Open Access

Complexity Constraints and Error Tolerance in Learning Processes on Small Graphs

H. Atmanspacher[*,1,2], T. Filk[1,2,3], R. Finke[1] and G. Gruber[1]

[1]*Institute for Frontier Areas of Psychology and Mental Health, Wilhelmstr. 3a, D-79098 Freiburg, Germany*

[2]*Parmenides Center for the Study of Thinking, Kardinal-Faulhaber-Str. 14a, Munich, Germany*

[3]*Institut für Physik, Universität Freiburg, Hermann-Herder-Str. 3, D-79104 Freiburg, Germany*

Abstract: Continuing previous studies, we present further results about the behavior of small abstract networks during supervised learning. In particular, we show that constraints on the complexity that a network is permitted to assume during learning reduces its learning success in ways that depend on the nature of the applied limitation. Moreover, we show that relaxing the criterion due to which changes of the network structure are accepted during learning leads to a dramatic improvement of the learning performance. The non-monotonicity of network complexity during learning, which remains unchanged in both scenarios, is related to a similar feature in ε-machine complexity.

1. INTRODUCTION

The study of complex networks, with applications in numerous fields of research, has attracted enormous amounts of interest in recent years. In addition to purely structural investigations of such networks, it is often important to include their dynamics to get a more comprehensive picture. This is particularly significant in cases where the dynamics is explicitly assumed to change the structure, and *vice versa*. A pertinent example for such a situation is the dynamics of learning in networks.

Following up on previous work [1, 2], we are interested in an understanding of general mechanisms and features of the dynamics of learning. Therefore we do not specialize in certain areas of application (such as neuronal networks) but consider abstract networks. Moreover, we focus on small networks of (far) less than a hundred nodes under the simple condition of supervised learning. Although the behavior of such networks is already quite sophisticated, their small size facilitates, or so we hope, an analytical understanding of the intricate details of their dynamics.

A surprising result of earlier studies is a non-monotonic evolution of network complexity as learning proceeds [1]. Another key feature that was observed in fairly small networks is punctuated equilibrium [2]. In addition, we found tentative indications for a transition of the network structure from initial random graphs to small-world networks during learning. These and other results will be further investigated in detail in future work.

In this contribution we present results concerning the behavior of the complexity of the network and the impact of different degrees of error tolerance during learning. We start with a brief description of the model, of the simulation procedure, and of the definition of complexity that we are using in Section 2. Subsequently we describe how the learning process is implemented. Section 4 contains the results for learning under constraints on the complexity of the network and the influence of variations of error tolerance. Section 5 summarizes the paper and sketches some perspectives. An Appendix discusses the relationship between our complexity measure and ε-machine complexity due to Crutchfield and colleagues [3, 4].

2. DESCRIPTION OF THE MODEL AND THE SIMULATION PROCEDURE

We first define the mathematical model that is intended to simulate the dynamics of a network. The model belongs to a particular class of neural networks (for an introduction to neural networks see [5]) and consists of graphs together with two types of dynamics:

1. The dynamics of an activity potential defined on the vertices of the graph. This type of dynamics is described in the present section.
2. The dynamics of the learning process according to which the connectivity among the vertices is changed in order to improve the performance of the system. This dynamics will be described in section 3.

2.1. The Graphs

A *simple, directed graph* can be defined as a set V of *vertices* (sometimes also called *points* or *nodes*) together with a relation "$\rightarrow$" between vertices, where $y \rightarrow x$ denotes a *directed line* (sometimes also called *edge*) from vertex y to vertex x (for more details see e.g. [6, 7]). If two points are connected bidirectionally, i.e., both relations $y \rightarrow x$ and $x \rightarrow y$ are present, the line between x and y is called *undirected*. If all lines are undirected, the graph is called undirected.

*Address correspondence to this author at the Theory Division of the Institute for Frontier Areas of Psychology, Wilhelmstr. 3a, D--79098 Freiburg, Germany; E-mail: haa@igpp.de

The restriction to "simple" graphs means that there are no self-loops (no vertex is connected with itself) and no multiple connections for one direction (for each pair (*x*,*y*) there exists at most one line from *x* to *y* and from *y* to *x*). The connections between vertices can be expressed in terms of the adjacency matrix A with indices x and y :

$$A(x,y) = \begin{cases} 1 & \textit{if there is a directed line from y to x} \\ 0 & \textit{else} \end{cases} . \quad (1)$$

Depending on their role in the learning process we distinguish three types of vertices:

