To be published in:
GECCO-2002: Proceedings of the Genetic and Evolutionary Computation Conference

Balancing Learning and Evolution

Michael Hüsken **Christian Igel**
Institut für Neuroinformatik
Ruhr-Universität Bochum
44780 Bochum, Germany
{michael.huesken,christian.igel}@neuroinformatik.ruhr-uni-bochum.de

Abstract

Finding the right coupling of learning and evolution in a hybrid algorithm is an open problem. In this article, we present a strategy to adjust the time spent on learning during evolutionary optimization of neural networks. The proposed adaptation scheme leads to a significant improvement in performance. It is empirically shown that suitable learning strategies strongly depend on the problem and that it is advantageous to adapt the time spent on learning during evolution.

1 INTRODUCTION

Evolution and learning are biologically-inspired optimization paradigms that have proven to be efficient in a wide range of applications, particularly when—as in nature—combined in hybrid strategies. Both techniques have different characteristics and their coupling aims at getting the best of both worlds. Evolution and learning interact in complex ways (e.g., see Mayley, 1996; Nolfi and Parisi, 1999). Further, it appears to be obvious that the optimal way of combining them depends on the target problem and may change during optimization. However, a systematic way to balance the two strategies is still missing. In this article, we present an adaptive method for controlling the time spent on learning in the framework of evolutionary optimization of neural networks.

Structure optimization of artificial neural networks is the most prominent technical example of a successful combination of evolution and learning, where evolution is employed to optimize mainly the network's architecture and learning (or training) is usually identified with gradient-based adaptation of the weights (Yao, 1999). In this paper, we assume that learning and evolution interact in a typical manner: Each offspring individual, representing one network architecture with the corresponding initial weights, is trained once immediately after its creation. The fitness of the individual is determined after training.

Most learning algorithms are iterative methods and their computational complexity usually scales linearly with the number of iterations. When searching for a neural network for a given task, several questions arise: Is learning necessary or is evolution sufficient? If learning is employed, how should evolution and learning be coupled? How long should each system learn? We address the last question, which might lead to an answer to the first one. For several reasons the learning time is a crucial parameter, e.g.:

- In most cases, the computational costs for generating and evaluating an offspring can be neglected compared to the costs for training. For standard neural network architectures, even a single learning iteration typically takes about twice the time of one fitness evaluation.
- If the learning time is too long, the neural network may overfit and lose its ability to generalize, i.e., to react in a desired way to data not used for learning. Even if no overfitting occurs, long learning might be a waste of resources in cases of bad local minima or insufficient architectures.
- If the maximum number of iterations is too small, the weights may not have enough time to adapt to an altered architecture. Hence, better architectures may be discarded, because the learning time is too short to uncover their benefits.

It is intuitive—and will become obvious in this study—that the right learning time depends on the problem at hand and on the course of evolution. Nevertheless,

no systematic way exists either to choose the learning time for a given task or to adjust it during evolution.

In this article, we present a strategy to adapt the time spent on learning. The basic idea is inspired by strategy adaptation methods in evolutionary computation (Smith and Fogarty, 1997; Eiben et al., 1999). The learning time is viewed as a strategy parameter that is adjusted on population level. As a basis for adaptation, different numbers of iterations have to be explored. Therefore, the learning time is described by a random variable. The maximum number of iterations an offspring is allowed to learn is a realization of this random variable, whose expectation is subject to adaptation: The expected learning time in the upcoming generations is shifted towards the learning time that gave good results recently.

We proceed with the presentation of a straightforward algorithm for evolutionary structure optimization of neural networks and with the detailed explanation of the adaptation scheme for the learning time. In section 3, we present experimental results showing the dynamics of the learning times induced by our adaptation scheme as well as the improved optimization performance. The paper ends with concluding remarks.

2 LEARNING TIME ADAPTATION IN STRUCTURE OPTIMIZATION

In the main part of this section, we introduce the adaptation scheme for the learning time. Beforehand, we describe a simple algorithm for structure optimization of neural networks, which serves as a testbed for our method. However, the proposed adaptation scheme is designed to work well with almost any of such algorithms.

