(1) **Paper no. NNK03085SPI, Recursive self-organizing network models**

(2) **Invited article**

(3) **Final version** of the article

(4) **Authors:**

(a) Barbara Hammer, Research group LNM, Department of Mathematics/ Computer Science, University of Osnabrück, Germany
hammer@informatik.uni-osnabrueck.de

(b) Alessio Micheli, Dipartimento di Informatica, Università di Pisa, Pisa, Italia
micheli@di.unipi.it

(c) Alessandro Sperduti, Dipartimento di Matematica Pura ed Applicata, Università degli Studi di Padova, Padova, Italia
sperduti@math.unipd.it

(d) Marc Strickert, Research group LNM, Department of Mathematics/ Computer Science, University of Osnabrück, Germany
marc@informatik.uni-osnabrueck.de

(5) **Corresponding author for proofs and reprints:**

(a) Barbara Hammer, Department of Mathematics/Computer Science, University of Osnabrück, Albrechtstr. 28, D-49069 Osnabrück, Germany,
e-mail: hammer@informatik.uni-osnabrueck.de
phone: +49-541-969-2488
fax: +49-541-969-2770

Invited article

Recursive self-organizing network models

Barbara Hammer

Research group LNM, Department of Mathematics/Computer Science,

University of Osnabrück, Germany

Alessio Micheli

Dipartimento di Informatica, Università di Pisa, Pisa, Italia

Alessandro Sperduti

Dipartimento di Matematica Pura ed Applicata,

Università degli Studi di Padova, Padova, Italia

Marc Strickert

Research group LNM, Department of Mathematics/Computer Science,

University of Osnabrück, Germany

Abstract

Self-organizing models constitute valuable tools for data visualization, clustering, and data mining. Here, we focus on extensions of basic vector-based models by recursive computation in such a way that sequential and tree-structured data can be processed directly. The aim of this article is to give a unified review of important models recently proposed in literature, to investigate fundamental mathematical properties of these models, and to compare the approaches by experiments. We first review several models proposed in literature from a unifying perspective, thereby making use of an underlying general framework which also includes supervised recurrent and recursive models as special cases. We shortly discuss how the models can be related to different neuron lattices. Then, we investigate theoretical properties of the models in detail: we explicitly formalize how structures are internally stored in different context models and which similarity measures are induced by the recursive mapping onto the structures. We assess the representational capabilities of the models, and we shortly discuss the issues of topology preservation and noise tolerance. The models are compared in an experiment with time series data. Finally, we add an experiment for one context model for tree-structured data to demonstrate the capability to process complex structures.

Key words: self-organizing map, Kohonen map, recursive models, structured data, sequence processing

1 Introduction

The self-organizing map introduced by Kohonen constitutes a standard tool for data mining and data visualization with many applications ranging from web-mining to robotics (Kaski, Kangas, and Kohonen, 1998; Kohonen, 1997; Kohonen et al., 1996; Ritter, Martinetz, and Schulten, 1992). Combinations of the basic method with supervised learning allow to extend the scope of applications also to labeled

Email addresses: `hammer@informatik.uni-osnabrueck.de` (Barbara Hammer), `micheli@di.unipi.it` (Alessio Micheli), `sperduti@math.unipd.it` (Alessandro Sperduti), `marc@informatik.uni-osnabrueck.de` (Marc Strickert).

data (Kohonen, 1995; Ritter, 1993). In addition, a variety of extensions of SOM and alternative unsupervised learning models exist which differ from the standard SOM, e.g. with respect to the chosen neural topology or with respect to the underlying dynamic equations; see (Bauer and Villmann, 1997; Heskes, 2001; Kohonen, 1997; Martinetz, Berkovich, and Schulten, 1993; Martinetz and Schulten, 1993), for example.

The standard learning models have been formulated for vectorial data. Thus, the training inputs are elements of a real-vector space of a finite and fixed dimension. As an example, pixels of satellite images from a LANDSAT sensor constitute 6-dimensional vectors with components corresponding to continuous values of spectral bands. In this case, evaluation with the SOM or with alternative methods is directly possible, as demonstrated e.g. in (Villmann, Merenyi, and Hammer, 2003). However, in many applications data are not given in vector form: sequential data with variable or possibly unlimited lengths belong to alternative domains, such as time series, words, or spatial data like DNA sequences or amino acid chains of proteins. More complex objects occur in symbolic fields for logical formulas and terms, or in graphical domains, where arbitrary graph structures are dealt with. Trees and graph structures also arise from natural language parsing and from chemistry. If unsupervised models shall be used as data mining tools in these domains, appropriate data preprocessing is usually necessary. Very elegant text preprocessing is offered for the semantic map (Kohonen, 1997). In general, however, appropriate preprocessing is task-dependent, time consuming, and often accompanied by a loss of information. Since the resulting vectors might be high-dimensional for complex data, the curse of dimensionality applies. Thus, standard SOM might fail unless the training is extended by further mechanisms such as the metric adaptation to given data, as proposed in (Hammer and Villmann, 2001; Kaski, 2001; Sinkkonen and

Kaski, 2002).