1. *Input vertices* serve to provide an input which the system has to process. The activity potential at input vertices is determined by the input pattern and does not take part in the dynamics on the graph. In our model, input vertices are only connected to internal vertices by directed links, i.e., there are no links to an input vertex and there are no direct links from input vertices to output vertices. Due to redundancies in the input pattern it can happen that during or after learning an input vertex has no link at all to other vertices. In our simulations we always use 16 input vertices.
2. *Output vertices* (z_i) are those vertices at which the activity potential represents the output of the system. Output vertices can be connected to internal vertices in both ways, i.e., the value of the activity potential at the output vertices can influence the dynamics of the activity potential at all internal vertices. In our simulations we always define two output vertices. Input vertices and output vertices together are sometimes referred to as *peripheral vertices*.
3. *Internal vertices* are the actual processing units (although the activity potential at output vertices takes part in the dynamics). They can be arbitrarily connected among each other and to the output vertices, and there can be arbitrary directed connections from input vertices to internal vertices. Thus, the studied networks are highly recurrent. For the simulations presented here we use graphs with 6 internal vertices, sometimes also referred to as *hidden vertices*.

2.2. The Dynamics of the Activity Potential

On the set of vertices V we define a field (called *activity potential* or, sometimes, *firing rate*) $u(t,x)$. At each (discrete) time step t, the activity at each vertex x can assume one of several values: $u(t,x) \in \{0,1,2,\ldots,u_{\max}\}$. The restriction to discrete integer values for the activity potential has numerical reasons (program speed). As a consequence of this discreteness, the possible attractors (see Section 2.3) can only be fixed points or limit cycles, and they are reached after a finite number of steps.

The configuration $\{u(t,x)\} \forall x \in V$ will be called the *state* of the system at discrete time t . Its dynamics is defined by the following equation:

$$u(t+1,x) = f\left(\sum_{y \to x} u(t,y)\right) = f\left(\sum_{y} A(x,y) u(t,y)\right), \quad (2)$$

where $\sum_{y \to x}$ indicates a summation over all vertices y for which a (directed) line from *y* to *x* exists. Hence, the activity $u(t+1,x)$ of vertex *x* at time *t*+1 depends only on the sum of the activities of the neighbors *y* at time *t*. For the transfer function *f*(*x*) (see Fig. **1**) we use a "triangular" function defined by:

$$f(x) = \begin{cases} \left\lceil \dfrac{u_{\max}}{u_0} \cdot x \right\rceil & \textit{for } x \le u_0 \\ \left\lceil \dfrac{u_{\max}}{u_1 - u_0} \cdot (u_1 - x) \right\rceil & \textit{for } u_0 < x < u_1 \\ 0 & \textit{for } x \ge u_1 \end{cases} , \quad (3)$$

where $\lceil u \rceil$ denotes the smallest integer greater or equal to *u*.

Simulations with different values for the parameters u_0 and u_1, and even with various other forms of the transfer function, left the results reported in the following qualitatively unchanged. However, the number of optimal learners per 1000 runs depends crucially on the transfer function. We chose the parameters $u_0 = u_{max}$ and $u_1 = 30$ because they yielded a maximal number of optimal learners.

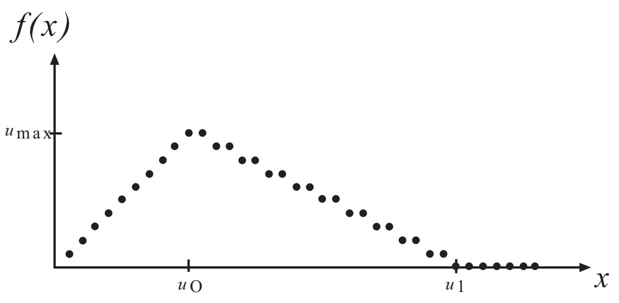

Fig. (1). Illustration of the transfer function *f*(*x*). In the simulations we used the values u_{max}= 10, u_1= 30, and u_0= 10.

For the input vertices the value of the activity is set to 0 or u_{max}= 10 according to a set of "input images" or input patterns B_i which are "presented" to the graph. Fig. (**2**) shows the 11 input patterns used in the simulations. The input patterns are grouped into three different classes. The first class contains only image 1, the second class consists of images 2 to 5, and the other six images 6 to 11 belong to the third class. Note that for the images in group two and three the number of vertices with activity 0 and maximal activity is the same: the system cannot distinguish these patterns by simply counting the number of vertices with maximal activity. Furthermore, it should be obvious that the arrangement of these vertices in form of a 4×4 matrix is of no relevance or meaning to the system.

