse set of systems the one that seems to be most appropriate. This can be realized by "true" multi-objective optimization (MOO, [46]). The MOO algorithms approximate the set of Pareto-optimal trade-offs, that is, those solutions that cannot be improved in any objective without getting worse in at least one other objective. From the resulting set of systems the final solution can be selected after optimization. There have been considerable advances in MOO recently, which can now be incorporated into machine learning techniques. In particular, it was realized that EAs are very well suited for multi-criterion optimization and they have become the MOO methods of choice in the last years [47, 48]. Recent applications of evolutionary MOO to neural systems address the design of multi-layer perceptron NNs [49–51, 36, 38, 52, 46] and support vector machines (SVMs) [53, 54].

Reinforcement Learning In the standard reinforcement learning (RL, [55–57]) scenario, an agent perceives stimuli from the environment and decides based on its policy which action to take. Influenced by the actions, the environment changes its state and possibly emits reward signals. The reward feedback may be sparse, unspecific, and delayed. The goal of the agent is to adapt its policy, which may be represented by (or be based on) a NN, such that the expected reward is maximized. The gradient of the performance measure with respect to NN parameters can usually not be computed (but approximated in case of stochastic policies, e.g., see [58, 59]).

Evolutionary algorithms have proved to be powerful and competitive methods for solving RL problems [60–62]. The recent success of evolved NNs in game playing [63–66] demonstrates the potential of combining NNs and evolutionary computation for RL. The possible advantages of EAs compared to standard RL methods are that they allow—in contrast to the common temporal difference learning methods—for direct search in the space of (stochastic as well as deterministic) policies. Furthermore, they are often easier to apply and are more robust with respect to the tuning of the meta-parameters (learning rates, etc.). They can be applied if the function approximators are non-differentiable, and can also optimize the underlying structure of the function approximators.

Closely related is the research area of evolutionary robotics devoted to the evolution of "embodied" neural control systems [67–70]. Here promising applications of the principle of co-evolution can be found.

Evolving Network Ensembles Ensembles of NNs that together solve a given task can be preferable to monolithic systems. For example, they may allow for a task decomposition that is necessary for efficiently solving a complex problem and they are often easier to interpret [71]. The population concept in EAs appears to be ideal for designing neural network ensembles, as, for example, demonstrated for classification tasks in [72, 52]. In the framework of decision making and games, Mark et al. [73] developed a combination of NN ensembles and evolutionary computation. Two ensembles are used to predict the opponents strategy and to optimize the own action. Using an ensemble instead of a single network ensures to be able to maintain different opponent experts and counter-strategies in parallel. The EA is used to determine the optimal input for the two network ensembles. Ensembles of networks have also been shown to be a superior alternative to single NNs for fitness approximation in evolutionary optimization. In [74] network ensembles have been optimized with evolution strategies and then used in an evolutionary computation framework as meta-models. Besides the increase in approximation quality an ensemble of networks has the advantage that the fidelity of the networks can be estimated based on the variance of the ensemble.

Optimizing Kernel Methods Adopting the extended definition of structure as that part of the adaptive system that cannot be optimized by the learning algorithm itself, model selection of kernel-based methods is a structure optimization problem. For example, choosing the right kernel for a SVM [75–77] is important for its performance. When a parametrized family of kernel functions is considered, kernel adaptation reduces to finding an appropriate parameter vector. These "hyperparameters" are usually determined by grid search, which is only suitable for the adjustment of very few parameters, or by gradient-based approaches. When applicable, the latter methods are highly efficient albeit susceptible to local optima. Still, often the gradient of the performance criterion w.r.t. the hyperparameters can neither be computed nor accurately approximated. Therefore, there is a growing interest in applying EAs to model selection of SVMs. In [78, 79, 53, 54], evolution strategies (i.e., EAs tailored for real-valued optimization) were proposed for adapting SVM hyperparameters, in [80, 81] genetic algorithms (EAs that represent candidate solutions as fixed-length strings over a finite alphabet) were used for SVM feature selection.

Computational Neuroscience and Brain-inspired Architectures There are only a few applications of evolutionary computation in brain science [82–86], although evolutionary "analysis by synthesis" guided by neurobiological knowledge may be a powerful tool in computational neuroscience. The challenge is to force artificial evolution to favor solutions that are reasonable from the biological point of view by incorporating as much neurobiological knowledge as possible in the design process (e.g., by a deliberate choice of the basic system structure and constraints that ensure biological plausibility).

In the field of brain inspired vision systems [87, 88] evolutionary algorithms have been used to optimize the structure of the system (i.e., feature banks or hierarchical layers) and to determine a wide variety of parameters. Evolutionary algorithms have been successfully applied to the Neocognitron structure [89–91], which was one of the first hierarchical vision systems based on the structure of its biological counterpart [87]. More recent work employed evolution strategies to optimize the nonlinearities and the structure of a biologically inspired vision network, which is capable of performing a complex 3D real world object classification task [92, 93]. The authors used a evolutionary optimization with direct encoding that was capable of performing well in a 1800-dimensional search space. In a second experiment evolutionary optimization was successfully combined with local unsupervised learning based on a sparse representation. The resulting architecture outperformed alternative approaches.