It is worthwhile to investigate extensions of SOM methods in order to deal directly with non-vectorial complex data structures. Two fundamentally different ways can be found in literature, as discussed in the tutorial (Hammer and Jain, 2004). On one hand, the basic operations of single neurons can be extended to directly allow complex data structures as inputs; kernel matrices for structures like strings, trees, or graphs constitute a popular approach within this line (Gärtner, 2003). On the other hand, one can decompose complex structures into their basic constituents and process each constituent separately within a neural network, thereby utilizing the context imposed by the structure. This method is particularly convenient, if the considered data structures, such as sequences or trees, possess recursive nature. In this case, a natural order in which the single constituents should be visited is given by the natural order within the structure: sequence entries can be processed sequentially within the context defined by the previous part of the sequence; tree structures can be processed by traversing from the leaves to the root, whereby the children of a vertex in the tree define the context of this vertex.

For supervised learning scenarios the paradigm of recursive processing of structured data is well established: time series, tree structures, or directed acyclic graphs have been tackled very successfully in recent years with so-called recurrent and recursive neural networks (Frasconi, Gori, and Sperduti, 1998; Hammer, 2002; Hammer and Steil, 2002; Kremer, 2001; Sperduti, 2001; Sperduti and Starita, 1997). Applications can be found in the domain of logic, bio-informatics, chemistry, natural language parsing, logo and image recognition, or document processing (Baldi et al., 1999; Bianucci et al., 2000; Costa et al., 2003; Diligenti, Frasconi, amd Gori, 2003; De Mauro et al., 2003; Pollastri et al., 2002; Sturt et al., 2003; Vullo and

Frasconi, 2003). Extensive data preprocessing is not necessary in these applications, because the models directly take complex structures as inputs. These data are recursively processed by the network models: sequences, trees, and graph structures of possibly arbitrary size with real-valued labels attached to the nodes are handled step by step. The output computed for one time step depends on the current constituent of the structure and the internal model state obtained by previous calculations, i.e. the output of the neurons computed in recursively addressed previous steps. The models can be formulated in a uniform way and their theoretical properties such as representational issues and learnability have been investigated (Frasconi, Gori, and Sperduti, 1998; Hammer, 2000).

In this article, we investigate the recursive approach to unsupervised neural processing of data structures. Various unsupervised models for non-vectorial data are available in literature. The approaches presented in (Günter and Buhnke, 2001; Kohonen and Sommervuo, 2002) use a metric for SOM that directly works on structures. Structures are processed as a whole by extending the basic distance computation to complex distance measures for sequences, tree, or graphs. The edit distance, for example, can be used to compare words of arbitrary length. Such a technique extends the basic distance computation for the neurons to a more expressive comparison which tackles the given input structure as a whole. The corresponding proposals fall thus into the first class of neural methods for non-vectorial data introduced above. In this article, we are interested in the alternative possibility to equip unsupervised models with additional recurrent connections and to recursively process non-vectorial data by a decomposition of the structures into their basic constituents. Early unsupervised recursive models, such as the temporal Kohonen map or the recurrent SOM, include the biologically plausible dynamics of leaky integrators (Chappell and Taylor, 1993; Koskela et al., 1998a; Koskela et al., 1998b).

This idea has been used to model direction selectivity in models of the visual cortex and for time series representation (Farkas and Miikkulainen, 1999; Koskela et al., 1998a; Koskela et al., 1998b). Combinations of leaky integrators with additional features can increase the capacity of the models as demonstrated in further proposals (Euliano and Principe, 1999; Hoekstra and Drossaers, 1993; James and Miikkulainen, 1995; Kangas, 1990; Vesanto, 1997). Recently, also more general recurrences with richer dynamics have been proposed (Hagenbuchner, Tsoi, and Sperduti, 2001; Hagenbuchner, Sperduti, and Tsoi, 2003; Strickert and Hammer, 2003a; Strickert and Hammer, 2003b; Voegtlin, 2000; Voegtlin, 2002; Voegtlin and Dominey, 2001). These models transcend the simple local recurrence of leaky integrators and they can represent much richer dynamical behavior, which has been demonstrated in many experiments. While the processing of tree-structured data is discussed in (Hagenbuchner, Tsoi, and Sperduti, 2001; Hagenbuchner, Sperduti, and Tsoi, 2003), all the remaining approaches have been applied to time series.