For each class of input patterns we define an *optimal reaction* $u(B_i;z_i)_{opt}$ of the system coded by the activity of the two output vertices. The optimal reactions for the three different classes are: (0,0) for group 1, (10,0) for group 2, and (0,10) for group 3. An *optimal learner* is a graph for

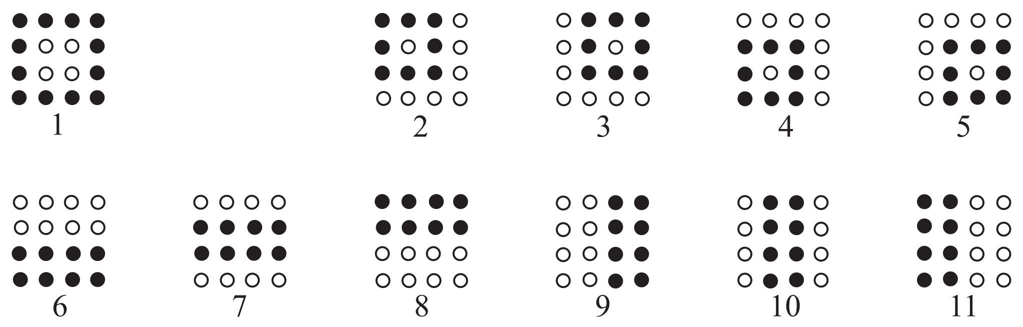

Fig. (2). The 11 input patterns B_i, shown as 4×4 matrices, which were presented by the 16 input vertices. ∘ indicates activity 0, • indicates activity u_{max}. Note that this also represents the corrected sequence of inputs in [1].

which the activities at the output vertices agree, after the learning process, with the predefined optimal reaction for each of the 11 images.

The following expressions define a measure for the performance of a graph. Sometimes they are called *error functions* or *performance functions*, and they may be interpreted as an inverse fitness of the system:

$$\Delta_2 = \sum_{11\,images\,B_i} \; \sum_{20\,time\,steps\,t} \; \sum_{2\,output\,vertices\,z_i} (u(t,z_i) - u(B_i;z_i)_{opt})^2, \quad (4)$$

and:

$$\Delta_1 = \sum_{11\,images\,B_i} \; \sum_{20\,time\,steps\,t} \; \sum_{2\,output\,vertices\,z_i} | u(t,z_i) - u(B_i;z_i)_{opt} |. \quad (5)$$

Although the quantitative results for these two error functions differ, we found no actual difference in the qualitative behavior of the systems.

The sums in Eqs. (4) and (5) extend over the 11 input patterns B_i, 20 time steps t (updates) according to the defined dynamics, and the two output vertices z_1 and z_2. Δ_i vanishes for optimal learners, while for random graphs Δ_2 assumes values between 15000 and 25000, and Δ_1 assumes values between 2000 and 2200. After each change of the input pattern, the system relaxes onto an attractor after some transient time steps (usually 10 to 20). After this relaxation time the output is measured.

The simulations are programmed in C^{++} and the random number generator used is the standard random number generator "rand()" provided by the C^{++} library.

2.3. Attractors and Attractor States

We now define the notion of attractors and attractor states in the context of neural networks.

While the configuration $\{u(t,x)\}_{x\in V}$ characterizes the *state* of the system at time t, $u(t,x)$ is the state of the vertex x at time t. Furthermore, as the states of the input vertices depend on the input pattern "presented" to the graph and do not take part in the dynamics, we typically exclude the input vertices and characterize the state of the system by the collection of states of the internal vertices and the output vertices only.

The restriction of $u(t,x)$ to integer values (and the restriction to networks with a finite number of vertices) implies that there is only a finite number of states. An *attractor* (or *attractor set*) of a dynamical system may be defined as a subset of all possible states which is closed under the dynamics, i.e., once the system assumes one of the states of the attractor set then it will stay within the attractor set (Usually it is assumed that this attractor set is also minimal, i.e., the system approaches each state within this attractor set arbitrarily closely). The *attractor landscape* consists of all attractor sets of a dynamical system.

Starting from an arbitrary initial state, the dynamical system corresponding to the neural network used in our simulations runs into an attractor set after a finite number of time steps. In many cases, this attractor set consists of one element only, i.e. one single state that is asymptotically stable. Such an attractor is called a *fixed point*. Otherwise the attractor is a *limit cycle*, i.e. a periodic succession of *attractor states*. Strange attractors do not occur in our case since the number of states is finite.

In general, the attractor onto which a system relaxes after some time depends only on the initial state at time t=0. In our case, however, there is an additional complication, because the attractor landscape depends on the input pattern "presented" to the network. In the context of dynamical systems, these input patterns can be interpreted as boundary conditions and the attractor landscapes depend on these boundary conditions. Strictly speaking, there are as many attractor landscapes as there are input patterns.

Our aim is to define a minimal set of attractor states which is closed in the following sense. Taking an arbitrary attractor state in this set as an initial condition and any of the eleven input patterns as a boundary condition, the attractor state (or the attractor states) reached by the system after a sufficient amount of time should again be an element (elements) of this set of attractor states. Atmanspacher and Filk [1] defined the number of attractor states in this set as a complexity measure for learning processes (See Appendix for a comparison with ε-machine complexity as introduced by Crutchfield and Young [3]). This complexity measure turns out to be non-monotonic as a function of randomness. It is (in general) small for random graphs as well as for (least random) optimal learners, but it is usually large for intermediate epochs during the learning process.

