Evolutionary optimised ontogenetic neural networks with incremental problem complexity during development

Bernhard Sendhoff

2000

Preprint:

This is an accepted article published in Congress on Evolutionary Computation (CEC). The final authenticated version is available online at: https://doi.org/[DOI not available]

published in:
Congress on Evolutionary Computation (CEC 2000), San Diego, IEEE Press, pages 1443-1450, 2000.

Evolutionary optimised ontogenetic neural networks with incremental problem complexity during development

Bernhard Sendhoff
Future Technology Research Division
HONDA R&D Europe (Deutschland) GmbH
63073 Offenbach/Main, Germany

Abstract- In order to optimise unconstrained, large neural network structures with evolutionary algorithms indirect encodings have been proposed. However, if the evolutionary process is combined with network learning, which is sensible both with respect to technical applications in dynamical environments and to the biological paragon, a way has to be found to combine learning with the evolutionary optimisation of such large structures. Utilising the development of neural systems during ontogeny seems a logical starting point for the realization of a step by step learning in networks. Furthermore, the combination of network growth during the developmental phase with an incremental problem complexity might allow the optimisation of large network structures *together with learning*. In this paper we will propose a model to simulate such a combined approach and apply it to the problem of time series modelling. By introducing several measures for the transfer of information from one developmental step to the next, we will be able to quantitatively analyse the behaviour of our proposed model.

1 Introduction

The determination of the structure of neural systems holds the key not only to their successful application in several domains, but also to a better understanding of information processing in neural systems. However, structure optimisation of neural networks is a formidable task, even if we consider only a few neurons, the number of possible connectivity patterns becomes astronomically high. Therefore, it seems logical to utilise evolutionary algorithms for this task. The £rst approaches [1, 2] used a direct encoding of the network structure and together with some improvements in the genetic operators, they are still among the most feasible methods, when our aim is to design one particular neural network of moderate size for one speci£c problem. In order to be able to optimise large network structures, i.e., in order to achieve good scaling of the algorithm, the idea of an indirect encoding of neural systems was introduced by Kitano [3] and by Gruau [4]. At £rst these networks were restricted to binary connections between the neurons. If such restrictions are removed, the problem arises that for large neural networks it is very dif£cult to achieve successful learning in particular if no speci£c topology is £xed apriori[1].

Of course in nature, there is a constant interaction between genetically speci£ed information relating to the formation of the neural structure, which might even include the initial speci£cation of synaptic strength [5], and information generated due to learning or spontaneous activity patterns [6, 7]. During this ontogenetic and epigenetic developmental phase of the neural system, all information sources act together in a very intrinsic way. Several systems have been proposed on how the ontogenetic phase can be used for the formation of arti£cial neural structures [8, 9, 10]. Often these approaches are coupled to a Lamarckian type of evolution where learned behaviour is transferred to the next generation. Besides not being "biological", Lamarckian evolution has also received some criticism with respect to the robustness of solutions [11]. On the other hand, analysis with regard to the optimisation of the learning capability of neural systems has shown that a kind of averaged Lamarckian evolution can be bene£cial even in dynamic environments [12]. In order to solve the above mentioned problem of learning in large neural systems, the developmental phase of the network structure can be exploited to utilise the information learned and stored in the connection weights in an earlier developmental step to achieve a better starting point for the network after further growth. A model to study such an interaction between learning and evolution by transferring connection weights from smaller networks (developmental step i) to larger structures (developmental step $i+1$) after learning was proposed by Sendhoff and Kreutz [13]. In this model, however, the task during the whole developmental phase of the network remained the same. Firstly, this is certainly not the case in biological systems, where the task complexity grows with the ability of the neural structure, and secondly, it seems that also for technical applications, a combined growth process would be bene£cial. In this paper, we will describe an extension of the model which incorporates such a combined growth process and apply it to the modelling of a chaotic time series.

The paper is organised as follows: In the next section we will describe our idea of a combined growth process and in Section 3, we will outline the model, which will be applied to the task of time series modelling in Section 4. We will discuss our £ndings in Section 5.

[1]With respect to structure optimisation there is no reason why a multilayer topology should be forced on the neural structure.

2 Evolution, development and learning

The interaction between evolution and learning is intrinsic both for natural as well as for arti£cial systems. Firstly, the capability of the network to adapt to changing environments, i.e, to learn, is the product of evolution. Although this has hardly been addressed in the context of arti£cial systems, it nevertheless is very important in particular for technical applications in dynamic environments, see e.g. [12]. On the other hand, learning in¤uences the survival and reproduction probabilities and therefore indirectly, as opposed to the direct Lamarckian inheritance, the course of evolution – the so-called Baldwin effect. Introducing a developmental phase, this interaction between evolution and learning[2] becomes even more complicated because both information sources for the structure formation overlap. In the neurobiological context such analysis has also recently received increasing interest [14], because it can be argued that in order to understand the architecture and the function of the brain it is inevitable to look at both its phylogenetic and ontogenetic design process [5]. With respect to arti£cial systems, the introduction of a developmental phase might be the key to the ef£cient optimisation of large neural structures *including learning*. The main two ideas behind the model proposed in this paper are as follows, see also Figure 1:

- The evolutionary optimisation of a growth process of the neural network, while transferring the weights from smaller neural networks to a speci£c part of larger neural networks in each step. The network learns a speci£c task in each of these developmental steps about which information is stored in the connection weights.
- The complexity[3] of the problem which the network learns in each developmental step grows with the network size which is assumed to scale with its potential performance. The biological counterpart is the observation that learning occurs in speci£c patterns, i.e., actively tasks are searched for which are *just* achievable.