2.1 EVOLUTIONARY FRAMEWORK

We use an evolutionary algorithm for structure optimization of neural networks inspired by Angeline et al. (1994). The search space contains all arbitrarily connected feed-forward neural networks. The validity of the architectures is the only constraint, i.e., each hidden neuron has to lie on a path from at least one input to at least one output neuron. Each of the μ individuals in the population encodes the architecture and the weights of one neural network by means of a direct encoding scheme. Prior to the first generation, the individuals are randomly initialized and assigned a fitness value, depending on the target problem. In every generation, each parent produces one offspring by reproduction and mutation. One out of five problem specific mutation operators is randomly chosen and applied: adding and deleting single neurons and single connections as well as disturbing each weight normally distributed with zero mean and standard deviation σ; connections with a lower weight are removed with an increased probability (Braun, 1997). Thereafter, each individual is trained using the iRprop$^+$ learning method (Igel and Hüsken, 2002), an improved version of the well known Rprop algorithm (Riedmiller and Braun, 1993). Learning starts with the genetically encoded weight configuration. After learning the modified weights are inherited, i.e., coded back onto the individual's genome, following the Lamarckian paradigm, which is very efficient for technical purposes. Finally, the individual's fitness is evaluated and the parent population of the next generation is determined by means of EP-tournament selection (Fogel, 1995).

2.2 ADAPTATION OF THE LEARNING TIME

The aim of adjusting the learning time is to choose its value such that in the next generations a maximum fitness improvement relative to the costs can be expected. This task is similar to the adaptation of the strategy parameters of an evolutionary algorithm (e.g., population size, mutation rates, ...), which is a key concept in evolutionary computation (Smith and Fogarty, 1997; Eiben et al., 1999). Our scheme is inspired by these methods, in particular by the covariance matrix adaptation (CMA) evolution strategy (Hansen and Ostermeier, 2001). The adjustment of the learning time is based on the heuristic that learning times that have given good results recently will also perform well in future generations.

The learning strategy is altered every G generations; this interval is called *adaptation cycle*. For the adaptation of the strategy (i.e., the average learning time in the g^{th} adaptation cycle), it is necessary to explore the space of possible strategies and to compare the improvements achieved by different strategies. There has to be a variety of different learning times for the individuals instead of the same for all of them, in order to estimate the efficiency of different strategies in every adaptation cycle to find out the putative best strategy. The variable $\tau_i^{(g)} \in \mathbb{N}_0$ denotes the learning time of the i^{th} individual in the g^{th} adaptation cycle.

One possibility for the adaptation would be to understand learning as some kind of operator. For each individual, one operator from a set of learning operators with different learning times is applied and rated depending on the fitness gain. The probability of these operators to be applied in the next adaptation cycle is

adjusted based on the rating (Davis, 1989). In the domain of evolutionary optimization of learning systems, such a procedure was successfully applied by Igel and Kreutz (1999, 2001) for the adaptation of the probabilities to apply different mutation operators. In our context, this procedure is not suitable, because the number of different operators needed to represent the various learning times would be too high. Further, the learning operators strongly differ in their computational costs. To allow for an ongoing probability adaptation, none of these operators should be allowed to become extinct. Hence, even when some expensive operators are not suitable in the current phase of optimization, they have to be applied every now and then and may dominate the average computational complexity. Simulations support this assumption.

Instead of the operator approach, we adapt the learning time more directly. To explore different learning strategies, the value of $\tau_i^{(g)}$ is drawn independently for each individual from a Poisson distribution with the expectation $m^{(g)}$.[1] The parameter $m^{(g)}$ is subject to adaptation, which permits a smooth adjustment of the learning periods. The expectation of the learning time in adaptation cycle $g+1$ is given by

$$m^{(g+1)} = \max\left[(1-\gamma)m^{(g)} + \gamma\,\tilde{\tau}^{(g)}, m_{\min}\right]. \quad (1)$$

The variable $\gamma \in [0,1]$ determines the influence of $\tilde{\tau}^{(g)}$, the value which would have been the most suitable choice of $m^{(g)}$ in the last adaptation cycle. In the end, (1) is a weighted average over the whole history, but the influence of past generations is exponentially suppressed depending on γ. This weighted average is similar to the evolution path in the CMA evolution strategy. The lower bound $m_{\min}$ in (1) enables a minimum diversity of the learning times to allow for continuing adaptation.