4 Networks that Learn to Learn

The ability to learn (online) is one of the most distinguishing features of artificial NNs. The idea behind the "learn to learn" concept discussed in this section is that the target of the evolutionary structure optimization of NNs should be the ability to efficiently learn new related problems during operation time, see [94, 95]. Here "efficient" means fast and based on only few data. The term "new related problems" is more difficult to define. The problems must have some common structure that can be captured by the EA and reflected in the NN architecture. Learning a different problem class goes beyond standard generalization. The latter means generalizing from a finite set of training samples to arbitrary samples drawn from the same distribution as, for example, formally defined at the end of section 2.1. Facing a different problem from the same class means that the underlying distribution has changed while belonging to the set of distributions which define the class and which have some common features that can be represented by the structure.

Therefore we speak of "second order generalization" for being able to efficiently switch between problems, see Fig. 4 (left). In the notation introduced by Thrun and Pratt [96], this ability belongs to the area of representations and functional decompositions. However, in the evolutionary approach, this functional decomposition is self-organized during the evolutionary process. There are basically two different ways in which second order generalization can be achieved and used: the parallel and the sequential way. In Fig. 4 (right, a) the standard approach to learn one problem with an NN is shown. In part (b), the parallel approach is shown. The network is optimized during evolution in order to learn one of a number of possible problems. The actual decision is made after the network's structure has been fixed by the evolutionary search. However, during evolutionary search the network's structure must be optimized in order to cope with any of the possible problems. Thus, each structure is applied to all problems (or a random subset of problems of the respective class). For each problem the weights are newly initialised. The fitness of the network is determined by the mean (or median or weighted sum) of the network's individual performances. In Fig. 4 (right, c), the network has to learn a number of problems one after the other during operation time. The network's structure has been optimized in such a way that switching from problem to problem can be achieved most efficiently in the above sense. The weights are not randomly initialized (like in (b)), but an averaged Lamarckian inheritance [97] is used for exploiting information on previous problems for the next problem belonging to the same class. Again the

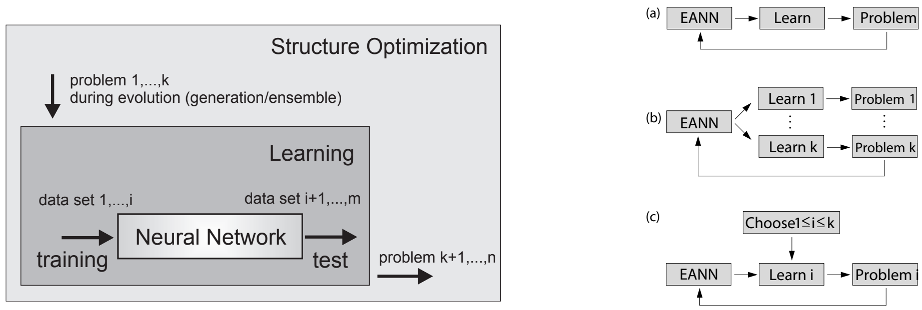

Fig. 4. Evolutionary structure optimization for problem classes. Left: Methods to achieve first and second order generalization in neural network learning and evolution (structure optimization). Right: Standard neural network learning (a), parallel switching between problems (b) and sequential switching (c). EANN denotes a neural network optimized with an evolutionary algorithm.

fitness of the network is determined by the mean (or median or weighted sum) of the network's individual performances.

From the NFL theorems (see section 2.3) we conclude that adaptive systems have to be specialized towards a particular problem class to show above average performance aross this class. Second order generalization can be viewed as such a specialization.

4.1 Modularity

A simple example of how to build NNs that "learn to learn" was given in the study by Hüsken et al. [98], where NN architectures were evolved that quickly learn instances from a particular problem class. Modular structures were designed which were biased towards separable problems.

Among others, Hüsken et al. considered feed-forward NNs that had to learn binary mappings $\{0,1\}^6 \rightarrow \{0,1\}^2$ assigning target values $\boldsymbol{y} = (y_1, y_2)' \in \{0,1\}^2$ to inputs $\boldsymbol{s} = (s_1, \dots, s_6)' \in \{0,1\}^6$. The class of mappings was restricted to those which are separable in the strict sense that y_1 does only depend on the inputs $s_1, \dots, s_3$ and y_2 only on $s_4, \dots, s_6$. The mappings changed over time and a simple EA was employed to create feed-forward network structures that quickly learn a new instance of the problem class. The fitness of an NN structure was determined by the time needed to learn a randomly chosen problem instance, that is, the sequential switching approach depicted in Fig. 4 was used.

After a few generations, the networks adapted to the special, restricted problem class and the learning time drastically decreased, see Fig. 5. In this toy example, it is obvious that NN structures that are modular in the sense that they process the inputs $s_1, \dots, s_3$ and $s_4, \dots, s_6$ separately without interference are advantageous. When measuring this special kind of modularity during the course

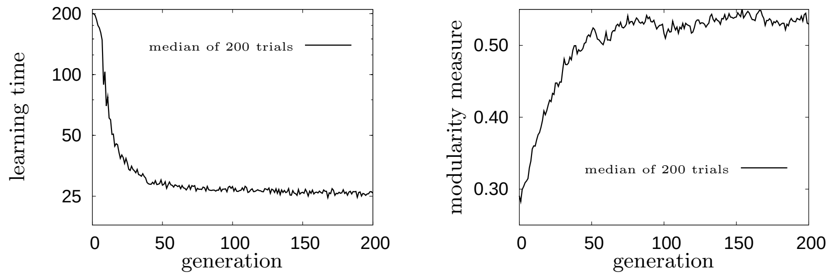

Fig. 5. Evolving networks that "learn to learn." During evolution the network structures adapt to a special problem class. This specialization leads to a reduced time for learning a new instance of the problem class, see left plot. Since the class in this example consists of separable problems, the degree of modularity of the network structures increases over time, as shown in the right plot (cf. [98]).

of evolution, it turned out that the modularity indeed increased, see Fig. 5, right plot.

