ESANN'2005 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 27-29 April 2005, d-side publi., ISBN 2-930307-05-6.

Synergies between Evolutionary and Neural Computation

Christian Igel[1] and Bernhard Sendhoff[2]

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

Abstract. Evolutionary and neural computation share the same philosophy to use biological information processing for the solution of technical problems. Besides this important but rather abstract common foundation, there have also been many successful combinations of both methods for solving problems as applied as the design of turbomachinery components. In this paper, we will introduce evolutionary algorithms primarily for a "neural" audience and demonstrate their usefulness for neural computation. Furthermore, we will introduce a list of some more recent trends in combining evolutionary and neural computation, that will show that synergies between the two fields go beyond the typically quoted example of topology optimisation of neural networks. We strive to increase the awareness for these trends in the neural computation community and spark some interest in one or the other of the shown directions.

1 Introduction

Artificial evolutionary and neural systems have a long history, which in many respects resembles each other. In their beginnings, they were both met with considerable scepticism from both the biological as well as the technological world. 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 it 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 evolutionary algorithms (EAs) and may be traced back to the 50s, see [15]. However, it took another 30 years until first papers were published describing explicitly the application of EAs to neural networks (NNs) [40, 48]. The subject quickly received considerable interest—mostly however in the evolutionary computation community—and several papers were published in the early nineties concentrating on both the optimisation 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, to evolve neural systems showing *qualitatively* new behaviour 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, the nature of learning, and the development of neural systems have been raised and important results have been obtained.

There are still only few works on connecting brain science or computational neuroscience with evolutionary computation, however, first promising attempts have been made. On more general terms, it is a reasonable question to ask whether it will be possible to understand the brain without understanding how it evolved. 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 organisation 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 organisation. If we follow this line of thought, the only viable conclusion is that much more work should be done in combining evolutionary methodology with models of the biological brain.

From a technical point of view, the information processing capabilities of vertebrate brains outperform artificial systems in many respects. Abstract models of NNs exist that, in principle, exhibit universal approximation and computation properties. However, the general question of how to efficiently design an appropriate neural system for a given task remains open and complexity theory reveals the need for using heuristics (e.g. [68]). The answer is likely to be found by investigating the three major organisation principles of biological NNs: evolution, self-organisation, and learning.

In the following, we introduce EAs in the framework of stochastic search. In section 3, we discuss evolutionary structure optimisation of neural systems. Thereafter, we will suggest some more recent trends in combining EAs and neural systems. Needless to say that such a collection is a subjective, biased selection. We do neither claim to be fair to other researchers nor that the list is complete, but hope to present some interesting and relatively novel approaches. General surveys (beyond the scope of this paper) can be found in [51, 56, 64, 75].

2 Evolutionary Computation

Evolutionary algorithms (EAs) can be considered a special class of global random search algorithms. Let the search problem under consideration be described by a quality function $f : \mathcal{G} \to \mathcal{F}$ to be optimised, 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 as follows:

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

② Obtain λ points $\boldsymbol{g}^{(t)}_1, \ldots, \boldsymbol{g}^{(t)}_\lambda$ 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.

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 $\tilde{\boldsymbol{g}}_1^{(t)}, \ldots, \tilde{\boldsymbol{g}}_\mu^{(t)} \in \mathcal{G}$. Each point corresponds to the *genotype* of an *individual*. In each generation, λ *offspring* $\boldsymbol{g}_1^{(t)}, \ldots, \boldsymbol{g}_\lambda^{(t)} \in \mathcal{G}$ 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 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 $\mathcal{G}^l$, $1 \leq k \leq \mu, 1 \leq l \leq \lambda$. 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 $P_{\mathcal{G}^\lambda}\left(\boldsymbol{g}_1, \ldots, \boldsymbol{g}_\lambda \,|\, \tilde{\boldsymbol{g}}_1^{(t)}, \ldots, \tilde{\boldsymbol{g}}_\mu^{(t)}; \boldsymbol{\theta}^{(t)}\right)$ be the probability that parents $\tilde{\boldsymbol{g}}_1^{(t)}$, $\ldots$, $\tilde{\boldsymbol{g}}_\mu^{(t)}$ create offspring $\boldsymbol{g}_1^{(t)}, \ldots, \boldsymbol{g}_\lambda^{(t)}$. This distribution is additionally parameterised 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. 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 optimisation problems that are non-differentiable, non-continuous, multi-modal, and noisy. They are easy to parallelise by distributing the fitness evaluations of the offspring. In single-objective optimisation, we usually have $\mathcal{F} \subset \mathbb{R}$, whereas in multi-objective optimisation, see section 4.1, vector-valued functions (e.g. $\mathcal{F} \subset \mathbb{R}^k, k > 1$) are considered. In co-evolution (see section 4.2), where individuals in one generation contribute reciprocally to the forces of selection, the scenario is extended to fitness functions that do not consider each individual in isolation, but in the context of the current population (i.e. $f : \mathcal{G}^\lambda \to \mathcal{F}^\lambda$ or even $f : \mathcal{G}^{\lambda+\mu} \to \mathcal{F}^{\lambda+\mu}$ 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 optimisation of NNs in [30, 33].