It should be noted that the notion of randomness here refers to the structure of the graph *in relation to* the learning algorithm: During the learning process the graphs become less random in their performance with respect to the task to be learned. This does not imply that the graphs become "ordered" in the sense that the degree distribution or other common statistical properties change significantly with respect to the corresponding properties for random graphs. However, recent simulations (to be published elsewhere) show that distributions for certain motifs change drastically as compared to random graphs.

In detail, we determine this set of attractor states as follows (see also [1]): We start from a random initial configuration, apply input pattern 1 to the input vertices and let the system run for a sufficiently long time (we chose t=101) until the system enters a fixed point or a cycle. We identify all states of this cycle (only one state in case of a fixed point) and subsequently use these states as initial conditions, applying successively all other input patterns and determining the corresponding attractor states. In a second step, all new attractor states are used as initial conditions and all input patterns are tested to check if new attractor states appear. This procedure stops when no new attractor states occur after all existing attractor states have been used as initial conditions with all input patterns applied.

In a more mathematical terminology, this procedure defines a multiplicative structure of evolution operators which depend on the input patterns. The set of attractor states constitutes a representation space of this structure. Some of its general properties have been analyzed previously [1].

3. THE LEARNING PROCESS

The simulation procedure for obtaining optimal learners mimics an evolutionary scenario: starting from a graph with a random adjacency matrix (an arbitrary "species"), we obtain a sequence of new graphs (new "species") by randomly injecting changes ("mutations") in the connectivities of the graph. We measure both the performance function (inverse "fitness") of each graph and its complexity in terms of the number of attractor states. If the performance function is smaller than for any previous graph, i.e., if the fitness is better, and if the graph satisfies certain constraints imposed on its complexity (which will be described below), then the new graph is accepted and becomes the "surviving species". Otherwise it is rejected and the previous "fitter" graph survives.

In order to distinguish between the single time (update) steps of the iterative dynamics (Eq. (2)) and the iterative sequence of graphs obtained by randomly inserting and deleting lines, we refer to the latter as "generation steps". One generation step comprises 330 or 440 time steps, depending on whether the (transient) relaxation time after each image change is chosen to be 10 or 20 time steps.

In detail, the simulations proceed as follows: For n vertices (input, internal, and output) we start with an $n\times n$ adjacency matrix $A(x,y)$ whose initial entries are all 0. Next we set each of the elements in the upper diagonal part of $A(x,y)$ to 1 with some suitable probability p=0.4 with respect to all *admissible* links (some links are prohibited: no self-loops, no output-output connections). This number turned out to yield a high percentage of optimal learners for graphs with n_{int} = 6 internal vertices, i.e., n=24 vertices in total, in previous simulations without complexity restrictions.

Note that p=0.4 is twice the threshold $p_{th} = 1/(n_{\text{int}} - 1)$ for the emergence of a "giant component" in the statistical sense of Renyi-Erdos network theory [8] for a random graph with n_{int} = 6 internal vertices. As a rule of thumb, it turns out that this factor of about two is a suitable choice for the initial random graph in the simulations even if larger graphs are considered.

The initial $n\times n$ random matrix is symmetrized (i.e., the lower diagonal part is symmetrically completed), so that it becomes the adjacency matrix for an undirected random graph. All entries which correspond to a connection between an input and an ouput vertex are set to 0. All connections from internal vertices to input vertices are also set to 0 (although this does not have any influence on the simulation because the input vertices do not participate in the dynamics).

Each image is presented for a total number of 40 (sometimes 30) time steps, and for the last 20 time steps the difference between the actual activity and the predefined optimal activity (the "error" Δ_i) is measured. The images are always presented in the same sequence (not in randomized order). Then we determine the number of attractor states of the resulting graph.

Next, one of the connections among all admissible links, including those between input and internal vertices, is changed by inserting or deleting a single directed line for two randomly chosen vertices which are not both input and output points. This way, directed graphs become admissible: even though the initial random matrix was symmetrized (undirected lines only), insertion or deletion refers to directed lines. The performance function of the new graph is determined.

In [1], the only criterion for accepting a graph was an improvement of the performance function Δ_2. Now we add a second criterion concerning the complexity of the graph, which is applied in addition whenever the criterion for Δ_i alone is satisfied. For this second criterion we investigate two variants: a fixed complexity limit and an adaptive complexity limit.

1. *Fixed complexity limit*: The number of attractor states must not exceed a certain limit, determined in relation to the number of attractor states N_0 of the initial (random) graph. We define the maximum number of attractor states which a network is allowed to have by $N_{fix} = p_1 N_0$, where p_1 is a parameter between 1 and 2 (p_1 <1 is obviously meaningless, while for p_1 >2 the performance rapidly approaches the unrestricted case). Any new graph is only accepted if *both* the performance is better than for the previously

accepted graph *and* the number of attractor states does not exceed N_{fix}.