In [99] modularity is analyzed in the context of problem decomposition and a novel modular network architecture is presented. Modularity is related to multi-network systems or ensembles for which a taxonomy is presented. A co-evolutionary framework is used to design modular NNs. The model consists of two populations. The first population is made up of a pool of modules and the second population synthesizes complete systems by drawing elements from the pool of modules. In this framework, modules represent parts of the solution which co-operate with each other to form a complete solution. Using two artificial tasks the authors demonstrate that modular neural systems can be co-evolved. At the same time, the usefulness of modularity depends on the learning algorithm and the quality function.

4.2 Real-world Application

Evolutionary algorithms combined with computational fluid-dynamics (CFD) have been applied successfully to a large variety of design optimization problems in engineering (e.g., [100–102]). The fluid-dynamics simulations that are necessary to determine the quality of each design are usually computationally expensive, for example the calculation of the three-dimensional flow field around a car takes between 10-30 hours depending on the required accuracy. Therefore, meta-models or surrogates are used during the search to approximate the results of the CFD simulations. Although first approaches to combine fitness approximation with EAs are relatively old [103], it is only in the last couple of years that the field has received wider attention, see [104] for a review. It has been revealed that the strategy to keep the update of the meta-model and the optimization process separate is not advisable, since the optimization is easily misled if the modeling quality is limited (which is often the case in practical applications). Jin et al. [105] have suggested to use the meta-model alongside the true objective function to guarantee correct convergence. Furthermore, the use of NNs as models is particularly advantageous because of their online learning ability. Thus, the approximation quality of NNs can continuously be improved during the optimization process when new CFD data is available (e.g, [105–108]).

It is interesting to note that the standard mean squared error measure of NNs is not necessarily the best means to determine the quality of NNs that are employed as surrogates. Figure 6 shows why this is the case. During evolutionary search, the absolute error of the NN is of no concern, as long as the model is able to distinguish between "good" and "bad" individuals.

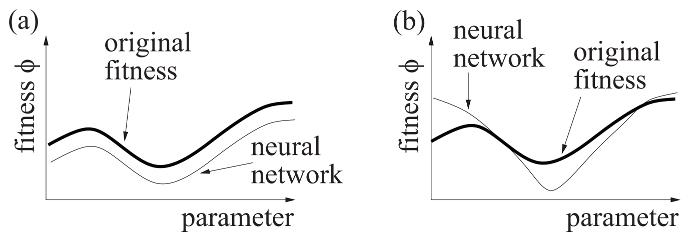

Fig. 6. Although the approximation errors of the neural network models are quite high, the optimization based on the approximation models leads to the desired minimum of the fitness under rank-based selection.

Evolution of the meta-model Neural networks that are used as meta-models during evolutionary search should have the best possible architecture for the approximation task. Therefore, EAs are employed to determine the structure of the networks offline, for example using data from previous optimization tasks. Weight adaptation is conducted during the evolutionary design optimization whenever new data are available.

This framework has been employed in [97] for the optimization of turbine blades of a gas turbine engine. The flow field around a turbine blade and the engine are shown in Fig. 7. Navier-Stokes equations with the (k-ϵ) turbulence model were used for the two dimensional CFD simulations. During optimization the pressure loss was minimized subject to a number of geometrical and functional constraints in particular the target outflow angle α was set to 69.70 deg. The turbine blades were represented by 26 control points of non-uniform rational B-splines. The (x, y)-coordinates of the control points were optimized using a (2,11)-evolution strategy, further details can be found in [97]. The results of the optimization are given in Fig. 8. The average pressure loss and outflow angle are shown that have been reached in the evolutionary design optimization of the turbine blade. The three curves represent three different strategies to define the

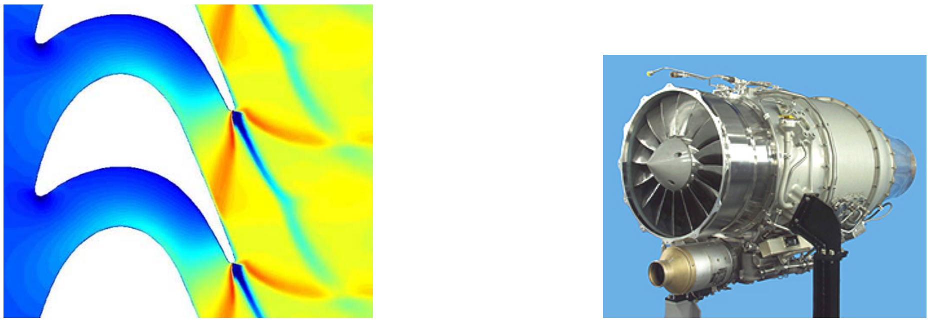

Fig. 7. The flow field of a turbine blade cascade for a gas turbine engine shown on the right.