Now one is only left with the estimation of $\tilde{\tau}^{(g)}$. The efficiency of learning for τ iterations is measured by the benefit $B^{(g)}(\tau)$ (Tuson and Ross, 1998), normalized to the costs of learning $c(\tau)$ as proposed by Igel and Kreutz (1999):

$$B^{(g)}(\tau) = \frac{1}{N_\tau^{(g)}} \sum_\iota \max\left[\frac{\phi(\text{parent}(\iota)) - \phi(\iota)}{c(\tau)}, 0\right]. \quad (2)$$

The sum runs over all $N_\tau^{(g)}$ offspring ι in adaptation cycle g that have been trained for τ iterations; $\phi(.)$ assigns each individual a fitness value. As an alternative to (2), one might relate $B^{(g)}(\tau)$ only to the fitness gain achieved by learning, i.e., the difference of the offspring's fitness before and after learning replaces the numerator in (2). However, (2) has empirically proven to be more efficient, as it allows for evaluating the number of iterations in the context of mutations. For instance, it is able to take into account the time necessary to counterbalance mutational disturbances.

The computational costs $c(\tau)$ depend on the implementation of the feed-forward neural network. For simplicity, we utilize an approximation. It takes roughly twice as much time to calculate the gradient of the network error with respect to all weights than to compute the network's error itself (Rummelhart et al., 1986): First, the input is "propagated forward" through the network and thereafter it is "propagated backward" through it. Additionally, one "forward-propagation" has to be performed after the last iteration of learning to calculate the individual's fitness $\phi(\iota)$. As we use "propagations" as the unit of the costs, we set

$$c(\tau) = 2\tau + 1 \quad . \quad (3)$$

The learning time $\tilde{\tau}^{(g)}$ should be the time for which the improvements have been maximal in the near past and therefore might also be in the near future. This seems to be fulfilled for $\tilde{\tau}^{(g)} = \arg\left[\max_\tau \left\{B^{(g)}(\tau)\right\}\right]$. However, this might not be optimal, as in the next generations not only learning times equal to $\tilde{\tau}^{(g)}$ are applied, but also learning times randomly drawn from a Poisson distribution. In the limiting case of an isolated maximum of $B^{(g)}(\tilde{\tau}^{(g)})$, learning times $\tau_i^{(g+1)} = \tilde{\tau}^{(g)}$ would yield a maximum improvement, but slightly differing learning times would mainly lead to an exploration of bad strategies. Therefore, we do not consider the maximum of $B^{(g)}(\tau)$, but the maximum of

$$b^{(g)}(\tau) = \sum_{\tau'=0}^{\infty} B^{(g)}(\tau') \cdot \frac{\tau^{\tau'}}{\tau'!}\,\mathrm{e}^{-\tau} \quad , \quad (4)$$

the convolution of $B^{(g)}(\tau)$ with the Poisson distribution with mean τ. The value of $b^{(g)}(\tau)$ is an estimation of the expected improvement in the case of $m^{(g)} = \tau$, as the distribution of learning times is taken into account. As a side effect, the convolution yields a smoothing of $B^{(g)}(\tau)$, which makes the evaluation of the benefit more robust. Finally, the estimation of the optimal learning time is given by

$$\tilde{\tau}^{(g)} = \arg\left[\max_\tau \left[b^{(g)}(\tau)\right]\right] \quad . \quad (5)$$

[1] The Poisson distribution of $\tau_i^{(g)} \in \mathbb{N}_0$ is given by $p_{\tau_i^{(g)}} = \frac{(m^{(g)})^{\tau_i^{(g)}}}{\tau_i^{(g)}!}\,\mathrm{e}^{-m^{(g)}}$. As the expectation as well as the variance are equal to $m^{(g)}$, the width of the distribution increases with its expectation.

In (4) the case of non-instantiated learning times, i.e., times that have not been sampled, has to be considered, as for the corresponding values of $B^{(g)}(\tau)$ no estimations are available. This is particularly true for learning periods that strongly deviate from the mean $m^{(g)}$, but by chance it can also occur quite close to $m^{(g)}$. If information about longer and shorter learning times is available, we substitute missing values by means of linear interpolation. Otherwise, we make the most conservative assumption of $B^{(g)}(\tau) = 0$.