2. *Adaptive complexity limit*: The rate of growth of the number of attractor states must not exceed a certain limit. Let N_{prev} be the number of attractor states of any precedingly accepted graph, then the number of attractor states of the subsequent graph must not exceed $N_{adap} = p_2 N_{prev}$. As above p_2 is varied between 1 and 2 (Here, $p_2 < 1$ is possible but leads to a restriction which is almost never satisfied. For $p_2 > 2$ the performance again approaches the unrestricted case).

With both kinds of complexity limits, the learning process stops when a certain number of successive line changes (in our case 1500 generation steps) has been rejected.

4. RESULTS

4.1. Complexity Constraints

For a reasonable characterization of the influence of fixed and adaptive complexity limits on the learning behavior, we compiled 3000 runs for each value of p_1 and p_2. The error in these runs is measured as Δ_1, i.e., as the absolute value (rather than the square) of the difference between actual output and desired output.

The number of terminal graphs is then plotted as a function of the absolute deviation from the desired output at the end of the learning process. Depending on the complexity constraint applied, more or less of these terminal graphs are optimal learners. The mean of the resulting histograms for each value of p_1 and p_2 provides a statistical characterization of the learning success. Without any complexity constraint, the value of $\langle \Delta_1 \rangle$ is about 600.

In Fig. (**3**) we plot these mean absolute deviations $\langle \Delta_1 \rangle$ as a function of the limiting parameters p_1 (for fixed constraints, squares) and p_2 (for adaptive constraints, crosses). It is evident that changes of fixed complexity constraints influence the success of learning only faintly, from $\langle \Delta_1 \rangle \approx 800$ at $p_1 = 1.0$ to $\langle \Delta_1 \rangle \approx 750$ at $p_1 = 2.0$. This is identical with the value of $\langle \Delta_1 \rangle$ without limiting parameters. Not only the means but also the form of the histograms remains almost invariant over the considered range of p_1.

For adaptive complexity constraints, where the limiting parameter p_2 refers to the preceding (rather than the first) learning step, the situation turns out to be completely different. The mean absolute deviations $\langle \Delta_1 \rangle$ are higher than 1600 if only a small increase in complexity is permitted (p_2 = 1.05) and they decrease exponentially with increasing p_2, until they reach (asymptotically) a value close to that for fixed constraints, $\langle \Delta_1 \rangle \approx 750$, for p_2 = 2.0.

There are presumably several reasons for the different learning behavior with fixed versus adaptive complexity constraints. One of them is that fixed constraints allow arbitrary fluctuations of the complexity of the network as long as they stay below the fixed limit, determined by the initial complexity. This gives the network some flexibility to explore different possible trajectories. By constrast, an adaptive constraint always relates to the complexity for the preceding learning step, so that a temporary decrease in complexity may enforce a limit that is smaller than under fixed constraints. This explains, at least in part, why the learning success for small p_2 is so much worse than under fixed constraints.

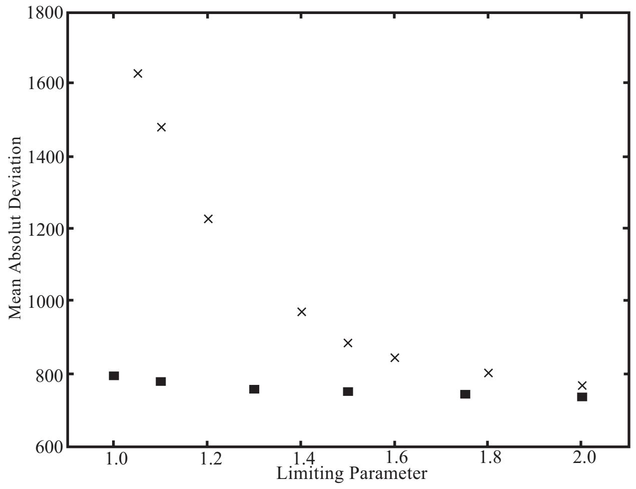

Fig. (3). Mean absolute deviation $\langle \Delta_1 \rangle$ as a function of limiting parameter p_1 (■ for fixed constraints) and p_2 (× for adaptive constraints). High values indicate low average learning success and vice versa. Errors are of the size of the plotted symbols. For further discussion see text.

An unexpected feature of learning on small graphs observed earlier [1] is the non-monotonic behavior of complexity during learning for both optimal and non-optimal learners: The complexity at the beginning and at the end of learning is low compared to intermediate stages (Note that others, such as Crutchfield [9], indicated similar results). This behavior is also found with complexity constraints, but we still have no compelling explanation for it.