In the next section we will introduce a speci£c model which incorporates these two properties. However, beforehand, it should be noted that of course one major task remains and that is the decomposition of the problem into sub-problems which the network can learn *and* about which information can be meaningfully transferred from one developmental step to the next. The long-term aim has to be to achieve such a problem decomposition in a self-organised manner; research in the area of active learning of neural networks might be a sensible starting point [15]. In the application in Section 4 this problem decomposition has been done manually and we will come back to this point in the discussion.

[2]In the framework of biological systems, we refer to learning to describe all phenomena which relate to the formation of the neural structure/synaptic strength and which are not purely genetic based.

[3]The notion of the complexity of a task is not well de£ned and somewhat arbitrary, here we will understand it in the context of "dif£cult to learn for a standard arti£cial neural network", see Section 4.

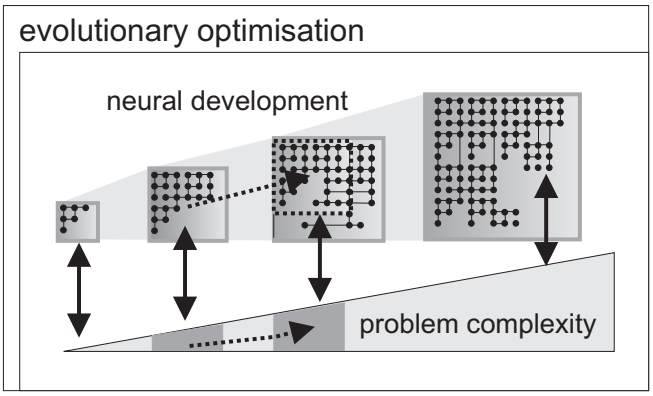

Figure 1: Schema of the combined approach of neural development with incremental problem complexity and intermediate weight transfer from one developmental step to the next.

3 Outline of the model

In order to analyse the potential of intermediate learning and subsequent transfer of information in the form of the connection weights, we need a model which incorporates the genotype phenotype map as a developmental process in the evolutionary optimisation. The recursive encoding method of neural networks, which is described in more detail in [16], is based on the grammar encoding by Kitano [3]. In addition to the speci£cation of the structure, an initialisation of the weights is possible which is "hidden" in the developmental process.

3.1 The recursive encoding scheme

In the recursive encoding, the connection matrix of a feed-forward network without a layered structure is developed in different stages. The elements of the connection matrix de£ne the initial weights between neurons; in the case of a zero element the neurons are not connected. Since the matrix is feed-forward, only the upper triangular part is needed for the speci£cation of the network. The diagonal elements de£ne the threshold values for each neuron. The developmental process of the matrix is speci£ed by a mapping from a £rst vector S_C (small chromosome) with integer elements to a second vector L_C (large chromosome), also with integer elements. In each step, each element of the connection matrix is replaced by a 2×2 matrix of new elements. In Figure 2 the growth of the connection matrix is shown. The following notation will be used for the recursive encoding:

$a^k_{\mu\nu}$	element at position (μ, ν) of the connection matrix in the kth recursion step. If the element is non zero, it represents the initial weight between neuron μ and ν ($a^k_{\mu\nu} \in \{0, \ldots, N_{sym}\}$).
R	the number of recursion steps
k	the recursion step, ($k = 1, \ldots, R$)
N_{sym}	maximum integer number permitted in (S_C, L_C)
$N(a^k_{\mu\nu})$	the index of the £rst element in S_C,

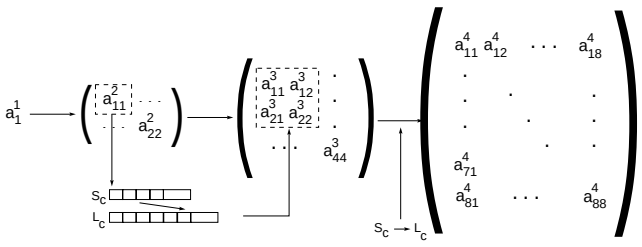

Figure 2: Scheme of the recursive development of the connection matrix up to a size of 8×8. In each step, each element is replaced by a 2×2 matrix via the mapping $S_C \rightarrow L_C$.