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 optimisation 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 [59]), there exist rigorous expected runtime analyses of simplified algorithms on restricted, but important classes of optimisation problems, see [34, 13] and references therein. The article [5] provides good starting points for reading about EA theory.

3 Evolutionary Structure Optimisation of Neural Systems

Although NNs are successfully applied to support evolutionary computation (see section 4.7), the most prominent combination of EAs and NNs is evolutionary optimisation 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 the sample data. Learning of an adaptive (e.g. neural) system can be defined as goal-directed, data-driven changing of its behaviour. Examples of learning algorithms for technical NNs include gradient-based heuristics or quadratic program solvers. Such "classical" optimisation methods are usually considerably faster than pure evolutionary optimisation of these parameters, although they might be more prone to getting stuck in local minima. However, there are cases where "classical" optimisation methods are not applicable, for example when the neural model or the objective function is non-differentiable (e.g. see section 4.2). Still, the main application of evolutionary optimisation in the field of neurocomputing is adapting the structures of neural systems, that is, optimising those parts that are not altered by the learning algorithm. Both in biological and technical neural systems the structure is crucial for the learning behaviour—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" [4]. 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.

Encodings in Structure Optimisation. A prototypical example of evolutionary optimisation of a neural architecture on which a learning algorithm operates is the search for an appropriate topology for a multi-layer perceptron NN, see [74, 27, 21] for some real-world applications. Here, the search space ultimately consists of graphs. When using EAs to design NN graphs, the key questions are

how to encode the topologies and how to define variation operators that operate on this representation. In terms of section 2, operators and representation both determine the search distribution and thereby the neighbourhood 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 optimisation process itself. The fitness function can then be written as $f = f' \circ \phi$, where $\phi : \mathcal{G} \to \mathcal{P}$ and $f' : \mathcal{P} \to \mathcal{F}$. The definition of the phenotype space is to a certain degree arbitrary (note that the freedom in the definition of the phenotype space equally exists in evolutionary biology [46] 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 (in the context of NNs, e.g. see [31]). We assume that $\mathcal{P}$ is equipped with an extrinsic (i.e. independent of the evolutionary process) metric or at least a consistent neighbourhood 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 optimisation 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 programme or a mapping to build, grow or develop the phenotype. Such encoding foster the design of large, modular systems. Examples can be found in [42, 24, 17, 64, 65].

4 Trends in Combining EAs and Neural Computation

4.1 Multi-objective Optimisation of Neural Networks

Designing a neural system usually requires optimisation 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 [11]. 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 tradeoffs between the objectives explicit (e.g. visualise them) after the design process and select from a diverse set of systems the one that seems to be most appropriate. This can be realised by

"true" multi-objective optimisation (MOO). The MOO algorithms approximate the set of Pareto-optimal tradeoffs, 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 optimisation. There have been considerable advances in MOO recently, which can now be incorporated into machine learning techniques. In particular, it was realised that EAs are very well suited for multi-criterion optimisation and they have become the MOO methods of choice in the last years [10, 12]. Recent applications of evolutionary MOO to neural systems address the design of multi-layer perceptron NNs [1, 2, 36, 74, 21, 8] and support vector machines (SVMs) [29].

4.2 Reinforcement Learning

In the standard reinforcement learning (RL) 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 maximised. 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 [70, 43]). Evolutionary algorithms have proven to be powerful and competitive methods for solving RL problems [50, 28, 41]. The recent success of evolved NNs in game playing [9, 16] 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, are often easier to apply and are more robust with respect to the tuning of the meta-parameters (learning rates, etc.), can be applied if the function approximators are non-differentiable, and can also optimise 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 [52, 44, 55, 72]. Here promising applications of the principle of co-evolution can be found.