4.2. Error Tolerance

So far [1] we studied learning processes in which changes of the structure of the graph were only accepted if the error Δ strictly decreased. Since networks often operate under conditions in which they tolerate increasing error temporarily, it is reasonable to relax this criterion. A first, important step toward such a relaxation is to also accept changes leading to the same error, achieving neither improvement nor its opposite. In the theory of molecular evolution, such a criterion was proposed in 1968 as "neutral evolution" by Kimura [10] and has been studied intensely since then [11]. It is now well-known that the difference between strict improvement and neutrality can dramatically change the evolutionary capabilities of species.

As Fig. (**4**) shows, the same is the case for learning processes on small graphs studied here. The plot shows the percentage of optimal learners among 1000 runs as a function of error tolerance. The acceptance criterion for a change of the graph structure is that Δ_1 at any given generation is strictly smaller than the value of Δ_1 for the generation surviving so far plus an error tolerance δ. The value δ=0 corresponds to strict improvement (as in [1]) and δ=1 corresponds to neutrality. Higher values of δ reflect actual "error tolerance", i.e. it is permissible that Δ_1 increases locally during evolution or learning, respectively.

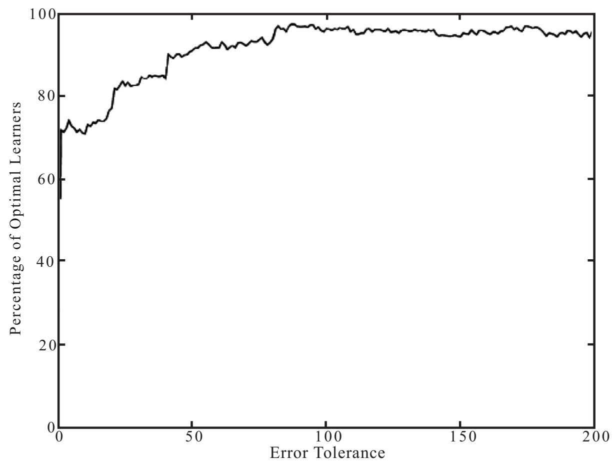

Fig. (4). Percentage of optimal learners as a function of error tolerance. Note the dramatic increase of optimal learners if the acceptance condition is relaxed from strict improvement (error tolerance δ = 0) to neutrality (error tolerance δ = 1). Further relaxation of acceptance conditions leads only gradually to more optimal learners, before the curve saturates.

The plotted curve yields a dramatic increase of optimal learners, i.e. of learning success, from 2% in the condition of strict improvement ($\delta = 0$) to more than 70% in the neutrality condition ($\delta = 1$). Further increases in error tolerance yet lead to more optimal learners, although the curve becomes increasingly flatter and saturates at $\Delta \approx 85$ with more than 95% optimal learners. For larger graphs, the change from $\delta = 0$ to $\delta = 1$ is less dramatic but still considerable.

In view of future work, in which we plan to analyze the possible emergence of small-world or scale-free properties [12] from initially random graphs during learning, it will be conducive to work at least under neutrality conditions. In this case, the achievable statistics might be good enough to find evidence for or against such properties even though the graphs themselves are small. Moreover, investigations of the nature of punctuated equilibrium in small graphs [2], which will be carried out as well, will profit from better statistics.

5. SUMMARY AND PERSPECTIVES

Continuing previous studies of supervised learning in small graphs [1, 2], this paper presents results on essentially two issues. First, the influence of constraints on the complexity, which the network is allowed to assume during learning, on the learning performance is investigated. Second, the learning performance is analyzed under different criteria for the acceptance of a change of the network structure.

With respect to complexity constraints, we found that the learning performance, measured by the terminal distance Δ of the actual output from the desired output, is generally reduced if the complexity of the network is not permitted to exceed particular limitations. This reduction is moderate in case of a fixed criterion, given as a multiple p_1 of the initial value of Δ, and it does hardly depend on p_1. If such a rigid complexity limitation is replaced by an adaptive criterion, given as a multiple p_2 of the preceding value of Δ, respectively, then the learning performance is greatly obstructed if p_2 is not much greater than 1. Increasing p_2 leads to an improved performance, approaching the behavior for fixed constraints at about $p_2 = 2$.

With respect to an increased error tolerance for the acceptance of changes of the graph during learning, we found a dramatic increase of the number of optimal learners. if the acceptance criterion is relaxed from strict improvement to neutrality. A further increase of tolerated error, i.e. temporary deterioration of performance, leads to further, but less drastic increases in the number of optimal learners up to about 96% of all runs.