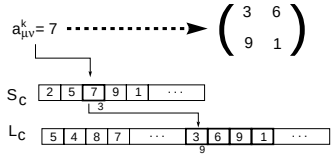

Figure 3: One element is replaced by four elements in the recursion step via the small chromosome $S_C \rightarrow$ large chromosome L_C mapping.

	which is identical to $a^k_{\mu\nu}$
d_{S_C}	dimension of the vector S_C (small chromosome)
d_{L_C}	dimension of the vector L_C (large chromosome)
s_i, l_i	elements of the vector S_C and L_C

At each recursion step k, the index $N(a^k_{\mu\nu})$ of the £rst element in S_C, which is identical to the connection matrix element $a^k_{\mu\nu}$, is determined; for example, index $N(a^k_{\mu\nu} = 7) = 3$ in Figure 3 (a). The element is then replaced by the four elements at the positions

$$\begin{aligned} \big(& 4 \cdot (N(a^k_{\mu\nu}) - 1) + 1, \qquad 4 \cdot (N(a^k_{\mu\nu}) - 1) + 2, \\ & 4 \cdot (N(a^k_{\mu\nu}) - 1) + 3, \qquad 4 \cdot (N(a^k_{\mu\nu}) - 1) + 4 \big) \end{aligned} \quad (1)$$

in the large chromosome L_C. Figure 3 (a) shows the replacement of an element $a^k_{\mu\nu} = 7$ by the four elements $(3, 6, 9, 1)$ at the positions $(9, 10, 11, 12)$. Should $a^k_{\mu\nu}$ not be identical to any element of S_C, it is replaced by four so-called terminal symbols (in the notation of integer strings, the most convenient choice is zero). The terminal symbol (zero) denotes that no connection exists between neuron μ and ν. The terminal symbol is in turn always replaced by another four terminal symbols in a recursion step. Finally, the connection matrix is simpli£ed by deleting all neurons which do not contribute to the network output. Thus, the main components of the recursive encoding are:

- In each step, each element of the connection matrix is replaced by a 2×2 matrix whose elements are speci£ed by equation (1).
- If the element is not identical to any entry of S_C, the elements of the 2×2 matrix are given by $(0, 0, 0, 0)$.
- In each step, 0 is replaced by $(0, 0, 0, 0)$.
- R steps are carried out.

In its simplest form the recursive encoding develops a connection matrix from a vector S_C with elements $s_i \in \{1, \ldots, N_{sym}\}$ and a vector L_C with elements $l_i \in \{1, \ldots, N_{sym}\}$ with the described mapping. The connection matrix is then translated into a neural network in the following way. If $a^R_{\mu\nu} = 0$, neuron μ and neuron ν are not connected. All other integer values $a^R_{\mu\nu} \in \{1, \ldots, N_{sym}\}$ are mapped to an interval $[-\delta, \delta]$. The activation function $tanh(x)$ is used for all neurons.

This scheme has been successfully applied to network optimisation for time series predictions by Sendhoff and Kreutz [17, 16]. Furthermore, extensions of the basic scheme to guarantee certain properties of the encoding, like completeness, have been proposed, i.e. additional negative integers are introduced, which map to the zero symbol *after* the replacement process [16].

3.2 Utilising pre-learned weights in the recursive encoding

In order to be able to transfer learned weights from neural structures in earlier developmental steps to larger neural networks in later steps, it is necessary to extend the replacement scheme. Each connection matrix during development (i.e. all three matrices in Figure 2) represents a neural network at different *ontogenetic stages* which can learn speci£c tasks. In order to transfer information in this scheme it must be possible to include the network connection matrix after simpli£cation and after the learning process at step $(k-1)$ in the connection matrix at step k. Two different methods can be used to achieve this; see Figure 4. Firstly, in each step the connection matrix is built in the original form using the *basic* recursive encoding method described in the last section including the initial weight speci£cation. In addition, the network matrix from the previous ontogenetic stage is copied to the upper left part of the new matrix. Therefore, replacing some of the genetically determined initial weights by the pre-learned weights of the previous ontogenetic stage. Alternatively, the network matrix is split up into four parts of equal size, which are copied to the four edges of the connection matrix at step k. In this way, the network grows *in the middle*, which means that input and output neurons and their connections are not changed during the growth process. Only hidden neurons are added in the developmental phase. The differences both with respect to performance and to the qualitative behaviour of the interaction measures, between the schemes Figure 4 (left) and (right) are however marginal. In Section 4 only the left scheme will be used. Using one of these replacement schemes in the developmental process an overlap between the learning process (connection matrix which is copied from the last growth step) and the genetically determined structure and weight initialisation is therefore realized. Thus, it is possible to let each network at each developmental step learn a