4.3 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 [66]. The population concept in EAs appears to be ideal for designing neural network ensembles, as, for example, demonstrated for classification tasks in [45, 8]. In the framework of decision making and games, Mark et al. [47] developed a combination of NN ensembles and evolutionary computation. Two ensembles are used to predict the opponents strategy and to optimise 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 optimisation (see section 4.7). In [38] network ensembles have been optimised 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.

4.4 Evolutionary Optimisation of Learning

Neural networks are often chosen as approximation or classification models because of their learning and generalisation behaviour. Therefore, it seems reasonable to target an improvement of these capabilities using evolutionary computation. This includes the optimisation of learning rules (e.g. see [6, 7]). Further, the neural structure can be evolved to improve the learning behaviour [25, 26]. In [25], *networks that learn to learn* are evolved for efficiently solving classes of toy as well as real-world problems. The authors propose different ways to realise this additional generalisation property called 2^{nd} order generalisation.

4.5 Optimising Kernel Methods

Adopting the extended definition of structure as that part of the adaptive system that cannot be optimised by the learning algorithm itself, model selection of kernel-based methods is a structure optimisation problem. For example, choosing the right kernel for a SVM is important for its performance. When a parameterised 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 [18, 60, 29], evolution strategies (i.e. EAs tailored for real-valued optimisation) were proposed for adapting SVM hyperparameters, in [14, 39, 49, 19] genetic algorithms (EAs that represent candidate solutions as fixed-length strings over a finite alphabet) were used for SVM feature selection.

4.6 Computational Neuroscience and Brain-inspired Architectures

There are only a few applications of evolutionary computation in brain science [3, 63, 69, 32, 58], 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 favour 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 [20, 73] evolutionary algorithms have been used to optimise 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 [71, 53, 67], which was one of the first hierarchical vision systems based on the structure of its biological counterpart [20]. More recent work employed evolution strategies to optimise the nonlinearities and the structure of a biologically inspired vision network, which is capable of performing a complex 3D real world object classification task [61, 62]. Schneider et al. employed a directly coded evolutionary optimisation that was capable of performing well in a 1800-dimensional search space. In a second experiment evolutionary optimisation was successfully combined with local unsupervised learning based on a sparse representation. The resulting architecture outperformed other methods like SVMs.

4.7 Neural Networks as Meta-models

Linear or quadratic models or in more general terms meta-models have been used in experimental optimisation since many years to approximate experimental data. The optimisation is then carried out using the model as the objective function. After the optimisation algorithm has converged, the optimal solution is tested in an experiment. If necessary the response surface model is rebuild and the optimisation process is started again. More recently, systems have been subject to optimisation that are so complex that even simulations are very time-consuming and costly. Therefore, the meta-model framework has been applied to optimisation in general and in particular to EAs. Although first approaches to combine fitness approximation with EAs are relatively old [23], it is only in the last couple of years that the field has received wider attention, see [35] for a review. It has been revealed that the strategy to keep the update of the meta-model and the optimisation process separate is not advisable, since the optimisation is easily misled if the modelling quality is limited (which is often the case in practical applications). Jin et al. [37] 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 optimisation process (e.g. [37, 57, 54]). In [27, 22] evolutionary structure optimisation methods have been successfully used to improve NNs which are in turn used as approximation models in an evolutionary design optimisation framework.

5 Conclusion

Finding an appropriate neural system for a given task usually requires the solution of difficult optimisation problems. Evolutionary algorithms (EAs) are well

suited to solve many of these problems, especially when higher order optimisation methods cannot be applied. Therefore, we argue that EAs should be a tool for the design of neural systems that is as common as gradient descent methods.

At the same time there is plenty of room for synergies that go beyond the use of evolutionary computation as structure optimisation methods for neural systems. Some of these have been highlighted in the trend list which we presented in this paper. We can only speculate which one might have the highest potential for a long-lasting contribution. However, it seems that bringing evolutionary computation together with brain science bears many advantage both on a technical level (e.g. for data analysis) and on an explanatory level (e.g. as a means to better understand the brain structure).

References

[1] H. A. Abbass. An evolutionary artificial neural networks approach for breast cancer diagnosis. *Artificial Intelligence in Medicine*, 25(3):265–281, 2002.