One might argue that due to the adaptation of the learning time, some new parameters are introduced and have to be chosen. However, these new parameters are, compared to the learning time, intuitive and therefore much easier to choose. Further, they are not as problem dependent as the fixed learning time parameter is. Therefore, although the absolute number of parameters has increased, the choice of their values has become easier and more robust. In addition, the online adaptation of the learning times adds a new quality to the algorithm, as dissimilar learning times can be realized in different stages of optimization.

3 EXPERIMENTAL VERIFICATION

In the following, we empirically show how the adaptation of the learning times improves the efficiency of the optimization of neural networks. Additionally, we consider the adaptation dynamics of the learning time, which provide insights into the different roles of learning and evolution.

All results presented stem from 50 independently initialized trials per setting. The population size was $\mu = 20$, the tournament size $q = 5$, and the standard deviation for the weight mutations $\sigma = 0.05$. Standard parameters were chosen for the iRprop$^+$ algorithm: $\eta^+ = 1.2$, $\eta^- = 0.5$, $\Delta_0 = 0.05$, $\Delta_{\min} = 10^{-8}$, and $\Delta_{\max} = 50$. The hidden neurons of the neural network had the sigmoidal activation function $f(x) = x/(1 + |x|)$ and the output neurons had linear ones. Adaptation of the learning time was conducted every second generation ($G = 2$) with the adaptation rate $\gamma = 1/4$ and a lower bound of $m_{\min} = 1$. This parameter setting should serve as an example rather than as a recommendation for the optimal choice. Finally, we explore different values for the initial expectation of the learning time $m^{(g)}$ and compare the results with an algorithm with the same value as a constant learning time.

3.1 SAMPLE PROBLEMS

We use established neural network benchmark problems for the formulation of different types of structure optimization tasks, in order to observe different dynamics of the learning time.

3.1.1 Smallest Network for 6-Parity

The task is to find the neural network with the lowest number of weights $n^{(\text{dof})}$ that completely solves the classification problem 6-parity. The network is to decide the parity of a bit-string of length 6 (i.e., whether the number of "1" in this string is even or not). Learning was conducted with all $2^6 = 64$ possible patterns, the target values were given by "0" and "1", and the mean squared error $E^{(\text{mse})}$ was used for training. The individual's fitness consists of three addends, the classification error $E^{(\text{class.})}$, the number of weights of the network, and the mean squared error, evaluated using the whole data set:

$$\phi = \nu E^{(\text{class.})} + n^{(\text{dof})} + E^{(\text{mse})} \quad . \tag{6}$$

The parameter $\nu = 10^6$ was chosen with respect to the expected magnitudes of the different addends to describe the previously mentioned importance of the different optimization goals. Normally, it holds $\nu E^{(\text{class.})} \gg n^{(\text{dof})} \gg E^{(\text{mse})}$. The inclusion of $E^{(\text{mse})}$ yields ongoing optimization, even in the absence of improvements with respect to the main goals. However, reduction of the mean squared error may contribute to better classification performance of the offspring and may prepare the deletion of non-relevant weights.

3.1.2 Prediction of Sunspots

The task is to optimize a neural network with respect to its ability to predict the number of sunspots s_{t+1} in the next year, based on their number in the current and previous years. Figure 1 shows the annual number of sunspots. The states s_t, s_{t-1}, s_{t-3}, and s_{t-7}, each of them normalized to $[0.2, 0.8]$, serve as the input for the network.