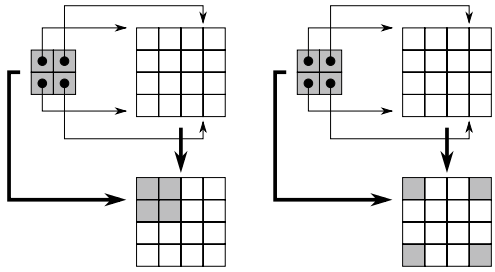

Figure 4: The extension of the matrix replacement scheme of the basic recursive encoding to include learning of networks represented by matrices of earlier developmental steps. Left: The original replacement scheme is used to build the matrix at step k, however the network connection matrix with *learned* weights from step $(k-1)$ are copied to the 1st quadrant of the new matrix. Right: The four quadrants of the network matrix from step $(k-1)$ are copied to the upper left, lower left, upper right and lower right section of the new matrix.

sub-problem and transfer the information gained at this step to the next-level network structure, as described in Section 2 and depicted in Figure 1. The £nal network is then trained for the problem, which we aimed to solve initially. In Section 4 we will propose how the problem of time series modelling can be decomposed in sub-problems with incremental complexity and analyse the model outlined in this section.

3.3 Measuring the impact of pre-learned weights during development

In the last section we introduced the model for transferring pre-learned weights on a sub-problem to the network structure of the next developmental phase. In order to analyse to what extent this information from learning a problem with lower complexity can be used in the next step, it is necessary to de£ne appropriate measures. In each developmental step, the network has to carry out sub-tasks. The network is trained for 75 cycles[4] and after each replacement step the trained weights are transferred to a part of the enlarged network of the next step. Therefore, after each developmental step the structure and the weights of the network are determined partially by the trained network of the last step and partially by the genetic information contained in the vectors (*chromosomes*) (S_C, L_C). If the matrix is not simpli£ed, i.e., no neurons are removed, one quarter of the matrix originates from the previously trained network and three-quarters from the replacement rules of the step, although in the experiments this relation can be very different. The matrix is forced to grow, i.e., replacement schemes in which the network does not expand in later steps, due to the deletion of neurons, are removed from the population. In this way, the border case, i.e., the network at step $(k+1)$ is identical to the one from step k, cannot occur. Since both the overall network structure and the problem have changed, it is by no means clear to what extent the enlarged network at step $(k+1)$ will bene£t from the fact that part of its weights from network k have already been learned for a somewhat simpler version of the problem. The evolution-learning interaction was analysed by studying the relative in¤uence of the pre-learned weights (from step k) on the prediction error of the network at step $(k+1)$, both before (head start, HS) and after (£nal relation, FR) learning. $\mathcal{E}^{va}_{wt}(c)$ refers to the network error (standard RMSE) after cycle c *with* weight transfer and $\mathcal{E}^{va}_{nt}(c)$ to the network error after cycle c *without* weight transfer on the validation data set.

$$HS = \frac{\mathcal{E}^{va}_{nt}(c=0)}{\mathcal{E}^{va}_{wt}(c=0)} \quad (2)$$

$$FR = \frac{\mathcal{E}^{va}_{nt}(c=75)}{\mathcal{E}^{va}_{wt}(c=75)} \quad (3)$$

Additionally, the "weight difference relation" (WD) between the pre-learned and the genetically determined weights was recorded. The weight difference relation takes the relative changes of the weights induced by the learning process into account. In order to calculate these measures, each individual had to be trained once with weight transfer from previous steps and once without weight transfer. The following notation will be used:

- $\mathcal{M}^{learn}$ contains all index tuples of the connection matrix which refer to weights transferred from the previous network.
- $\mathcal{M}^{gene}$ contains all index tuples of the connection matrix determined by the recursive encoding method.
- d_M is the dimension of the connection matrix
  d^t_M is the dimension of the transferred matrix.
- $a_{\mu\nu}$ and $a^L_{\mu\nu}$ are the matrix elements (weights) before and after learning, respectively.

Now, the weight difference relations can be de£ned as follows:

$$WD_1 = \frac{\text{card}(\mathcal{M}^{gene})}{\text{card}(\mathcal{M}^{learn})} \frac{\sum_{(\mu,\nu)\in\mathcal{M}^{learn}} |a_{\mu\nu} - a^L_{\mu\nu}|}{\sum_{(\mu,\nu)\in\mathcal{M}^{gene}} |a_{\mu\nu} - a^L_{\mu\nu}|}$$

$$WD_2 = \frac{d_M}{d^t_M} \frac{\text{card}(\mathcal{M}^{gene} \cup \mathcal{M}^{learn})}{\text{card}(\mathcal{M}^{learn})} \frac{\sum_{(\mu,\nu)\in\mathcal{M}^{learn}} |a_{\mu\nu} - a^L_{\mu\nu}|}{\sum_{(\mu,\nu)\in\mathcal{M}^{gene}\cup\mathcal{M}^{learn}} |a_{\mu\nu} - a^L_{\mu\nu}|}$$

4 Experiments with Lorenz time series

4.1 Time series prediction and modelling

Time series prediction with arti£cial neural networks is a well established £eld. Usually the task is to forecast future values, e.g. one or several time steps ahead, of a series where knowledge about past states can be used to teach the neural network. Chaotic time series for example from numerical