Unlike their supervised counterparts, the proposed unsupervised recursive models differ fundamentally from each other with respect to the basic definitions and the capacity. For supervised recurrent and recursive models, basically one formulation has been established, and concrete models just differ with respect to the connectivity structure, i.e. the concrete functional dependencies realized in the approaches (Hammer, 2000; Hammer, 2002; Kremer, 2001). This is possible because the context of recursive computations is always a subset of the outputs of the neurons in previous recursive steps. Unsupervised models do not possess a dedicated output. Thus, the proposed unsupervised recursive models use different dynamical equations and fundamentally different ways to represent the context. Recently, approaches to review and unify models for unsupervised recursive processing have been presented (Barreto and Araújo, 2001; Barreto, Araújo, and Kremer, 2003;

Hammer, Micheli, and Sperduti, 2002; Hammer et al., 2004). The articles (Barreto, Araújo, and Kremer, 2003; Hammer, Micheli, and Sperduti, 2002; Hammer et al., 2004) identify the context definition as an important design criterion according to which the unsupervised recursive models can be classified. Additionally, the articles (Hammer, Micheli, and Sperduti, 2002; Hammer et al., 2004) provide one unified mathematical notation which exactly describes the dynamic of recursive models. The concrete models can be obtained by the substitution of a single function within this dynamic. The proposed set of equations constitutes a generalization of both supervised recurrent and recursive networks.

However, substantially more work has to be done to make recursive unsupervised models ready for practical applications. The articles (Barreto, Araújo, and Kremer, 2003; Hammer, Micheli, and Sperduti, 2002; Hammer et al., 2004) provide a taxonomy of sequence and structure processing unsupervised models, and the latter two articles also formulate unified mathematical equations. The contributions do not resolve under which conditions a given specific model is best suited. In addition, they do not identify the inherent similarity measure induced by the models. Thus, the mathematics behind these models is only vague and a sound theoretical foundation of their applicability is still missing.

The classical SOM computes a mapping of data points from a potentially high-dimensional manifold into two dimensions in such a way that the topology of the data is preserved as much as possible. Therefore, SOMs are often used for data visualization. However, the standard SOM heavily relies on the metric for the data manifold, and its semantic meaning, expressed by the specific projections of data clusters, is related to this metric. For structure processing recursive models a metric for characterizing the data manifold is not explicitly chosen. Instead, a similarity measure arises implicitly by the recursive processing. It is not clear what semantics is connected to the visualization of structured data in a two-dimensional recursive SOM. Also the representational capabilities of the models are hardly understood: for example, which model of recursive unsupervised maps can represent the temporal context of time series in the best way? Experimentally, obvious differences between the model architectures can be observed, but no direct comparison of all models has been made so far. The internal model representation of structures is unclear, and it is also difficult to tell the differences between the architectures.

The purpose of this article is to give a unified overview and investigation of important recurrent and recursive models based on the context definition. The notation introduced in (Hammer, Micheli, and Sperduti, 2002; Hammer et al., 2004) is used and extended to one new context model. We point out which training models and lattice structures can be combined with specific context models, and we shortly discuss how the models can be trained. The core of this article is given by a comparison of the models from a mathematical point of view which is complemented by experimental comparisons. We investigate how structures are internally represented by the models and which metrics are thus induced on structures. In addition, the representational capabilities are considered, and fundamental differences of the models are proved. We shortly discuss the important aspects of topology preservation and noise tolerance. Since most concrete models have been proposed for sequential data only, we compare the models by executing Voegtlin's experiment for time-series data with all models (Voegtlin, 2002). Finally, one experiment is added to illustrate the applicability of unsupervised recursive learning to tree-structured data.

2 General dynamics

Self-organizing models are based upon two basic principles: a winner-takes-all dynamic of the network for mapping input patterns to specific positions in the map, and a Hebbian learning mechanism with neighborhood cooperation. The standard Kohonen SOM represents real-valued input vectors in a topologically faithful way on the map. Inputs are points from $\mathbb{R}^n$, and this vector space is characterized by a similarity measure d_W which is usually the standard squared Euclidean metric $d_W(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{n}(x_i - y_i)^2$. $x_1, \ldots, x_n$ refer to the components of a n-dimensional vector $\mathbf{x}$. The SOM consists of a set of neurons enumerated by $1, \ldots, N$. A weight $\mathbf{w}_i \in \mathbb{R}^n$ is attached to neuron i specifying the center of the neuron's receptive field. The winner-takes-all dynamic is given by a simple rule which maps an input signal to the best matching unit, the winner:

$$\mathbf{x} \mapsto I(\mathbf{x}) = \mathrm{argmin}_{i \in \{1,\ldots,N\}} d_W(\mathbf{w}_i, \mathbf{x}) \,.$$