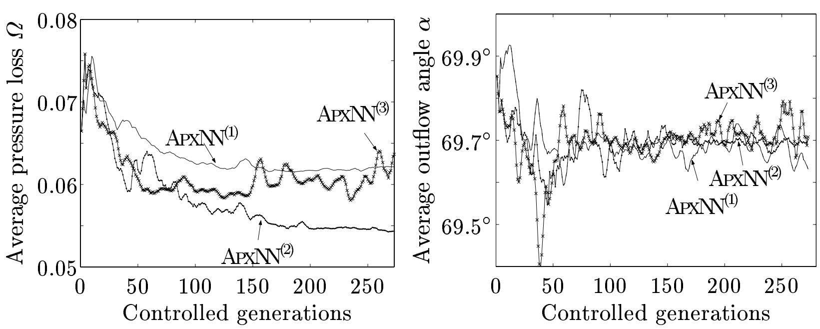

Fig. 8. Results normalized to the number of generations where the CFD simulations have been used. ApxNN$^{(1)}$ denotes a fully connected neural network, for ApxNN$^{(2)}$ the network structure has been evolutionary optimized and for ApxNN$^{(3)}$ the network has been optimized to switch between different design domains (problem classes) most efficiently.

architecture of the NN that has been used as a meta-model during search. The model of the first type (ApxNN$^{(1)}$) use a fully connected architecture. The weights are initialized by means of offline learning, using a number of given training data collected in a comparable blade optimization trial (e.g., different initialization but the same number of control points of the spline and the same fitness function). The second type of network model (ApxNN$^{(2)}$) was optimized offline with an EA using data that was generated in a previous optimization run. The third approach will be discussed in the next section. It is evident that the evolutionary optimized NN structure clearly outperforms the fully connected model in the practical application.

***Learn* Surrogates to *Learn* CFD** We already discussed the idea to evolve the architecture of NNs not just for one specific problem but instead to optimize the network so that it is able to quickly adapt to problems belonging to one class. We can transfer this idea to the problem domain of surrogates for approximation during evolutionary design optimization by sub-dividing the CFD samples into groups (problems) belonging to one and the same class namely the approximation of CFD data for evolutionary search. This is a reasonable approach because we do not expect to evolve a network that approximates the CFD results well for the whole optimization. Instead since the surrogate and the original CFD simulation are mixed during search, new data samples are available and the network can be adapted online. Thus, the best network is the one that is particularly well suited to continuously and quickly learn new CFD approximations during the evolutionary design optimization. In Fig. 9 the framework for the evolutionary optimization of the NN for problem classes is shown. Since it is easily confused, we should point out, that the evolutionary optimization of the architecture is still decoupled from the evolutionary design optimization where the best network is used as a surrogate. The results for the network that has been evolved to quickly adapt to problems from one and the same class are shown as ApxNN$^{(3)}$ in Fig. 8. We observe that during the first generations ApxNN$^{(3)}$ scores much better than ApxNN$^{(1)}$ (the fully connected NN) and similar to ApxNN$^{(2)}$ (the network whose structure was optimized using a standard evolutionary approach to minimize the approximation error for all data offline). However, in later generations, the performance of ApxNN$^{(3)}$ deteriorates and becomes unstable. Although this behavior is not yet fully understood, we believe that one reason might be the different update frequencies between the offline problem class training and the online design optimization. The update frequency denotes how often the original CFD simulation is called within a certain number of generations. As this frequency is adapted depending on the fidelity of the approximation model, it changes differently during offline structure optimization of the NN and online application of the network as a surrogate for the design optimization. Therefore, the definition of the problem class might change which is difficult to cope with for the network.

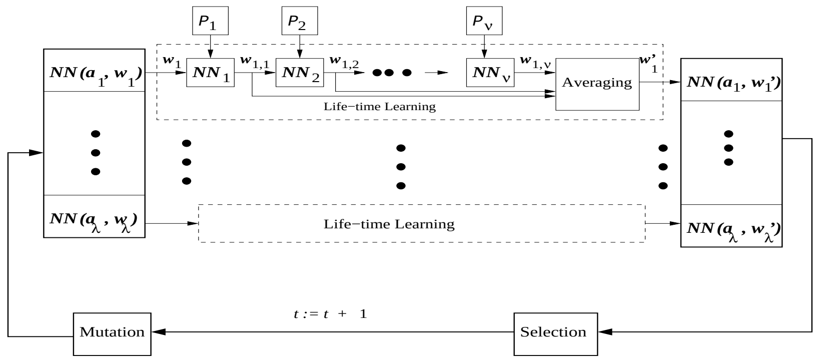

Fig. 9. Evolutionary structure optimization for problem classes with averaged Lamarckian inheritance, where the $\mathcal{P}_i$ denote the different problems belonging to one class and NN_i the network after learning $\mathcal{P}_i$. The symbol $w_{i,j}$ denotes the set of weights of the ith network after learning the jth problem, w_i' refers to the weights of the ith network after learning all ν problems. Averaged Lamarckian evolution is used to take the different problem characteristics into account to determine the set w_i', details can be found in [97]. The symbol a_i denotes the architecture or structure of the neural network which is not changed during the sequential learning of problems $1 \ldots \nu$.