The non-monotonic behavior of the complexity of the graph observed earlier [1] resembles speculations by Crutchfield [9] based on ε-machine complexity (outlined in an Appendix to this paper). It remains unchanged under both complexity constraints and increased error tolerance. Our previous hypothesis that this feature might teach us something about semantic and pragmatic information processed in the network is still inconclusive. In particular, we are still unable to find a significant correlation of the intermediate maximum of complexity during learning with other network observables. Further work along these lines might profit from comparison with recent results [13] about the emergence of small-world networks during the solution of cognitive insight problems.

APPENDIX: ε -MACHINE COMPLEXITY

The idea to use algorithms or automata for a definition of complexity goes back to Solomonoff [14], Kolmogorov [15] and Chaitin [16]. It is based on deterministic automata and represents, loosely speaking, a measure of randomness. Crutchfield and Young [3] suggested to use stochastic automata as well, which they called ε-machines. For a survey of the theory of computational complexity, with particular emphasis on ε-machines, see [4]. The determination of the complexity C_ε of an ε-machine, characterizing a given symbol sequence, can be divided into four main steps.

1. Construction of a tree: A binary tree $T = (V,E,a)$ of length l_1, with a finite set V of vertices, a set $E \subset V \times V$ edges, and an origin $a \in V$ of T, is assigned to a given symbol sequence S. A conditional probability $p_e = p_{v_1 \to v_2}$ is assigned to each edge $e = (v_1, v_2)$.
2. Search for equivalent subtrees: Consider all subtrees of T that have length l_2 ($l_2 < l_1$). Within the set of subtrees an equivalence relation (ε-similarity) is

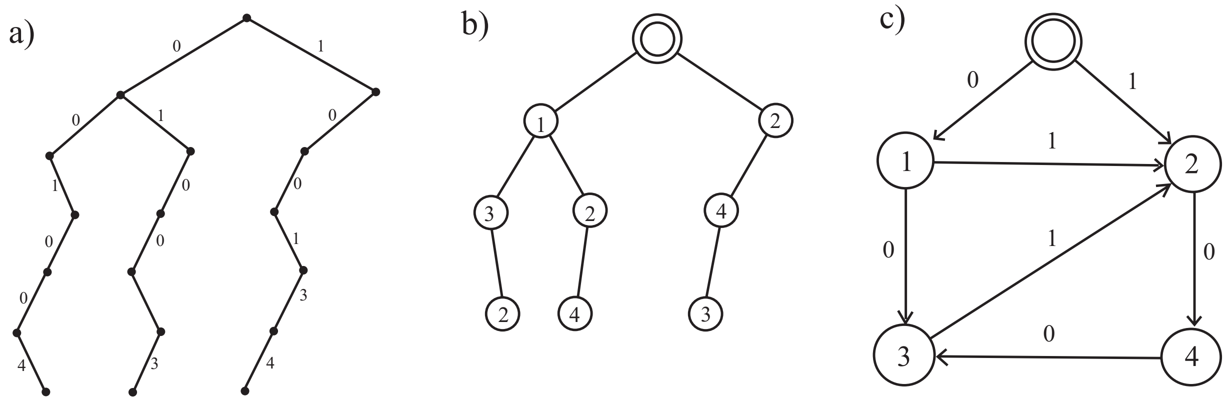

Fig. (5). Illustration of how ε-complexity is calculated for a (deterministic) period-3 process. For a binary tree of length l_1=6 (**a**) and subtrees of length l_2=3 as equivalence classes (**b**), the automaton (**c**) is derived.

defined such that any two subtrees are equivalent iff the difference of the probabilities assigned to their edges is smaller than ε, $|\, p_{e_1} - p_{e_2} \,| < \varepsilon$.

3. Construction of stochastic automaton: This similarity condition generates a classification in the set of subtrees. Each equivalence class is regarded as a state of the automaton, a so-called *causal state*, and each edge between the origins of two subtrees represents an edge between two states of the automaton. The transition probabilities between different states of the automaton are determined such that the original sequence is reconstructed in an ε-similar manner. Increasing l_1 and l_2 results in an increasing resolution of the dynamics.

4. Determination of complexity: The ε-complexity C_ε is defined as the Shannon information of the state probabilities of the automaton.

For more details concerning an optimal choice of the parameters l_1, l_2, and ε, we refer the interested reader to the quoted literature, see also [17] for an overview.

An intelligible example illustrating how the calculation of C_ε works can be given for a deterministic period-three process, where $S = 001001001001...$. This sequence can be assigned to a binary tree as in Fig. (**5a**) Defining equivalence classes 0, 1, 2, 3, 4 as in Fig. (**5b**) leads to an automaton shown in Fig. (**5c**). It consists of a closed loop composed of three automaton states, each of state probability 1, thus providing $C_\varepsilon = \log 3$. In case of doubly stochastic behavior ($p_{v_1 \to v_2} = 0.5$, binary tree), the automaton has only one state (Fig. **6**). The resulting ε-similar sequence of automaton states has period 1, and $C_\varepsilon = 0$.