Training accounts for topology preservation that corresponds to the neighborhood structure of the neurons *nh* $: \{1, \ldots, N\} \times \{1, \ldots, N\} \to \mathbb{R}$. Often neurons are arranged in a regular low-dimensional grid and the neighborhood is related to the distance of the neurons in that grid. Training initializes the neuron weights at random and then adapts iteratively all neurons according to a presented pattern $\mathbf{x}$ by the following rule:

$$\triangle \mathbf{w}_i = -\eta(nh(i, I(\mathbf{x}))) \cdot \frac{\partial d_W(\mathbf{w}_i, \mathbf{x})}{\partial \mathbf{w}_i} \,,$$

where $I(\mathbf{x})$ is the index of the best matching unit, η is the learning rate which decreases with increasing distance of the current unit i from the winner unit, and the direction of adaptation is determined by the gradient with respect to $\mathbf{w}_i$ of the distance of the signal $\mathbf{x}$ and the weight $\mathbf{w}_i$. We assume that this gradient of d_W is defined, which is given by the direction $(\mathbf{w}_i - \mathbf{x})$ for the squared Euclidean metric (ignoring constant factors).

Many alternatives to the original SOM have been proposed. Popular models introduce a different topological structure of the map or modified neighborhood cooperation: simple vector quantization does not rely on a topology and adapts only the best matching unit at each step (Kohonen, 1997); usually, this setting is taken for batch processing, i.e. adaptation accounting at each update step for all training points (Linde, Buzo, and Gray, 1980). However, this procedure is very sensitive to the initialization of the prototypes, and it is therefore often modified by utilizing soft assignments (Bezdek et al., 1987). The hyperbolic SOM introduced by Ritter uses a hyperbolic lattice structure to define the neighborhood structure of neurons (Ritter, 1999). A hyperbolic lattice differs from a Euclidean lattice in the fundamental property that in a hyperbolic lattice the number of neighbors of a neuron increases exponentially for linearly growing neighborhood sizes, whereas for Euclidean lattices the number of neighbors follows a power law. Both grid types share the property of easy visualization, but hyperbolic lattices are particularly suited for the visualization of highly connected data such as documents or web structures (Ontrup and Ritter, 2001). Neural gas and topology representing networks form other popular alternatives which develop a data-driven optimal lattice during training (Martinetz, Berkovich, and Schulten, 1993; Martinetz and Schulten, 1993). A changing neighborhood structure is defined dynamically in each training step based on the distance of the neurons from the presented pattern. The final neighborhood structure is data optimal. However, the resulting neighborhood structure cannot be directly visualized, because arbitrarily shaped, high-dimensional lattices can arise depending on the underlying data set. Neural gas has got the benefit that its learning dynamic can be interpreted as a stochastic gradient descent of an energy cost function. It is well known that the SOM learning rule given above can be interpreted only as an approximate gradient descent of a cost function related to the quantization error (Heskes, 2001), but it does not possess an exact cost function for continuous input distributions (Erwin, Obermayer, and Schulten, 1992).

Here, we are interested in generalizations of self-organizing models to more complex data structures, sequences and tree structures. The key issue lies in an expansion of the winner-takes-all dynamic of SOM from vectors to the more complex data structures. In this case, training can be directly transferred to the new winner computation. Since the winner-takes-all dynamic of SOM is independent of the choice of the neighborhood, we first focus on the definition of the dynamic and its implication on the internal representation of structures, and we say a few words on expansions to alternative lattice models later on.

2.1 Recursive winner-takes-all dynamics

The data types we are interested in are sequences and tree structures. A sequence s over an alphabet W, e.g. $W = \mathbb{R}^n$, is denoted by $s = (\mathrm{s}^1, \dots, \mathrm{s}^t)$ with elements $\mathrm{s}^i \in W$ and t being the length of the sequence. The empty sequence is referred to by ( ), and the set of sequences over W is denoted by W^*. Sequences are a natural way to represent temporal or spatial data such as language, words, DNA-sequences, economical time series, etc. Trees constitute a generalization of sequences for expressing branching alternatives. We confine our considerations to trees with limited fan-out k. Then a tree over a set W is either the empty tree ξ, or it is given by a root vertex v with label $l(v) \in W$, and k subtrees $t_1, \dots, t_k$ which might be empty. We represent such a tree in prefix notation by $l(v)(t_1, \dots, t_k)$ and we address trees by their root vertices in the following. The set of trees with fan-out k over W is denoted by $Tree_W^k$. The height of a vertex is the length of a longest path from the root to the vertex. A leaf is a vertex with only empty successors. The height of a tree is the maximum height of its vertices. For simplicity, we will restrict ourselves to $k = 2$ in the following, because the formulation and the results for larger k are analogous. Note that symbolic terms can be represented as tree structures. Hence, we can model data in both logic and structural domains with this approach. In addition, acyclic directed graphs can often be represented as trees by rooting the graph or by adding one supersource. This makes chemistry or graphical image processing interesting application areas.