[4] One cycle corresponds to the presentation of all training and/or test pattern.

solutions of the Lorenz or the Mackey-Glass equations have frequently been used as benchmark problems. From excellent prediction results of a trained neural network, it can however not be deduced that the dynamics of the underlying system has been learned, indeed in most cases this is unlikely. For chaotic time series successful learning of the underlying dynamics can be quanti£ed by measuring invariant properties of the attractor of the neural network and comparing them to the original system, e.g. the Liapunov spectrum and the fractal dimension. Learning the dynamics is dif£cult for purely feed-forward neural networks [18]. The learning algorithm usually has to be modi£ed to include an iteration task in order to achieve good results and the network topology has to be expanded to allow recurrent connections [19, 20, 21].

The iteration task can be seen as a step towards successful modelling of the time series. The neural network receives the system state $\vec{x}(t)$ at time t as input and iterates for m time steps. During the iteration, the network uses the prediction $\vec{\tilde{x}}(t_0 + n\,\Delta t)$ of the systems state as input for the prediction at step $n+1$, $\vec{\tilde{x}}(t_0 + (n+1)\,\Delta t)$. The iteration task is dif£cult, since there are two sources of error. Firstly, during the iteration the model itself constitutes a dynamical system. If the dynamic of the model is very different the predicted orbit will rapidly depart from the true orbit. For example, if the network has a £x-point dynamic it might only need a few iterations to reach this point. This effect can be seen in poorly trained networks for larger iteration times. Secondly, due to the irregularity of the system, the errors of the model in each time step will be ampli£ed even if it perfectly captures the dynamic of the original system.

In the following experiments we will use the Lorenz time series [22]:

$$\frac{dx(t)}{dt} = -y(t)^2 - z(t)^2 - a\,x(t) + a\,F \quad (4)$$

$$\frac{dy(t)}{dt} = x(t)\,y(t) - b\,x(t)\,z(t) - y(t) + G \quad (5)$$

$$\frac{dz(t)}{dt} = b\,x(t)\,y(t) + x(t)\,z(t) - z(t). \quad (6)$$

The differential equation system (4–6) was solved numerically with the Runge-Kutta 4^{th} order method with an integration step $\Delta t = 0.05$ for $N+1 = 10000$ time steps for the parameter setting ($a = 0.25, b = 4.0$, $F = 8.0$, $G = 1.0$). The attractor dimension for this setting is $d_A \approx 2.5$.

The iteration task for $m = 5$ constitutes the £nal task which the neural network has to ful£l after development terminates (the model is denoted by $\vec{M}$, the error of the model is denoted by $\mathcal{E}_n$ for each data point at time $(t_0 + n\,\Delta t)$ and is a function of the predicted state $\vec{\tilde{x}}(t_0 + (n+1)\Delta t)$ and of $\vec{x}(t_0 + (n+1)\Delta t)$.):

$$\begin{aligned}
\vec{\tilde{x}}(t_0 + \Delta t) &= \vec{M}(\vec{x}(t_0)) \quad (7)\\
&\rightarrow \mathcal{E}_1(\vec{\tilde{x}}(t_0 + \Delta t), \vec{x}(t_0 + \Delta t))\\
\vec{\tilde{x}}(t_0 + 2\,\Delta t) &= \vec{M}(\vec{\tilde{x}}(t_0 + \Delta t))\\
&\rightarrow \mathcal{E}_2(\vec{\tilde{x}}(t_0 + 2\,\Delta t), \vec{x}(t_0 + 2\,\Delta t))\\
&\vdots \quad \vdots\\
\vec{\tilde{x}}(t_0 + m\,\Delta t) &= \vec{M}(\vec{\tilde{x}}(t_0 + (m-1)\,\Delta t))\\
&\rightarrow \mathcal{E}_m(\vec{\tilde{x}}(t_0 + m\,\Delta t), \vec{x}(t_0 + m\,\Delta t))\\
\vec{\tilde{x}}(t_0 + (m+1)\,\Delta t) &= \vec{M}(\vec{x}(t_0 + m\,\Delta t))\\
&\rightarrow \mathcal{E}_{m+1}(\vec{\tilde{x}}(t_0 + (m+1)\,\Delta t),\\
&\qquad \vec{x}(t_0 + (m+1)\,\Delta t))\\
&\vdots \quad \vdots\\
\vec{\tilde{x}}(t_0 + N\,\Delta t) &= \vec{M}(\vec{\tilde{x}}(t_0 + (N-1)\,\Delta t))\\
&\rightarrow \mathcal{E}_N(\vec{\tilde{x}}(t_0 + N\,\Delta t), \vec{x}(t_0 + N\,\Delta t))
\end{aligned}$$