5 Conclusion

Organic computing calls for adaptive systems. In order to be efficient and robust, these systems have to be specialized to certain problem classes comprising those scenarios they may face during operation. Nervous systems are perfect examples of such specialized learners and thus are prime candidates for the substrate of organic computing.

Computational models of nervous systems like artificial neural networks (NNs) have to be re-visited in the light of new adaptation schemes that put the structure of the system in the center of the focus based on issues like modularity, second-order generalization and learning efficiency.

At the same time, we promote the combination of evolutionary algorithms (EAs) and NNs not just because of an appealing metaphor, but also and foremost because EAs have proved to be well suited to solve many of the difficult optimization problems occurring when designing NNs, especially when higher order optimization methods cannot be applied. The field of evolutionary neural systems is expanding in many different directions as we have shown in this chapter. We have demonstrated how NNs can be evolved that are specialized to certain problem classes. Although still in its beginnings, this second order learning is not restricted to toy problems but has already found real-world technical applications.

Still, there is much left to do towards establishing the design triangle learning – development – evolution of neural systems in such a way that they can demonstrate their full potential. Results from brain science highlight the importance of architecture and of the way the architecture is build up during ontogeny. Although the incorporation of evolution and development into computational neuroscience is still in its beginning, we believe that this will be a promising approach to organic computing.

Acknowledgments

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

References

1. Arbib, M.A., ed.: The Handbook of Brain Theory and Neural Networks. 2 edn. MIT Press (2002)
2. Bishop, C.M.: Neural Networks for Pattern Recognition. Oxford University Press (1995)
3. Haykin, S.: Neural Networks: A Comprehensive Foundation. Prentice Hall (1998)
4. Bishop, C.M.: Pattern Recognition and Machine Learning. Springer-Verlag (2006)
5. Dayan, P., Abbott, L.F.: Theoretical neuroscience: Computational and mathematical modeling of neural systems. MIT Press (2001)
6. Koch, C., Segev, I.: The role of single neurons in information processing. Nature Neuroscience **3** (2000) 1171–1177
7. Siegelmann, H.T., Sontag, E.D.: On the computational power of neural nets. Journal of Computer and System Sciences **50**(1) (1995) 132–150
8. Šíma, J., Orponen, P.: General-purpose computation with neural networks: A survey of complexity theoretix results. Neural Computation **15**(12) (2003) 2727–2778
9. Sontag, E.D.: Recurrent neural networks: Some systems-theoretic aspects. In Karny, M., et al., eds.: Dealing with Complexity: A Neural Network Approach. Springer-Verlag (1997) 1–12
10. Šíma, J.: Training a single sigmoidal neuron is hard. Neural Computation **14** (2002) 2709–2728
11. Anthony, M., Bartlett, P.L.: Neural Network Learning: Theoretical Foundations. Cambridge University Press (1999)
12. Stagge, P., Sendhoff, B.: An extended Elman net for modeling time series. In Gerstner, W., Germond, A., Hasler, M., Nicoud, J.D., eds.: Artificial Neural Networks (ICANN'97). Volume 1327 of LNCS., Springer-Verlag (1997) 427–432
13. Mandic, D.P., Chambers, J.A.: Recurrent Neural Networks for Prediction: Learning Algorithms, Architectures and Stability. John Wiley and Sons Ltd (2001)
14. Igel, C., Kreutz, M.: Operator adaptation in evolutionary computation and its application to structure optimization of neural networks. Neurocomputing **55**(1-2) (2003) 347–361
15. Igel, C., Wiegand, S., Friedrichs, F.: Evolutionary optimization of neural systems: The use of self-adptation. In de Bruin, M.G., et al., eds.: Trends and Applications in Constructive Approximation. Number 151 in International Series of Numerical Mathematics. Birkhäuser Verlag (2005) 103–123