In contrast to 6-parity, the fitness is only given by the mean squared error of the network on a particular data set. We distinguish between two different settings:

sunspot$^{(=)}$: The same data points (i.e., the prediction of all years from 1708 to 1938) are used for learning and for fitness evaluation.

sunspot$^{(\neq)}$: The data points used for learning and fitness evaluation are disjoint subsets of the sunspot data set. The prediction error in the years 1708 to

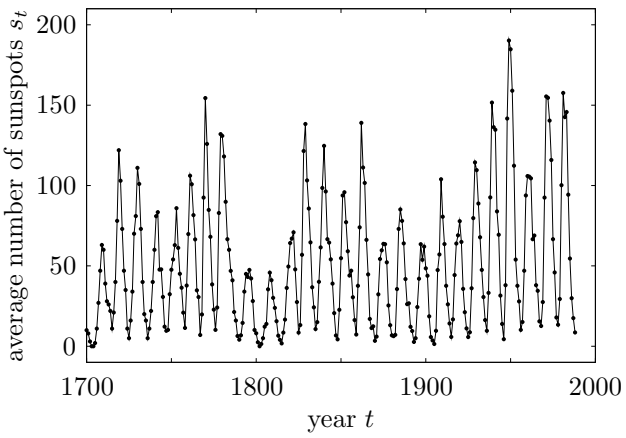

Figure 1: Evolution of the average number of sunspots in the last 300 years. The data are provided by the SIDC (`http://sidc.oma.be`).

1784 is used for learning and the weight configuration of the network with the best error over the years between 1785 and 1861 is inherited (hold-out samples). The individual's fitness is given by the mean squared error in the years between 1862 and 1938.

3.2 DISCUSSION OF RESULTS

The results of the structure optimization for the different problems are summarized in table 1. It becomes obvious that for both algorithms—with and without learning time adaptation—the achieved fitness strongly depends on the initial learning time. In all cases where the differences between both algorithms with equal initial learning time are statistically significant, the algorithm with adaptation performs better than the one with constant learning time; in the other cases we must assume both algorithms to be equally good. In particular, for short initial learning periods the advantage of the adaptation becomes obvious. Here, the adaptation can operate faster and therefore select more suitable strategies, where the speed of adaptation is defined as the change of $m^{(g)}$ per cost unit. The other way round, both algorithms perform equally well for long initial learning times, as the adaptation speed is slow. In 6-parity, after a very high number of generations and for moderate learning times both algorithms have achieved equally good results, as both have "converged". As can be seen from the upper diagram in figure 2, the algorithm with learning time adaptation reaches this state earlier.

Summarizing, learning time adaptation yields better results in most cases and is equally good in the remaining ones. In the following, we consider the learning strategies that emerged for the different problems.

3.2.1 Smallest Network for 6-Parity

The upper diagram in figure 2 shows fitness curves that emphasize the results in table 1. The lower diagram depicts the evolution of the adapted learning times, showing very clear directions of adaptation. The evolution of the learning times and the fitness trajectories in conjunction with (6) allow an identification of three phases of optimization.

1. Initially, the increase of the learning time coincides with a decrease of the classification error.

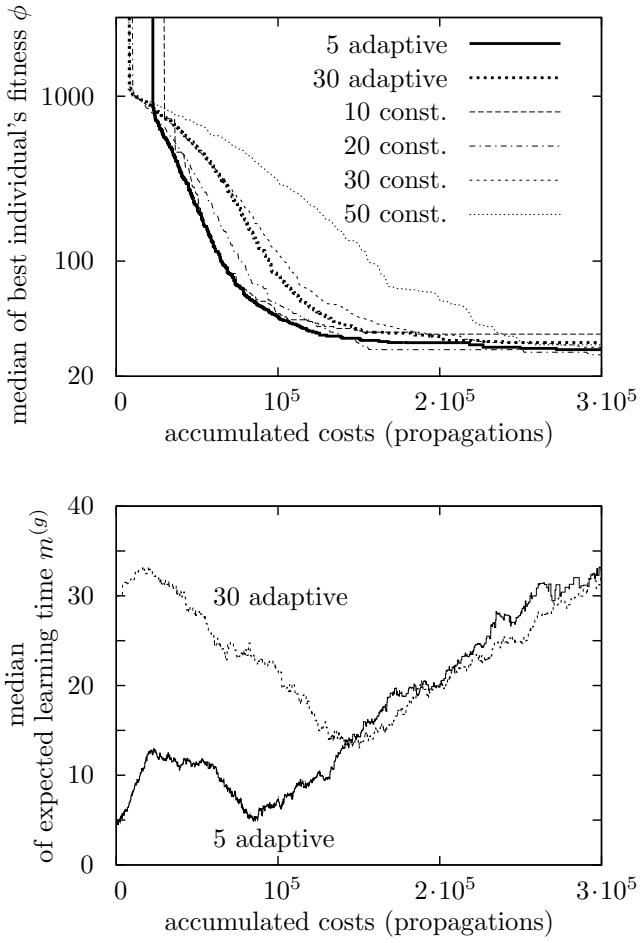

Figure 2: Evolution of the fitness and the adapted learning time for the 6-parity problem, where "adaptive" and "const." refer to the algorithms with and without adaptation of the learning times. The preceding numbers denote either the initial or the constant learning times.