The $m = 5$ iteration task can now be decomposed in the following way. After each developmental step the number of iterations is increased by one starting from $m = 1$, which is the usual prediction task. Therefore, the number of developmental steps is £xed during evolution by the number of iterations of the £nal network (here $m = 5$). Thus, incremental problem complexity will be realized in our example by incrementing the number of iterations required from the neural network in each developmental step. The underlying assumption is that information stored in the connection weights of the neural network about the k iteration task can be exploited by the neural structure of the next developmental step for the $(k+1)$ iteration task.

parameter	**value**	**parameter**	**value**
learning rate	0.0009	mom. factor	0.75
max. d_{S_C}	50	max. N_{sym}	50
weight interval	$[1.0, -1.0]$	max. R	7
max. train. time	120 sec.		
μ	20	λ	80
$p_m(S_C)$	$2 \cdot (d_{S_C})^{-1}$	$p_c(S_C)$	0.75
$p_m(L_C)$	$2 \cdot (d_{L_C})^{-1}$	$p_c(L_C)$	0.75
$p_m(C)$	0.05	$p_c(C)$	0.05
$p_m(input)$	0.33	$p_c(input)$	0.75

Table 1: Parameter values for the experiments for the £ve step iteration of the Lorenz time series with the recursive encoding method and back-propagation learning. p_m and p_c denote the probabilities for mutation and crossover for the different chromosomes, S_C, L_C, the coding C and the input weights $input$.

4.2 Experiments for the incremental time series iteration during network growth

In this section, we analysis the interplay between learning and evolution during the developmental phase for the case whereby the networks must perform different tasks (different number of iterations) during their growth process.

In the developmental step k, the k step iteration task is learned by the resulting network with standard back-propagation. All parameters of the back-propagation and the evolutionary algorithm are summarised in Table 1. A deterministic (μ, λ) selection scheme with one elitist was used together with speci£c mutation and crossover operators, which are described in detail in [16]. As mentioned in the previous section, the number of evaluations of networks was £xed to £ve and encoding schemes with a shorter developmental phase were removed from the population.

The most important measure with respect to performance FR is shown in Figure 5. It is noted that the £nal error relation between neural networks, where information from previously learned problems with lower complexity is transfered to the next developmental step, and neural networks, where the initial weights are solely determined by the evolutionary algorithm, is larger than 1.0 both on average and for the best network. For the best network a factor of two is reached during evolution whereas on average only a factor of just above 1.2 is realized with a slight upward tendency during evolution. This indicates, that the pre-learned weights for the k iteration problem indeed carry information about the $(k+1)$ iteration problem and that this knowledge can be exploited even though the network structure changes considerably. This interpretation is supported by the observation of the values for HS both for the population average, Figure 6, and for the best network, Figure 7.

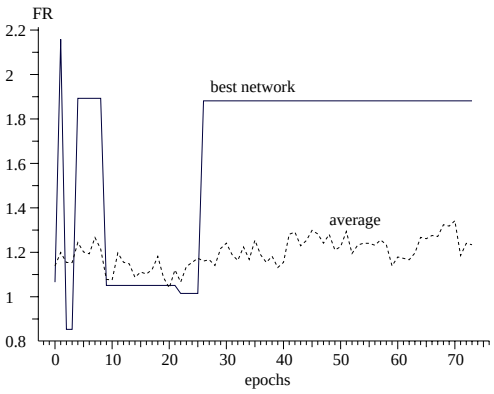

Figure 5: The £nal error relation FR for the best network and for the population average.

The transfer of the pre-learned weights from the k iteration task reduces the initial error for the $(k+1)$ iteration task by a factor of four for the best network and by a factor of three on average. Furthermore, selection pressure seems to be towards a network structure and initial weight setting which support

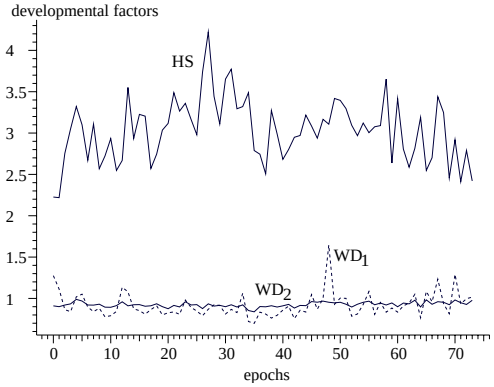

Figure 6: The population average of the measures HS and WD_1, WD_2 during evolution.