16. Rudolph, G.: Convergence Properties of Evolutionary Algorithms. Kovač, Hamburg (1997)
17. Jägersküpper, J.: How the (1+1) ES using isotropic mutations minimizes positive definite quadratic forms. Theoretical Computer Science **36**(1) (2006) 38–56
18. Droste, S., Jansen, T., Wegener, I.: On the analysis of the (1+1) evolutionary algorithm. Theoretical Computer Science **276** (2002) 51–81
19. Beyer, H.G., Schwefel, H.P., Wegener, I.: How to analyse evolutionary algorithms. Theoretical Computer Science **287** (2002) 101–130
20. Wolpert, D.H.: The lack of a priori distinctions between learning algorithms. Neural Computation **8**(7) (1996) 1341–1390
21. Wolpert, D.H., Macready, W.G.: No Free Lunch theorems for optimization. IEEE Transactions on Evolutionary Computation **1**(1) (1997) 67–82
22. Devroye, L., Gyorfi, L.: A Probabilistic Theory of Pattern Recognition. Springer-Verlag (1997)
23. Bousquet, O., Boucheron, S., Lugosi, G.: Introduction to Statistical Learning Theory. In: Advanced Lectures in Machine Learning. Volume 3176 of LNAI. Springer-Verlag (2004) 169–207
24. Schumacher, C., Vose, M.D., Whitley, L.D.: The No Free Lunch and description length. In Spector, L., et al., eds.: Genetic and Evolutionary Computation Conference (GECCO 2001), San Francisco, CA, USA, Morgan Kaufmann (2001) 565–570
25. Igel, C., Toussaint, M.: On classes of functions for which No Free Lunch results hold. Information Processing Letters **86**(6) (2003) 317–321
26. Igel, C., Toussaint, M.: A No-Free-Lunch theorem for non-uniform distributions of target functions. Journal of Mathematical Modelling and Algorithms **3**(4) (2004) 313–322
27. Wolpert, D.H., Macready, W.G.: Coevolutionary free lunches. IEEE Transactions on Evolutionary Computation **9** (2005)
28. Fogel, D.B., ed.: Evolutionary Computation: The Fossile Record. IEEE Press (1998)
29. Kampfner, R.R., Conrad, M.: Computational modeling of evolutionary learning processes in the brain. Bulletin of Mathematical Biology **45**(6) (1983) 931–968
30. Miller, G.F., Todd, P.M.: Designing neural networks using genetic algorithms. In Schaffer, J.D., ed.: Proceeding of the 3rd International Conference on Genetic Algorithms, Morgan Kaufmann (1989) 379–384
31. Nolfi, S.: Evolution and learning in neural networks. In Arbib, M.A., ed.: The Handbook of Brain Theory and Neural Networks. 2 edn. MIT Press (2002) 415–418
32. Patel, M., Honavar, V., Balakrishnan, K., eds.: Advances in the Evolutionary Synthesis of Intelligent Agents. MIT Press (2001)
33. Sendhoff, B.: Evolution of Structures – Optimization of Artificial Neural Structures for Information Processing. Shaker Verlag, Aachen (1998)
34. Yao, X.: Evolving artificial neural networks. Proceedings of the IEEE **87**(9) (1999) 1423–1447
35. Arbib, M.A.: Towards a neurally-inspired computer architecture. Natural Computing **2**(1) (2003) 1–46
36. Wiegand, S., Igel, C., Handmann, U.: Evolutionary multi-objective optimization of neural networks for face detection. International Journal of Computational Intelligence and Applications **4**(3) (2004) 237–253
37. Hüsken, M., Jin, Y., Sendhoff, B.: Structure optimization of neural networks for aerodynamic optimization. Soft Computing **9**(1) (2005) 21–28

38. Gepperth, A., Roth, S.: Applications of multi-objective structure optimization. Neurocomputing **6**(7-9) (2006) 701–713
39. Mahner, M., Kary, M.: What exactly are genomes, genotypes and phenotypes? And what about phenomes? Journal of Theoretical Biology **186**(1) (1997) 55–63
40. Igel, C., Stagge, P.: Effects of phenotypic redundancy in structure optimization. IEEE Transactions on Evolutionary Computation **6**(1) (2002) 74–85
41. Kitano, H.: Designing neural networks using genetic algorithms with graph generation system. Complex Systems **4** (1990) 461–476
42. Gruau, F.: Automatic definition of modular neural networks. Adaptive Behavior **3**(2) (1995) 151–183
43. Friedrich, C.M., Moraga, C.: An evolutionary method to find good building-blocks for architectures of artificial neural networks. In: Sixth International Conference on Information Processing and Management of Uncertainty in Knowledge Based Systems (IPMU'96). Volume 2. (1996) 951–956
44. Sendhoff, B., Kreutz, M.: A model for the dynamic interaction between evolution and learning. Neural Processing Letters **10**(3) (1999) 181–193
45. Das, I., Dennis, J.E.: A closer look at drawbacks of minimizing weighted sums of objectives for pareto set generation in multicriteria optimization problems. Structural Optimization **14**(1) (1997) 63–69
46. Jin, Y.: Multi-objective Machine Learning. Springer-Verlag (2006)
47. Coello Coello, C.A., Van Veldhuizen, D.A., Lamont, G.B.: Evolutionary Algorithms for Solving Multi-Objective Problems. Kluwer Academic Publishers (2002)
48. Deb, K.: Multi-Objective Optimization Using Evolutionary Algorithms. Wiley (2001)
49. Abbass, H.A.: An evolutionary artificial neural networks approach for breast cancer diagnosis. Artificial Intelligence in Medicine **25**(3) (2002) 265–281
50. Abbass, H.A.: Speeding up backpropagation using multiobjective evolutionary algorithms. Neural Computation **15**(11) (2003) 2705–2726
51. Jin, Y., Okabe, T., Sendhoff, B.: Neural network regularization and ensembling using multi-objective evolutionary algorithms. In: Congress on Evolutionary Computation (CEC'04), IEEE Press (2004) 1–8
52. Chandra, A., Yao, X.: Evolving hybrid ensembles of learning machines for better generalisation. Neurocomputing **69**(7–9) (2006) 686–700
53. Igel, C.: Multiobjective model selection for support vector machines. In Coello Coello, C.A., et al., eds.: Proceedings of the Third International Conference on Evolutionary Multi-Criterion Optimization (EMO 2005). Volume 3410 of LNAI., Springer-Verlag (2005) 534–546
54. Suttorp, T., Igel, C.: Multi-objective optimization of support vector machines. In Jin, Y., ed.: Multi-objective Machine Learning. Volume 16 of Studies in Computational Intelligence. Springer-Verlag (2006) 199–220
55. Tsitsiklis, J., Bertsekas, D.: Neurodynamic programming. Belmont, MA: Athena Scientific, U.S.A. (1996)
56. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press (1998)
57. Powell, W.B., Barto, A.G., Si, J.: Handbook of Learning and Approximate Dynamic Programming. Wiley-IEEE Press (2004)
58. Sutton, R.S., McAllester, D., Singh, S., Mansour, Y.: Policy gradient methods for reinforcement learning with function approximation. In Solla, S.A., et al., eds.: Advances in Neural Information Processing Systems 12, MIT Press (2000) 1057–1063