We want to process this kind of structured data by self-organizing models. A popular way to directly deal with data structures has been proposed for supervised network models: the recursive architectures are used to process sequences and tree structures in a natural way (Frasconi et al., 2001; Frasconi, Gori, and Sperduti, 1998; Kremer, 2001; Sperduti and Starita, 1997). Assume that a sequence $(\mathbf{s}^1, \ldots, \mathbf{s}^t)$ is given. Then the functional dependence of a neuron $n_i(t)$ of a recurrent network for an input sequence given till time step t has the form $f(\mathbf{s}^t, n_1(t-1), \ldots, n_N(t-1))$, i.e. the function value depends on the current entry $\mathbf{s}^t$ and the values of all neurons $n_1, \ldots, n_N$ in the previous time step. Analogously, trees with root label $l(v)$ and children $ch_1(v)$ and $ch_2(v)$ are mapped by a recursive network to an output of neuron n_i with functional dependence

$$f(l(v), n_1(ch_1(v)), \ldots, n_N(ch_1(v)), n_1(ch_2(v)), \ldots, n_N(ch_2(v)))\ ,$$

i.e. the output depends on the value of the current vertex of the tree and the already computed values of the two children for all other neurons $n_1, \ldots, n_N$. The function f is usually a simple combination of an adaptive linear function and a fixed nonlinearity. The parameters of the function f are trained during learning according to patterns given in the supervised scenario. Networks of this type have been successfully used for different applications such as natural language parsing, learning search heuristics for automated deduction, protein structure prediction, and problems in chemistry (Bianucci et al., 2000; Baldi et al., 1999; Costa et al., 2003). All information within the structure is directly or indirectly available when processing a tree, and the models constitute universal approximators for standard activation functions (Hammer, 2000). Note that although several different formulations of recurrent and recursive networks exist, the models share the basic dynamic and the notations are mainly equivalent (Hammer, 2000; Kremer, 2001).

The same idea can be transferred to unsupervised models: complex recursive structures can be processed by unsupervised models using recursive computation. The functional dependence of all neurons' activations for a given sequence or tree structure can be transferred in principle from the supervised to the unsupervised case: in the case of sequences the value of a given neuron depends on the current sequence entry and the outputs of the neurons at the previous time step; in the case of tree structures the label of the considered vertex in a tree and the outputs of the neurons for the two children are of interest. However, in contrast to the supervised case, the transfer function cannot be adapted according to an error measure, because desired output values are not available in the considered unsupervised scenario. The connection of the output values of the neurons in an unsupervised map to semantic information about the map is not clear. Because of this fact, a variety of fundamentally different unsupervised recursive models has been proposed, see e.g. the articles (Barreto and Araújo, 2001; Barreto, Araújo, and Kremer, 2003) for an overview. Several questions arise: Which different types of unsupervised recursive models exist? What are the differences and the similarities of the approaches?

Which model is suited for specific applications? What are the consequences of design criteria on learning?

As pointed out in (Hammer, Micheli, and Sperduti, 2002; Hammer et al., 2004) the notation of context plays a key role for unsupervised recursive models: this handle makes it possible to identify one general dynamic of recursive unsupervised models which covers several existing approaches and which also includes supervised models. Different context models yield distinct recursive unsupervised networks with specific properties. We informally define the general dynamic of unsupervised networks for sequences to explain the basic idea. Afterwards, we give a formal definition for the more general case of tree structures.

Assume that a sequence $(\mathbf{s}^1, \ldots, \mathbf{s}^t)$ over W is given and that d_W denotes the similarity measure on the sequence entries, e.g. the standard squared Euclidean metric if $W = \mathbb{R}^n$. In the following, let W be embedded in a real-valued vector space. The neural map contains neurons $n_1, \ldots, n_N$. Sequences are processed recursively and the value of subsequent entries depends on the already seen elements. To achieve this, the network internally stores the first part of the sequence. A context or an interior representation of this part must thus be defined for the processing of the remaining entries. For this purpose, we introduce a set R in which the context lies. A similarity measure d_R is declared on this space to evaluate the similarity of interior context representations. Each neuron in the neural map represents a sequence in the following way: neuron i is equipped with a weight $\mathbf{w}_i \in W$ representing the expected current entry of the sequence, as for standard SOM. In addition, the neuron is given a descriptor $\mathbf{c}_i \in R$ to represent the expected context in which the current entry should occur and which refers to the previous sequence entries. Based on this context notion, the distance of neuron i to the sequence entry in time step t can be recursively computed as the distance of the current entry $\mathbf{s}^t$ and the expected entry $\mathbf{w}_i$ combined with the distance of the current context from the expected context $\mathbf{c}_i$.