[2] H. A. Abbass. Speeding up backpropagation using multiobjective evolutionary algorithms. *Neural Computation*, 15(11):2705–2726, 2003.

[3] K. Arai, S. Das, E. L. Keller, and E. Aiyoshi. A distributed model of the saccade system: simulations of temporally perturbed saccades using position and velocity feedback. *Neural Networks*, 12:1359–1375, 1999.

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

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

[6] J. A. Bullinaria. Evolving efficient learning algorithms for binary mappings. *Neural Networks*, 16:793–800, 2003.

[7] J. A. Bullinaria. Evolving neural networks: Is it really worth the effort? In *13th European Symposium on Artificial Neural Networks (ESANN 2005)*, 2005.

[8] A. Chandra and X. Yao. Evolutionary framework for the construction of diverse hybrid ensembles. In *13th European Symposium on Artificial Neural Networks (ESANN 2005)*, 2005.

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

[10] C. A. Coello Coello, D. A. Van Veldhuizen, and G. B. Lamont. *Evolutionary Algorithms for Solving Multi-Objective Problems*. Kluwer Academic Publishers, 2002.

[11] I. Das and J. E. Dennis. A closer look at drawbacks of minimizing weighted sums of objectives for pareto set generation in multicriteria optimization problems. *Structural Optimization*, 14(1):63–69, 1997.

[12] K. Deb. *Multi-Objective Optimization Using Evolutionary Algorithms*. Wiley, 2001.

[13] S. Droste, T. Jansen, and I. Wegener. On the analysis of the (1+1) evolutionary algorithm. *Theoretical Computer Science*, 276:51–81, 2002.

[14] D. R. Eads, D. Hill, S. Davis, S. J. Perkins, J. Ma, R. B. Porter, and J. P. Theiler. Genetic algorithms and support vector machines for time series classification. In B. Bosacchi et al., editors, *Applications and Science of Neural Networks, Fuzzy Systems, and Evolutionary Computation V.*, volume 4787 of *Proceedings of the SPIE*, pages 74–85, 2002.

[15] D. B. Fogel, editor. *Evolutionary Computation: The Fossile Record*. IEEE Press, 1998.

[16] D. B. Fogel, T. J. Hays, S. L. Hahn, and J. Quon. A self-learning evolutionary chess program. *Proceedings of the IEEE*, 92(12):1947–1954, 2004.

[17] C. M. Friedrich and C. Moraga. 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, pages 951–956, 1996.

[18] 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.

[19] H. Fröhlich, O. Chapelle, and B. Schölkopf. Feature selection for support vector machines by means of genetic algorithms. In *15th IEEE International Conference on Tools with AI (ICTAI 2003)*, pages 142–148. IEEE Computer Society, 2003.

[20] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 39:139–202, 1980.

[21] A. Gepperth and S. Roth. Applications of multi-objective structure optimization. In *13th European Symposium on Artificial Neural Networks (ESANN 2005)*, 2005.

[22] L. Graening, Y. Jin, and B. Sendhoff. Efficient evolutionary optimization using individual-based evolution control and neural networks: A comparative study. In *13th European Symposium on Artificial Neural Networks (ESANN 2005)*, 2005.

[23] J. J. Grefenstette and J. M. Fitzpatrick. Genetic search with approximate fitness evaluations. In J. J. Grefenstette, editor, *International Conference on Genetic Algorithms and Their Applications*, pages 112–120. Lawrence Erlbaum Associates, 1985.

[24] F. Gruau. Automatic definition of modular neural networks. *Adaptive Behavior*, 3(2):151–183, 1995.

[25] M. Hüsken, J. E. Gayko, and B. Sendhoff. Optimization for problem classes – neural networks that learn to learn. In X.Yao, editor, *IEEE Symposium on Combinations of Evolutionary Computation and Neural Networks*. IEEE Press, 2000. 98-109.

[26] M. Hüsken, C. Igel, and M. Toussaint. Task-dependent evolution of modularity in neural networks. *Connection Science*, 14(3):219–229, 2002.

[27] M. Hüsken, Y. Jin, and B. Sendhoff. Structure optimization of neural networks for aerodynamic optimization. *Soft Computing*, 9(1):21–28, 2005.