59. Konda, V.R., Tsitsiklis, J.N.: On actor-critic algorithms. SIAM Journal on Control and Optimization **42**(4) (2003) 1143–1166
60. Moriarty, D.E., Schultz, A.C., Grefenstette, J.J.: Evolutionary Algorithms for Reinforcement Learning. Journal of Artificial Intelligence Research **11** (1999) 199–229
61. Igel, C.: Neuroevolution for reinforcement learning using evolution strategies. In Sarker, R., et al., eds.: Congress on Evolutionary Computation (CEC 2003). Volume 4., IEEE Press (2003) 2588–2595
62. Pellecchia, A., Igel, C., Edelbrunner, J., Schöner, G.: Making driver modeling attractive. IEEE Intelligent Systems **20**(2) (2005) 8–12
63. Chellapilla, K., Fogel, D.B.: Evolution, neural networks, games, and intelligence. Proceedings of the IEEE **87**(9) (1999) 1471–1496
64. Fogel, D.B., Hays, T.J., Hahn, S.L., Quon, J.: A self-learning evolutionary chess program. Proceedings of the IEEE **92**(12) (2004) 1947–1954
65. Lucas, S.M., Kendall, G.: Evolutionary computation and games. Computational Intelligence Magazine, IEEE **1**(1) (2006) 10–18
66. Stanley, K.O., Bryant, B.D., Miikkulainen, R.: Evolving neural network agents in the NERO video game. In: Proceedings of the IEEE 2005 Symposium on Computational Intelligence and Games (CIG'05)., IEEE Press (2005)
67. Nolfi, S., Floreano, D.: Evolutionary Robotics: The Biology, Intelligence, and Techno logy of Self-Organizing Machines. Intelligent Robotics and Autonomous Agents. MIT Press (2000)
68. Lipson, H., Pollack, J.B.: Automatic design and manufacture of robotic lifeforms. Nature **406** (2000) 974–978
69. Pasemann, F., Steinmetz, U., Hülse, M., Lara, B.: Robot control and the evolution of modular neurodynamics. Theory in Biosciences **120**(3-4) (2001) 311–326
70. Walker, J., Garrett, S., Wilson, M.: Evolving controllers for real robots: A survey of the literature. Adaptive Behavior **11** (2003) 179–203
71. Sharkey, A.J.C.: On combining artificial neural nets. Connection Science **8**(3-4) (1996) 299–313
72. Liu, Y., Yao, X., Higuchi, T.: Evolutionary ensembles with negative correlation learning. IEEE Transactions on Evolutionary Computation **4**(4) (2000) 380–387
73. Mark, A., Wersing, H., Sendhoff, B.: A decision making framework for game playing using evolutionary optimization and learning. In Shi, Y., ed.: Congress on Evolutionary Computation (CEC). Volume 1., IEEE Press (2004) 373–380
74. Jin, Y., Sendhoff, B.: Reducing fitness evaluations using clustering techniques and neural network ensembles. In Deb, K., et al., eds.: Proceedings of the Genetic and Evolutionary Computation Conference - GECCO. Volume 1 of LNCS., Springer-Verlag (2004) 688–699
75. Cortes, C., Vapnik, V.: Support-vector networks. Machine Learning **20**(3) (1995) 273–297
76. Cristianini, N., Shawe-Taylor, J.: An Introduction to Support Vector Machines and other kernel-based learning methods. Cambridge University Press (2000)
77. Schölkopf, B., Smola, A.J.: Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond. MIT Press (2002)
78. Friedrichs, F., Igel, C.: Evolutionary tuning of multiple SVM parameters. Neurocomputing **64**(C) (2005) 107–117
79. Runarsson, T.P., Sigurdsson, S.: Asynchronous parallel evolutionary model selection for support vector machines. Neural Information Processing – Letters and Reviews **3**(3) (2004) 59–68