Now a problem occurs, because what is the current context of the computation? The available information of the previous time step is given by the distances $\tilde{d}_1(t-1)$, $\ldots$, $\tilde{d}_N(t-1)$ of neurons 1 to N. We could just take this vector of distances which describes the activation of the map as the context. However, it might be reasonable to represent the context in a different form, e.g. to focus only on a small part of the whole map. In order to allow different implementations we introduce a general function which transforms the given information of the current time step, $\tilde{d}_1(t-1)$, $\ldots$, $\tilde{d}_N(t-1)$, to an interior representation of sequences, to focus on the relevant part of the context necessary for further processing:

$$\mathit{rep} : \mathbb{R}^N \to R$$

The specific choice of *rep* is crucial since it determines what information is preserved and which notion of similarity is imposed on the contexts. Several different approaches for internal representations have been proposed which will be introduced later. Given *rep*, the distance for sequence entry $\mathbf{s}^t$ from neuron i can be formally defined as

$$\tilde{d}_i(t) = \alpha \cdot d_W(\mathbf{w}_i, \mathbf{s}^t) + \beta \cdot d_R(\mathbf{c}_i, R_{t-1}) \ ,$$

where α, $\beta \geq 0$ are constants which determine the influence of the actual entry and the context, and R_{t-1} is the interior representation of context, i.e. R_0 is a specified value and

$$R_{t-1} = \mathit{rep}(\tilde{d}_1(t-1), \ldots, \tilde{d}_N(t-1))$$

is the internal representation of the previous time step. Based on the recursively computed distances, the winner after time step t in the map is the best matching unit

$$I(t) = \text{argmin}_{i=1,\ldots,N} \tilde{d}_i(t) \; .$$

We generalize the discussed setting to tree structures and we provide a formal definition. For simplicity, we restrict ourselves to binary trees. A generalization to trees with fan-out $k > 2$ is obvious. Note that $k = 1$ yields sequences. The main alteration of the setting for binary trees in contrast to sequences is the fact that tree structures are recursively processed from the leaves to the root, where each vertex has two children. A vertex is processed within two contexts given by its two children instead of just one child as in the case of sequences.

Definition 1 *Assume that W is a set with similarity measure d_W. A* general unsupervised recursive map *for tree structures in Tree^2_W consists of a set of neurons* 1, *…,* N *and the following choices: a set R of context representations is fixed with a corresponding similarity measure d_R. A representation function*

$$\textit{rep} : \mathbb{R}^N \to R$$

maps the activations of the neural map to internal contexts. Each neuron i is equipped with a weight $\mathbf{w}_i \in W$ and two contexts $\mathbf{c}_i^1$ and $\mathbf{c}_i^2 \in R$. The distance of neuron i from a given vertex v of a tree with label $l(v)$ and children $ch_1(v)$ and $ch_2(v)$ is recursively computed by

$$\tilde{d}_i(v) = \alpha \cdot d_W(\mathbf{w}_i, l(v)) + \beta \cdot d_R(\mathbf{c}_i^1, R_{ch_1(v)}) + \beta \cdot d_R(\mathbf{c}_i^2, R_{ch_2(v)}) \; ,$$

where $\alpha \geq 0$ and $\beta \geq 0$ are constants. For $j = 1, 2$, if $ch_j(v)$ is empty, the context *vector* $R_{ch_j}(v) = \mathbf{r}_\xi$ *is a fixed value* $\mathbf{r}_\xi$ *in* R*. Otherwise, the context vector is given by*

$$\textit{rep}(\tilde{d}_1(ch_j(v)), \ldots, \tilde{d}_N(ch_j(v)))\,.$$

The winner for the vertex v *of a tree is defined as*

$$I(v) = \operatorname{argmin}_{i=1,\ldots,N} \tilde{d}_i(v)\,.$$

Tree structures are recursively processed, and the choice of α and β determines the importance of the current entry in comparison to the contexts. The winner is the neuron for which the current vertex label and the two contexts expressed by the two subtrees are closest to the values stored by the neuron. The contexts are internally represented in an appropriate form and the choice of this interior representation and the similarity measure d_R on contexts determines the behavior of the model. Note that we did not put restrictions on the functions d_W or d_R so far and that the general dynamic is defined for arbitrary real-valued mappings. In addition, we did not yet refer to a topological structure of the neurons since we have not yet specified how training of weights and contexts takes place in this model. Now, we consider several specific context models proposed in literature.

2.2 Context models

As a first observation, the dynamic given by Definition 1 generalizes the dynamic of supervised recursive networks (Hammer et al., 2004): for supervised recursive networks, R is the space $\mathbb{R}^N$, N denoting the number of neurons. The similarity measures d_W and d_R are the standard dot products and the representation function *rep*$(x_1, \ldots, x_N) = (\tanh(x_1), \ldots, \tanh(x_N))$ applies the nonlinearity $\tanh$

of recursive networks to each component of the internally computed activation of the previous recursion step. In this case, the neuron parameters $\mathbf{w}_i$, $\mathbf{c}_i^1$, and $\mathbf{c}_i^2$ can be interpreted as the standard weights of recursive networks. They are adapted in a supervised way in this setting according to the learning task to account for an appropriate scaling of the input and context features.