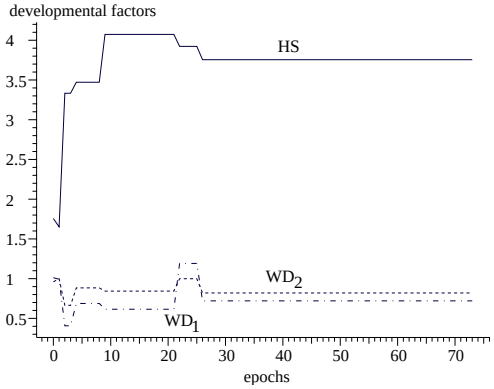

Figure 7: The measures HS and WD_1, WD_2 for the best network during evolution.

the integration of information from pre-learned weights into the larger network structure. For the best network the values for HS develop from just over 1.5 to a maximum of four, which is then again slightly reduced. The values of the weight difference relations WD_1, WD_2 for the best network show that this initial bene£t before the new training of the network also results in a larger change of the genetically determined weight values during the learning process compared to the weights from the previously trained network. However, it should be noted that this only applies to the best network, for the population average the weight changes during re-learning are roughly the same, see Figure 6. Another interesting observation is that the best network is not strictly the one which also exhibits the best performance for the lower order iteration task, as shown in Figure 8. During the £rst epochs the average error decreases, whereas in later stages of the optimisation it can also increase.

All of the aforementioned results indicate that an incremental problem complexity during development can indeed be exploited by the neural network to achieve better initial and overall performance compared to networks where no weight transfer occurs. Of course this comparison is a little misleading, because in either case we forced the network struc-

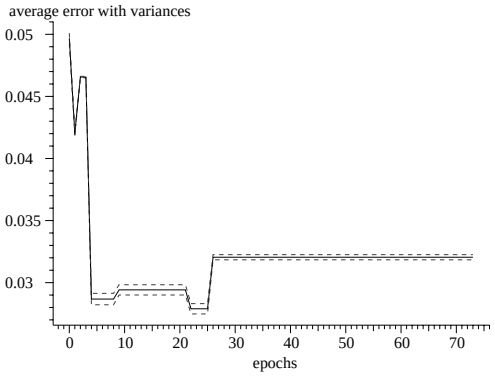

Figure 8: The average error for all networks in the developmental process with their variance for the £ve step iteration problem.

ture to grow for $(m = 5)$ developmental steps. Therefore, the results should also be compared to the optimisation of neural networks for the £ve step iteration problem without any constraints on the developmental process. These experiments have been carried out in [16]. If we compare the £nal performance of the respective networks, we observe that the best networks in this experiment exhibited lower performance than the best networks in [16] for the £ve step iteration task. There are several possible explanations for this observation. Firstly, the network size of the best network of the £ve step iteration problem tends to be considerably smaller than that of the one step prediction problem [16]. Thus, although the incremental approach might be sensible, it does not necessarily correspond to an incremental solution with regard to larger networks required for more dif£cult problems. This is supported by the observation that the effective growth in one developmental phase is kept very small by the evolutionary optimised encoding scheme for this task. In the best network, there are developmental steps where only one neuron is effectively added. This is possible due to the deletion process.

Therefore, with respect to an overall performance increase the iteration of the time series problem in the form used here, might not be the best example. It would be more appropriate to choose a task where more complex problems clearly correspond to larger networks. In such a case it would even be possible not to £x the number of network evaluations and to simply add a regularisation term in the £tness evaluation to reward the network for solving more dif£cult problems. Of course, this is closely connected to the question of achieving the most suitable problem decomposition for such an incremental learning approach. As we argued in Section 2, the £nal goal must be a self-organised approach.

5 Summary and Discussion

In this paper we suggested to combine the evolutionary optimisation of ontogenetic neural networks with learning increasingly complex problems during the different developmental steps. We proposed a model in order to quantitatively analyse the in¤uence of pre-learned weights for problems of lower complexity on the initialisation of larger neural networks, which were in turn trained for more complex problems. There are two main results. Firstly, transferring weights in between networks which have different structures and which have to solve different (although hierarchically organised) problems, can be bene£cial for both the overall and the initial network performance. Furthermore, the evolutionary optimisation aims at organising the developmental process in such a way that the exploitation of pre-learned weights is maximised. On the other hand the performance of the best network from the experiments presented in this paper is lower than the one optimised without any constraints, which was described in [16]. The reason is that one assumption inherent in the incremental approach does not seem to be ful£lled, namely that a problem with higher complexity can be learned better by larger neural networks. This does not seem to be the case for the problem decomposition proposed for the time series modelling task. Indeed the resulting network sizes in [16] are smaller than the one which are forced in this experiments due to £xing the number of developmental steps which are kept variable in [16].

It is interesting to note that the process of modelling a system using an incremental approach for both the model and the dif£culty of the task has also been used successfully for the non-parametric estimation of densities in statistics. In the "method of sieves" (see the work by Geman [23] and references therein) the optimisation of the density estimation is carried out for a subset of the parameter space. Iteratively, the size of the subset increases with the sample size. Thus, the number of free parameters of the model is increased together with the number of available data. In the method of iterative training and further development of the neural network structure, which has been proposed in this work, the exploitation of the available data also increases with the number of free parameters determined by the network size. In this approach, the way in which this interaction is organised is determined by the evolutionary optimisation.

The direction of future research is governed by the question of how a self-organised problem decomposition during the developmental phase can be achieved and whether the strict growth of the network should be relaxed in order to better cope with problems where a decrease of the network size can be sensible even at later developmental steps.