80. Eads, D.R., Hill, D., Davis, S., Perkins, S.J., Ma, J., Porter, R.B., Theiler, J.P.: Genetic algorithms and support vector machines for time series classification. In Bosacchi, B., et al., eds.: Applications and Science of Neural Networks, Fuzzy Systems, and Evolutionary Computation V. Volume 4787 of Proceedings of the SPIE. (2002) 74–85
81. Fröhlich, H., Chapelle, O., Schölkopf, B.: Feature selection for support vector machines using genetic algorithms. International Journal on Artificial Intelligence Tools **13**(4) (2004) 791–800
82. Arai, K., Das, S., Keller, E.L., Aiyoshi, E.: A distributed model of the saccade system: simulations of temporally perturbed saccades using position and velocity feedback. Neural Networks **12** (1999) 1359–1375
83. Schneider, S., Igel, C., Klaes, C., Dinse, H., Wiemer, J.: Evolutionary adaptation of nonlinear dynamical systems in computational neuroscience. Journal of Genetic Programming and Evolvable Machines **5**(2) (2004) 215–227
84. Sporns, O., Tononi, G., Edelman, G.M.: Theoretical neuroanatomy: relating anatomical and functional connectivity in graphs and cortical connection matrices. Cerebral Cortex **10**(2) (2000) 127–141
85. Igel, C., von Seelen, W., Erlhagen, W., Jancke, D.: Evolving field models for inhibition effects in early vision. Neurocomputing **44-46**(C) (2002) 467–472
86. Rolls, E.T., Stringer, S.M.: On the design of neural networks in the brain by genetic evolution. Progress in Neurobiology **6**(61) (2000) 557–579
87. Fukushima, K.: Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. Biological Cybernetics **39** (1980) 139–202
88. Wersing, H., Körner, E.: Learning optimized features for hierarchical models of invariant recognition. Neural Computation **15**(7) (2003) 1559–1588
89. Teo, M.Y., Khoo, L.P., Sim, S.K.: Application of genetic algorithms to optimise neocognitron network parameters. Neural Network World **7**(3) (1997) 293–304
90. Pan, Z., Sabisch, T., Adams, R., Bolouri, H.: Staged training of neocognitron by evolutionary algorithms. In Angeline, P.J., et al., eds.: Proceedings of the Congress on Evolutionary Computation. Volume 3., IEEE Press (1999) 1965–1972
91. Shi, D., Tan, C.L.: GA-based supervised learning of neocognitron. In: International Joint Conference on Neural Network (IJCNN 2000). IEEE Press (2000)
92. Schneider, G., Wersing, H., Sendhoff, B., Körner, E.: Coupling of evolution and learning to optimize a hierarchical object recognition model. In Yao, X., et al., eds.: Parallel Problem Solving from Nature (PPSN). LNCS, Springer-Verlag (2004) 662–671
93. Schneider, G., Wersing, H., Sendhoff, B., Körner, E.: Evolutionary optimization of an hierarchical object recognition model. IEEE Transactions on Systems, Man and Cybernetics, Part B **35**(3) (2005) 426–437
94. Hüsken, M., Gayko, J.E., Sendhoff, B.: Optimization for problem classes – Neural networks that learn to learn. In X.Yao, ed.: IEEE Symposium on Combinations of Evolutionary Computation and Neural Networks, IEEE Press (2000) 98-109.
95. Hüsken, M., Sendhoff, B.: Evolutionary optimization for problem classes with Lamarckian inheritance. In Lee, S.Y., ed.: Seventh International Conference on Neural Information Processing – Proceedings. Volume 2., Taejon, Korea (2000) 897–902
96. Thrun, S., Pratt, L., eds.: Learning to Learn. Kluwer Academic Publishers (1998)
97. Hüsken, M., Jin, Y., Sendhoff, B.: Structure optimization of neural networks for evolutionary design optimization. Soft Computing Journal **9**(1) (2005) 21–28

98. Hüsken, M., Igel, C., Toussaint, M.: Task-dependent evolution of modularity in neural networks. Connection Science **14**(3) (2002) 219–229
99. Khare, V.R., Yao, X., Sendhoff, B.: Multi-network evolutionary systems and automatic decomposition of complex problems. International Journal of General Systems **35**(3) (2006) 259–274
100. Sonoda, T., Yamaguchi, Y., Arima, T., Olhofer, M., Sendhoff, B., Schreiber, H.A.: Advanced high turning compressor airfoils for low Reynolds number condition, Part 1: Design and optimization. Journal of Turbomachinery **126** (2004) 350–359
101. Foli, K., Okabe, T., Olhofer, M., Jin, Y., Sendhoff, B.: Optimization of micro heat exchanger: CFD, analytical approach and multi-objective evolutionary algorithms. International Journal of Heat and Mass Transfer **49**(5-6) (2005) 1090–1099
102. Obayashi, S., Yamaguchi, Y., Nakamura, T.: Multiobjective genetic algorithm for multidisciplinary design of transonic wing planform. Journal of Aircraft **34**(5) (1997) 690–693
103. Grefenstette, J.J., Fitzpatrick, J.M.: Genetic search with approximate fitness evaluations. In Grefenstette, J.J., ed.: International Conference on Genetic Algorithms and Their Applications, Lawrence Erlbaum Associates (1985) 112–120
104. Jin, Y.: A comprehensive survey of fitness approximation in evolutionary computation. Soft Computing **9**(1) (2005) 3–12
105. Jin, Y., Olhofer, M., Sendhoff, B.: A framework for evolutionary optimization with approximate fitness functions. IEEE Transactions on Evolutionary Computation **6**(5) (2002) 481–494
106. Pierret, S.: Turbomachinery blade design using a Navier-Stokes solver and artificial neural network. ASME Journal of Turbomachinery **121**(3) (1999) 326–332
107. Papadrakakis, M., Lagaros, N., Tsompanakis, Y.: Optimization of large-scale 3D trusses using evolution strategies and neural networks. International Journal of Space Structures **14**(3) (1999) 211–223
108. Graening, L., Jin, Y., Sendhoff, B.: Efficient evolutionary optimization using individual-based evolution control and neural networks: A comparative study. In Verleysen, M., ed.: 13th European Symposium on Artificial Neural Networks (ESANN 2005). (2005) 273–278