Temporal Kohonen map:

For unsupervised processing, alternative choices of context have been proposed. They focus on the fact that the context determines which data structures are mapped to similar positions in the map. Most models found in literature have been defined for sequences, but some can also be expanded to tree structures. One of the earliest unsupervised models for sequences is the temporal Kohonen map (TKM) (Chappell and Taylor, 1993). The neurons act as leaky integrators, adding up the temporal differences over a number of time steps. This behavior is biologically plausible and it has successfully been applied to the task of training selection sensitive and direction sensitive cortical maps (Farkas and Miikkulainen, 1999). The recursive distance computation for a sequence $(\mathbf{s}^1, \ldots, \mathbf{s}^t)$ at time step t of neuron i is given by

$$\tilde{d}_i(t) = \alpha \cdot d_W(\mathbf{w}_i, \mathbf{s}^t) + (1 - \alpha) \cdot \tilde{d}_i(t-1) \, .$$

This fits into the general dynamic (restricted to sequences) if we set $\beta = 1 - \alpha$ and if we chose a context model which just focuses on the neuron itself, ignoring all other neurons of the map. Mathematically, this can be realized by choosing *rep* as the identity and by setting the context $\mathbf{c}_i$ attached to neuron i to the ith canonical basis vector. Then, the choice of d_R as the dot product yields the term $\tilde{d}_i(t-1)$ as second summand of the recursive dynamic.

Note that this model is quite efficient: the context vectors of a given neuron need not be stored explicitly, since they are constant. The model can be formally generalized to tree structures: given a vertex v with label $l(v)$ and children $ch_1(v)$ and $ch_2(v)$, the distance can be defined as $\alpha \cdot d_W(\mathbf{w}_i, l(v)) + (1-\alpha)/2 \cdot \tilde{d}_i(ch_1(v)) + (1-\alpha)/2 \cdot \tilde{d}_i(ch_2(v))$, for example. However, this choice does not distinguish between the children and it yields the same value for tree structures with permuted subtrees, which is thus only a limited representation. We will later see that any expansion of this context model to tree structures has only restricted capacity if the context just focuses on the neuron itself.

Recursive SOM:

A richer notion of context is realized by the recursive SOM (RecSOM) proposed by Voegtlin (Voegtlin, 2000; Voegtlin, 2002; Voegtlin and Dominey, 2001). Its recursive computation of the distance of neuron i from a sequence $(\mathbf{s}^1, \ldots, \mathbf{s}^t)$ at time step t is computed by

$$\tilde{d}_i(t) = \alpha \cdot d_W(\mathbf{w}_i, \mathbf{s}^t) + \beta \|\mathbf{c}_i - R_{t-1}\|^2 .$$

$\|\cdot\|$ denotes the standard Euclidean distance of vectors. The context vector R_{t-1} of the previous time step is given by the N-dimensional vector

$$R_{t-1} = (\exp(-\tilde{d}_1(t-1)), \ldots, \exp(-\tilde{d}_N(t-1))) ,$$

N being the number of neurons. Consequently, the representation function yields high-dimensional vectors

$$\mathit{rep}(x_1, \ldots, x_n) = (\exp(-x_1), \ldots, \exp(-x_n)),$$

and the context vector $\mathbf{c}_i$ of neuron i is also contained in $\mathbb{R}^N$. This context preserves all information available within the activation profile in the last time step, because it just applies an injective mapping to the distance computed for each neuron for the previous time step. Since $\exp$ has a limited codomain, numerical explosion of the distance vectors over time is prevented by this transformation. Note that this context model can be transferred to tree structures by setting

$$\begin{aligned}\tilde{d}_i(v) &= \alpha \cdot d_W(\mathbf{w}_i, l(v)) \\ &\quad + \beta \cdot \|\mathbf{c}_i^1 - (\exp(-\tilde{d}_1(ch_1(v))), \ldots, \exp(-\tilde{d}_N(ch_1(v))))\|^2 \\ &\quad + \beta \cdot \|\mathbf{c}_i^2 - (\exp(-\tilde{d}_1(ch_2(v))), \ldots, \exp(-\tilde{d}_N(ch_2(v))))\|^2 .\end{aligned}$$

However, like in the sequential model, high-dimensional contexts are attached to the neurons, making this architecture is computationally expensive.