[28] C. Igel. Neuroevolution for reinforcement learning using evolution strategies. In R. Sarker et al., editors, *Congress on Evolutionary Computation (CEC 2003)*, volume 4, pages 2588–2595. IEEE Press, 2003.

[29] C. Igel. Multiobjective model selection for support vector machines. In C. A. Coello Coello et al., editors, *Proceedings of the Third International Conference on Evolutionary Multi-Criterion Optimization (EMO 2005)*, volume 3410 of *LNAI*, pages 534–546. Springer-Verlag, 2005.

[30] 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.

[31] C. Igel and P. Stagge. Effects of phenotypic redundancy in structure optimization. *IEEE Transactions on Evolutionary Computation*, 6(1):74–85, 2002.

[32] C. Igel, W. von Seelen, W. Erlhagen, and D. Jancke. Evolving field models for inhibition effects in early vision. *Neurocomputing*, 44-46(C):467–472, 2002.

[33] C. Igel, S. Wiegand, and F. Friedrichs. Evolutionary optimization of neural systems: The use of self-adptation. In M. G. de Bruin et al., editors, *Trends and Applications in Constructive Approximation*, number 151 in International Series of Numerical Mathematics. Birkhäuser Verlag, 2005. In press.

[34] J. Jägersküpper. Rigorous runtime analysis of the (1+1) ES: 1/5-rule and ellipsoidal fitness landscapes. In *Foundations of Genetic Algorithms (FOGA VIII)*, LNCS. Springer-Verlag, 2005.

[35] Y. Jin. A comprehensive survey of fitness approximation in evolutionary computation. *Soft Computing*, 9(1):3–12, 2005.

[36] Y. Jin, T. Okabe, and B. Sendhoff. Neural network regularization and ensembling using multi-objective evolutionary algorithms. In *Congress on Evolutionary Computation (CEC'04)*, pages 1–8. IEEE Press, 2004.

[37] Y. Jin, M. Olhofer, and B. Sendhoff. A framework for evolutionary optimization with approximate fitness functions. *IEEE Transactions on Evolutionary Computation*, 6(5):481–494, 2002.

[38] Y. Jin and B. Sendhoff. Reducing fitness evaluations using clustering techniques and neural network ensembles. In K. Deb et al., editors, *Proceedings of the Genetic and Evolutionary Computation Conference - GECCO*, volume 1 of *LNCS*, pages 688–699. Springer, 2004.

[39] K. Jong, E. Marchiori, and A. van der Vaart. Analysis of proteomic pattern data for cancer detection. In G. R. Raidl et al., editors, *Applications of Evolutionary Computing*, number 3005 in LNCS, pages 41–51. Springer-Verlag, 2004.

[40] R. R. Kampfner and M. Conrad. Computational modeling of evolutionary learning processes in the brain. *Bulletin of Mathematical Biology*, 45(6):931–968, 1983.

[41] Y. Kassahun and G. Sommer. Efficient reinforcement learning through evolutionary acquisition of neural topologies. In *13th European Symposium on Artificial Neural Networks (ESANN 2005)*, 2005.

[42] H. Kitano. Designing neural networks using genetic algorithms with graph generation system. *Complex Systems*, 4:461–476, 1990.

[43] V. R. Konda and J. N. Tsitsiklis. On actor-critic algorithms. *SIAM Journal on Control and Optimization*, 42(4):1143–1166, 2003.

[44] H. Lipson and J. B. Pollack. Automatic design and manufacture of robotic lifeforms. *Nature*, 406:974–978, 2000.

[45] Y. Liu, X. Yao, and T. Higuchi. Evolutionary ensembles with negative correlation learning. *IEEE Transactions on Evolutionary Computation*, 4(4):380–387, 2000.

[46] M. Mahner and M. Kary. What exactly are genomes, genotypes and phenotypes? And what about phenomes? *Journal of Theoretical Biology*, 186(1):55–63, 1997.

[47] A. Mark, H. Wersing, and B. Sendhoff. A decision making framework for game playing using evolutionary optimization and learning. In Y. Shi, editor, *Congress on Evolutionary Computation (CEC)*, volume 1, pages 373–380. IEEE Press, 2004.

[48] G. Miller and P. Todd. Designing neural networks using genetic algorithms. In J. D. Schaffer, editor, *Proceeding of the 3rd International Conference on Genetic Algorithms*, pages 379–384. Morgan Kaufmann, 1989.