Table 1: Median of the best individual's fitness after a given number of propagations. Depending on the context, the first column shows either the constant learning time or the initial value of the expectation of the learning time. The numbers in parentheses give the number of propagations after which the fitness values were measured. In rows marked with "⋆", the difference between constant and adaptive learning time is statistically significant, Wilcoxon rank sum test (Wilcoxon, 1945), $p < 0.05$.

(initial)	6-parity (10^5)		6-parity ($3{\cdot}10^5$)		sunspot$^{(=)}$ (10^5)		sunspot$^{(\neq)}$ ($2{\cdot}10^5$)	
learning time	constant	adaptive	constant	adaptive	constant	adaptive	constant	adaptive
5	15652.5	46.0 ⋆	15647.1	29.0 ⋆	133.1	84.7 ⋆	139.9	130.3 ⋆
10	46.0	41.5 ⋆	33.0	26.0 ⋆	108.4	83.6 ⋆	144.5	129.2 ⋆
20	59.5	56.0	28.0	28.5	89.7	81.3 ⋆	142.4	128.2 ⋆
30	109.5	83.0 ⋆	30.0	32.0	78.6	74.1 ⋆	137.6	131.8 ⋆
50	323.5	286.5 ⋆	34.0	32.5	74.4	74.2	135.9	138.2

As the networks in the initial population are quite large, it is likely that the algorithm finds suitable architectures within this population to solve the classification problem. Therefore, learning seems to be the driving force in this phase and, in particular for short learning times, the amount of learning is increased compared to the amount of architectural changes.

2. The second phase starts after ϕ has dropped below approximately 1000, i.e., the classification problem is solved and $n^{(\mathrm{dof})}$ becomes the dominating addend in (6). Hence, the amount of the numerator in (2) is dominated by structural changes, i.e., mutations. As the denominator increases with the learning time, it is beneficial to learn only for a short period. In this phase, the task of learning might be to counterbalance mutational "damages" with respect to solving the classification task, rather than to achieve further improvement in the setting of the weights.

3. The third phase is characterized by a continuing increase of the learning time. As a further reduction of $n^{(\mathrm{dof})}$ without worsening the classification performance becomes more and more unlikely, the largest improvement can be gained by reducing the mean squared error. This can efficiently be realized by long learning periods. As the mean squared error is in the order of magnitude of 10^{-4}, these improvements are not visible in figure 2. Reductions of the architecture are rarely observed, maybe prepared by the fine tuning of the weights, as removing of single connections takes place with respect to the corresponding weight (cf. section 2.1).

From this analysis of the three phases, it becomes clear that an algorithm with constant learning time can barely cope with them efficiently, so that learning time adaptation becomes necessary.

3.2.2 Prediction of Sunspots

The results of the experiments using the sunspot data set are shown in figure 3. Interestingly, the evolution of learning times look completely different depending on whether one (sunspot$^{(=)}$) or three different (sunspot$^{(\neq)}$) data sets are involved. In the first case, long learning seems to be a good strategy, as overfitting does not have to be taken into account. A slight reduction of the number of iterations takes place only in the first generations of the trials with an initial learning time of 20, moving the balance towards evolution (see lower left plot in figure 3). This example also shows that the adaptation speed is limited due to the width of the Poisson distribution and the damping in (1) and that the longer the learning time the slower the adaptation.