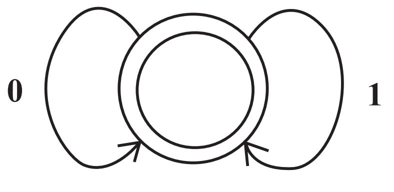

Fig. (6). Stochastic automaton for doubly stochastic behavior.

These simple examples show that ε-machine complexity is non-monotonic as a function of randomness: It is low for both purely deterministic and purely random (stochastic) symbol sequences. An overview of several other non-monotonic measures of complexity as compared to monotonic measures can be found in [17]. A direct application of a non-monotonic measure (based on mutual information) to charaterize the functional connectivity of neural systems is due to Tononi *et al.* [18].

The complexity measure proposed in [1] is closely related to C_ε insofar as the number of attractor states of the network dynamics during learning can be identified as the number of automaton states. In the approach presented here, we do already enter at the level of the automaton (the graph) as it results from the network dynamics over all inputs. Since the attractors in the scenario introduced in Sec. 2 are either fixed points or limit cycles, hence deterministic, the logarithm of their total number does precisely yield C_ε for each step of the considered learning process. It should be added that in our case this does not only characterize the actual realization of one learning process, but the complete set of all possible "trajectories".

We have speculated [1] that the stage of the learning process exhibiting maximum complexity could be related to a release of semantic or pragmatic information, signifying something like an "aha"-experience. This speculation is reinforced by corresponding remarks by Crutchfield [9] in the context of ε-machine complexity.

ACKNOWLEDGMENTS

Robert Finke and Günter Gruber are grateful for a research grant during their student internship in the theory department at IGPP, from which this work emerged. Comments and suggestions by three anonymous referees are greatly appreciated.

REFERENCES

[1] H. Atmanspacher and T. Filk, "Complexity and non-commutativity of learning operations on graphs", *Biosystems*, vol. 85, pp. 84-93, 2006.
[2] T. Filk and A. von Müller, "Evolutionary learning of small networks", *Complexity*, vol. 13(3), pp. 43-54, 2008.

[3] J.P. Crutchfield and K. Young, "Inferring statistical complexity", *Phys. Rev. Lett.*, vol. 63, pp. 105-108, 1989.
[4] C.R. Shalizi and J.P. Crutchfield, "Computational mechanics: pattern and prediction, structure and simplicity", *J. Stat. Phys.*, vol. 104, pp. 816-879, 2001.
[5] S. Haykin, *Neural Networks: A Comprehensive Foundation*. Saddle River NJ: Prentice Hall, 1999.
[6] R.J. Wilson, *Introduction to Graph Theory*. 3rd ed. Harlow Essex: Longman Scientific & Technical, 1985.
[7] S. Bornholdt and H.G. Schuster, Eds., *Handbook of Graphs and Networks*. Weinheim, Wiley-VCH, 2003.
[8] P. Erdos and A. Renyi, "On the evolution of random graphs", *Publ. Math. Inst. Hungarian Acad. Sci.*, vol. 5, pp. 17-61, 1960.
[9] J.P. Crutchfield. Private communication 2006; compare also his presentations of 2003 on his homepage.
[10] M. Kimura, "Evolutionary rate at the molecular level", *Nature*, vol. 217, pp. 624-626, 1968.
[11] E.G. Leigh, Jr., "Neutral theory: a historical perspective", *Evol. Biol.*, vol. 20, pp. 2075-2091, 2007.
[12] A. Albert and A.-L. Barabasi, "Statistical mechanics of complex networks", *Rev. Mod. Phys.*, vol. 74, pp. 47-97, 2002.
[13] M.A. Schilling, "A small-world network model of cognitive insight", *Creativ. Res. J.*, vol. 17, pp. 131-154, 2005.
[14] R. Solomonoff, "A formal theory of inductive inference", *Inf. Control*, vol. 7, pp. 1-22, 224-254, 1964.
[15] A.N. Kolmogorov, "Three approaches to the quantitative definition of information", *Probl. Inf. Transm.*, vol. 1, pp. 1-7, 1965.
[16] G.J. Chaitin, "On the length of programs for computing binary sequences", *J. ACM*, vol. 13, pp. 547-569, 1966.
[17] R. Wackerbauer, A. Witt, H. Atmanspacher, J. Kurths, and H. Scheingraber, "A comparative classification of complexity measures", *Chaos Solitons Fractals*, vol. 4, pp. 133-173, 1994.
[18] G. Tononi, O. Sporns, and G. Edelman, "A complexity measure for selective matching of signals by the brain", *Proc. Natl. Acad. Sci. USA*, vol. 93, pp. 3422-3427, 1996.

Received: May 30, 2009 Revised: September 1, 2009 Accepted: September 23, 2009