[49] M. T. Miller, A. K. Jerebko, J. D. Malley, and R. M. Summers. Feature selection for computer-aided polyp detection using genetic algorithms. In A. V. Clough and A. A. Amini, editors, *Medical Imaging 2003: Physiology and Function: Methods, Systems, and Applications*, volume 5031 of *Proceedings of the SPIE*, pages 102–110, 2003.

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

[51] 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.

[52] S. Nolfi and D. Floreano. *Evolutionary Robotics: The Biology, Intelligence, and Techno logy of Self-Organizing Machines.* Intelligent Robotics and Autonomous Agents. MIT Press, 2000.

[53] Z. Pan, T. Sabisch, R. Adams, and H. Bolouri. Staged training of neocognitron by evolutionary algorithms. In P. J. Angeline et al., editors, *Proceedings of the Congress on Evolutionary Computation*, volume 3, pages 1965–1972. IEEE Press, 1999.

[54] M. Papadrakakis, N. Lagaros, and Y. Tsompanakis. Optimization of large-scale 3D trusses using evolution strategies and neural networks. *International Journal of Space Structures*, 14(3):211–223, 1999.

[55] F. Pasemann, U. Steinmetz, M. Hülse, and B. Lara. Robot control and the evolution of modular neurodynamics. *Theory in Biosciences*, 120(3-4):311–326, 2001.

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

[57] S. Pierret. Turbomachinery blade design using a Navier-Stokes solver and artificial neural network. *ASME Journal of Turbomachinery*, 121(3):326–332, 1999.

[58] E. T. Rolls and S. M. Stringer. On the design of neural networks in the brain by genetic evolution. *Progress in Neurobiology*, 6(61):557–579, 2000.

[59] G. Rudolph. *Convergence Properties of Evolutionary Algorithms*. Kovač, Hamburg, 1997.

[60] T. P. Runarsson and S. Sigurdsson. Asynchronous parallel evolutionary model selection for support vector machines. *Neural Information Processing – Letters and Reviews*, 3(3):59–68, 2004.

[61] G. Schneider, H. Wersing, B. Sendhoff, and E. Körner. Coupling of evolution and learning to optimize a hierarchical object recognition model. In X. Yao et al., editors, *Parallel Problem Solving from Nature (PPSN)*, LNCS, pages 662–671. Springer Verlag, 2004.

[62] G. Schneider, H. Wersing, B. Sendhoff, and E. Körner. Evolutionary optimization of an hierarchical object recognition model. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, 2004. Accepted.

[63] S. Schneider, C. Igel, C. Klaes, H. Dinse, and J. Wiemer. Evolutionary adaptation of nonlinear dynamical systems in computational neuroscience. *Journal of Genetic Programming and Evolvable Machines*, 5(2):215–227, 2004.

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

[65] B. Sendhoff and M. Kreutz. A model for the dynamic interaction between evolution and learning. *Neural Processing Letters*, 10(3):181–193, 1999.

[66] A. J. C. Sharkey. On combining artificial neural nets. *Connection Science*, 8(3-4):299–313, 1996.

[67] D. Shi and C. L. Tan. GA-based supervised learning of neocognitron. In *International Joint Conference on Neural Network (IJCNN 2000)*. IEEE Press, 2000.

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

[69] O. Sporns, G. Tononi, and G. M. Edelman. Theoretical neuroanatomy: relating anatomical and functional connectivity in graphs and cortical connection matrices. *Cerebral Cortex*, 10(2):127–141, 2000.

[70] R. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In S. A. Solla et al., editors, *Advances in Neural Information Processing Systems 12*, pages 1057–1063. MIT Press, 2000.

[71] M.-Y. Teo, L.-P. Khoo, and S.-K. Sim. Application of genetic algorithms to optimise neocognitron network parameters. *Neural Network World*, 7(3):293–304, 1997.

[72] J. Walker, S. Garrett, and M. Wilson. Evolving controllers for real robots: A survey of the literature. *Adaptive Behavior*, 11:179–203, 2003.

[73] H. Wersing and E. Körner. Learning optimized features for hierarchical models of invariant recognition. *Neural Computation*, 15(7):1559–1588, 2003.

[74] S. Wiegand, C. Igel, and U. Handmann. Evolutionary multi-objective optimization of neural networks for face detection. *International Journal of Computational Intelligence and Applications*, 4(3):237–253, 2004.

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