In sunspot$^{(\neq)}$, where different data sets are used for learning and fitness evaluation, the effect of overfitting becomes important. The lower right plot in figure 3 shows different behaviors depending on the initial learning times. If the adaptive algorithm starts with 5 iterations, the learning time increases during the first generations. This happens because overfitting has not occurred yet and learning is efficient, as improvements on the learning data set coincide with improvements on the fitness data set. However, when after approximately 10^4 propagations overfitting becomes a problem, the learning time strongly decreases and stays at its minimum value $m_{\min}$. In this phase learning would only lead to overfitting and therefore to an increase of the error on the fitness data set, i.e., a worsening of fitness. Progress can only be expected to occur due to fortuitous mutational variations. When the adaptive algorithm is initialized with 20 iterations, the learning

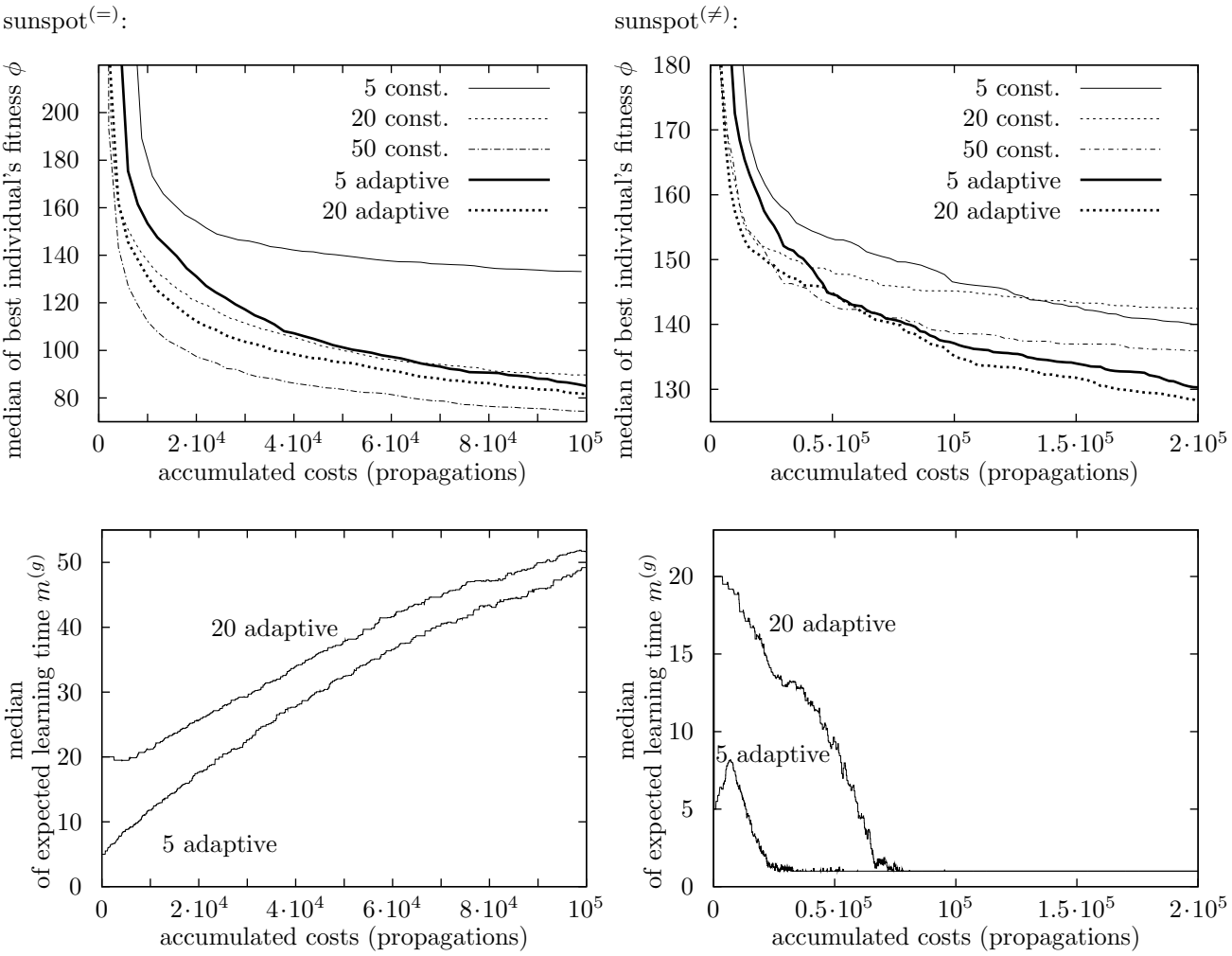

Figure 3: Evolution of the fitness and adapted learning time for the sunspot prediction problem.

time seems to be too long to avoid overfitting right from the start and decreases nearly monotonically to its minimum value.