Acknowledgements

The author would like to thank M. Kreutz together with whom the initial model was developed. Furthermore, W. von Seelen, C. von der Malsburg and E. Körner are acknowledged for discussions on the subject and for their support.

References

[1] Geoffrey F. Miller, Peter M. Todd, and Shailesh U. Hedge. Designing neural networks using genetic algorithms. In D.J Schaffer, editor, *Genetic Algorithms: Proceedings of the 3rd International Conferences (ICGA)*, pages 379–384. Morgan Kaufmann, 1989.

[2] W. Schiffmann, M. Joost, and R. Werner. Application of genetic algorithms to the construction of topologies for multilayer perceptrons. In R.F. Albrecht, C.R. Reeves, and N.C. Steele, editors, *Arti£cial Neural Nets and Genetic Algorithms*, pages 675–682. Springer Verlag, 1993.

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

[4] F. Gruau. Genetic synthesis of modular neural networks. In S. Forrest, editor, *Genetic Algorithms: Proceedings of the 5th International Conferences (ICGA)*, pages 318–325, San Mateo, CA, 1993. Morgan Kaufmann.

[5] A. Gierer. Spatial organization and genetic information in brain development. *Biological Cybernetics*, 59:13–21, 1988.

[6] A. Gierer and C.M. Müller. Development of layers, maps and modules. *Current Opinion in Neurobiology*, 5:91–97, 1995.

[7] R.O.L. Wong, M. Meister, and C. Shatz. Transient period of correlated bursting activity during development of the mammalian retina. *Neuron*, 11:923–938, 1993.

[8] Richard K. Belew. Interposing an ontogenetic model between genetic algorithms and neural networks. In J. Cowan, editor, *Advances in Neural Information Processing Systems*, volume 5, pages 99–106. Morgan Kaufmann, 1993.

[9] F. Gruau and D. Whitley. Adding learning to the cellular development of neural networks: Evolution and the Baldwin effect. *Evolutionary Computation*, 1(3):213–233, 1993.

[10] S. Nol£ and D. Parisi. Learning to adapt to changing environments in evolving neural networks. *Adaptive Behavior*, 5(1):75–98, 1997.

[11] Takahiro Sasaki and Mario Tokoro. Adaptation toward changing environments: Why darwinism in nature? In Phil Husbands and Harvey Inman, editors, *Fourth European Conference on Arti£cial Life*, pages 145–153. MIT Press, 1997.

[12] M. Hüsken, J. Gayko, and B. Sendhoff. Optimization for problem classes – neural networks that learn to learn. In *The First IEEE Symposium on Combinations of Evolutionary Computation and Neural Networks (accepted)*, 2000.

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

[14] C. Weber, H. Ritter, J. Cowan, and K. Obermayer. Development and regeneration of the retinotectal map in gold£sh: A computational study. *Phil. Trans. Roy. Soc. Lond. B*, 352:1603–1623, 1997.

[15] Byoung-Tak Zhang. Accelerated learning by active example selection. *International Journal of Neural Systems*, 5(1):67–75, 1994.

[16] Bernhard Sendhoff and Martin Kreutz. Variable encoding of modular neural networks for time series prediction. In V.W. Porto, editor, *Congress on Evolutionary Computation CEC*, pages 259–266. IEEE Press, 1999.

[17] Bernhard Sendhoff and Martin Kreutz. Evolutionary optimization of the structure of neural networks using a recursive mapping as encoding. In G.D. Smith, N.C. Steele, and R.F. Albrecht, editors, *Arti£cial Neural Nets and Genetic Algorithms – Proceedings of the 1997 International Conference*, pages 370–374. Springer Verlag, 1998.

[18] J. Principe, A. Rathie, and J.-M. Kuo. Prediction of chaotic time series with neural networks and the issue of dynamic modeling. *International Journal of Bifurcation and Chaos*, 2(4):989, 1992.

[19] J.C. Principe and J.-M. Kuo. Dynamic modelling of chaotic time series with neural networks. In R.P. Lippmann, J.E. Moody, and D.S. Touretzky, editors, *Advances in Neural Information Processing Systems*, volume 7. Morgan Kaufmann, 1995.

[20] G. Deco and B. Schürmann. Neural learning of chaotic dynamics. *Neural Processing Letters*, 2(2):23–26, 1995.

[21] P. Stagge and B. Sendhoff. An extended elman net for modeling time series. In W. Gerstner, A. Germond, M. Hasler, and J.- D. Nicoud, editors, *Arti£cial Neural Networks (ICANN'97)*, volume 1327 of *Lecture Notes in Computer Science*, pages 427–432. Springer Verlag, 1997.

[22] E.N. Lorenz. Irregularity: A fundamental property of the atmosphere. *Tellus*, A(36):98–110, 1984.

[23] Stuart Geman and Chii-Ruey Hwang. Nonparametric maximum likelihood estimation by the method of sieves. *The Annals of Statistics*, 10(2):401–414, 1982.