Again, different learning strategies, as well as a change of the strategy during the course of evolution, can be observed. This explains the significant improvement of the optimization performance by means of adaptive learning time control compared to a constant learning time in all phases of the optimization.

4 CONCLUSION AND OUTLOOK

The number of learning iterations in algorithms that combine learning and evolution is a crucial parameter for the efficiency of the optimization. We have empirically shown that the right learning strategy indeed strongly depends on the problem and on the course of evolution. This may not only include the choice of the initial learning time, but also any schedule for modifying the learning period during optimization. Unfortunately, the best strategy is usually not known *a priori*.

Therefore, we proposed an algorithm that adapts the learning time similarly to strategy parameters of pure evolutionary algorithms. The main idea is to randomize the learning time, to evaluate the fitness improvements resulting from different learning periods, and then to adapt the expectation of the learning time. In a number of examples from the domain of evolutionary optimization of neural networks the adaptation method works in an intuitive way. Moreover, in all examples the optimization is improved due to the better choice of the ratio between evolution and learning.

Based on these findings, our answer to the question of the right choice of the learning time is to start with a small one and use an adaptation scheme to adjust it to the given task. The answer to the question, whether learning is necessary at all, can be left to the evolutionary process itself.

This study is a step towards an evolutionary optimization algorithm for neural networks without any crucial parameters. In further investigations, we plan to combine the control of the learning time with adjustment of the operator probabilities (Igel and Kreutz, 1999, 2001) and adaptation of the population size.

Acknowledgement

We would like to thank the BMBF, grant LOKI, number 01 IB 001 C, for their financial support of our research.

References

Angeline, P. J., G. M. Saunders, and J. B. Pollack (1994). An evolutionary algorithm that constructs recurrent neural networks. *IEEE Transactions on Neural Networks 5*(1), 54–65.

Braun, H. (1997). *Neuronale Netze: Optimierung durch Lernen und Evolution*. Springer-Verlag.

Davis, L. (1989). Adapting operator probabilities in genetic algorithms. In J. D. Schaffer (Ed.), *Proceedings of the 3rd International Conference on Genetic Algorithms*, pp. 61–69. Morgan Kaufmann.

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

Fogel, D. B. (1995). *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*. IEEE Press.

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

Igel, C. and M. Hüsken (2002). Empirical evaluation of the improved Rprop learning algorithm. *Neurocomputing*. In press.

Igel, C. and M. Kreutz (1999). Using fitness distributions to improve the evolution of learning structures. In *Congress on Evolutionary Computation (CEC '99)*, Volume 3, pp. 1902–1909. IEEE Press.

Igel, C. and M. Kreutz (2001). Operator adaptation in structure optimization of neural networks. In L. Spector, E. D. Goodman, A. Wu, W. B. Langdon, H.-M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. Garzon, and E. Burke (Eds.), *Genetic and Evolutionary Computation Conference (GECCO 2001)*, pp. 1094. Morgan Kaufmann.

Mayley, G. (1996). Landscapes, learning costs and genetic assimilation. *Evolutionary Computation 4*(3), 213–234.

Nolfi, S. and D. Parisi (1999). Learning and evolution. *Autonomous Robots 7*(1), 89–113.

Riedmiller, M. and H. Braun (1993). A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *Proceedings of the IEEE International Conference on Neural Networks*, pp. 586–591. IEEE Press.

Rummelhart, D. E., G. E. Hinton, and R. J. Williams (1986). Learning internal representations by error backpropagation. In D. E. Rummelhart, J. L. McClelland, and the PDP Research Group (Eds.), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Volume 1, pp. 318–362. MIT Press.

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

Tuson, A. and P. Ross (1998). Adapting operator settings in genetic algorithms. *Evolutionary Computation 6*(2), 161–184.

Wilcoxon, F. (1945). Individual comparison by ranking methods. *Biometrics Bulletin 1*, 80–83.

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