SOM for structured data:

A more compact model, the SOM for structured data (SOMSD), has been proposed in (Hagenbuchner, Tsoi, and Sperduti, 2001; Hagenbuchner, Sperduti, and Tsoi, 2003; Sperduti, 2001). The information contained in the activity profile of a map is compressed by *rep* in this model: instead of the whole activity profile only the location of the last winner of the map is stored. Assume that d_R denotes the distance of neurons in a chosen lattice, e.g. the distance of indices between neurons in a rectangular two-dimensional lattice. For SOMSD we need to distinguish between a neuron index I, and the location of the neuron within the lattice structure. The location of neuron I is referred to by I^p in the following. The dynamic of SOMSD is given by

$$\tilde{d}_i(v) = \alpha \cdot d_W(\mathbf{w}_i, l(v)) + \beta \cdot d_R(\mathbf{c}_i^1, I^p(ch_1(v))) + \beta d_R(\mathbf{c}_i^2, I^p(ch_2(v))) \,,$$

where I denotes the winner of the two children, I^p its location in the map, and the contexts $\mathbf{c}_i^j$ attached to a neuron represent the expected positions of the last winners for the two subtrees of the considered vertex within the map. R is a vector space which contains the lattice of neurons as a subset; typically, R is given by $\mathbb{R}^q$ for a q-dimensional Euclidean lattice of neurons. In this case, the distance measure d_R can be chosen as the standard squared Euclidean distance of points within the lattice space. *rep*$(x_1, \ldots, x_N)$ computes the position I^p of the winner $I = \mathrm{argmin}_{i=1,\ldots,N} x_i$. This context model compresses information that is also contained in the context model of the RecSOM. For SOMSD, the position of the winner is stored; contexts that correspond to similar positions within the map refer to similar structures. For RecSOM, the same information is expanded in an activity profile of the whole map. The exponential transformation of the activity emphasizes regions of the map with high activity and suppresses regions with only small values. This way, the context model of RecSOM emphasizes the location of the winner within the map. However, more detailed information and possibly also more noise are maintained for RecSOM.

Merge SOM:

A further possibility to represent context has recently been introduced (Strickert and Hammer, 2003a): the merge SOM (MSOM) for unsupervised sequence processing. It also stores compressed information about the winner in the previous time step, and the winner neuron is represented by its content rather than by its location in the map. Unlike SOMSD, MSOM does not refer to a lattice structure to define the internal representation of sequences. A neuron i contains a weight $\mathbf{w}_i$ and a context $\mathbf{c}_i$. For the context vector these two characteristics are stored in a merged form, i.e. as linear combination of both vectors, referring to the previously active neuron. The recursive equation for the distance of a sequence $(\mathbf{s}^1, \ldots, \mathbf{s}^t)$ from neuron i is given by

$$\tilde{d}_i(t) = \alpha \cdot d_W(\mathbf{w}_i, \mathbf{s}^t) + \beta \cdot d_W(\mathbf{c}_i, \gamma \mathbf{w}_{I(t-1)} + (1-\gamma)\mathbf{c}_{I(t-1)}) \ ,$$

where $\gamma \in (0,1)$ is a fixed parameter, and $I(t-1)$ denotes the winner index of the previous time step. The space of representations R is the same space as the weight space, W, and the similarity measure d_R is identical to d_W. The representation computes *rep*$(x_1, \ldots, x_n) = \gamma \mathbf{w}_I + (1-\gamma)\mathbf{c}_I$ for $I = \operatorname{argmin}_{i=1,\ldots,N} x_i$, i.e. a weighted linear combination of the winner contents.

So far, the MSOM encoding scheme has only been presented for sequences, and the question is whether it can be transferred to tree structures. In analogy to TKM, MSOM can be extended to $\alpha \cdot d_W(\mathbf{w}_i, l(v)) + \beta \cdot (d_W(\mathbf{c}_i^1, R_{ch_1(v)}) + d_W(\mathbf{c}_i^2, R_{ch_2(v)}))$, where the contexts $R_{ch_j(v)}$ are given by $R_{ch_j(v)} = \gamma \mathbf{w}_{I(ch_j(v))} + (1-\gamma)/2 \cdot (\mathbf{c}^1_{I(ch_j(v))} + \mathbf{c}^2_{I(ch_j(v))})$, $j = 1, 2$. A drawback of this choice is that it does not distinguish between the branches of a tree, because the operation is commutative with respect to the children. Two trees resulting from each other by a permutation of the vertex labels at equal height have the same winner and the same internal representation. As a consequence, only certain tree structures can be faithfully represented in this model. An alternative though less intuitive possibility relies on the encoding of tree structures in prefix notation in which the single letters are stored as consecutive digits of a real number. A specified digit denotes an empty vertex. For example, if 9 denotes the empty vertex and labels come from the finite set $\{1, \ldots, 8\}$, the sequence 12399499599 represents the tree $1(2(3,4),5)$. This yields a unique representation. In addition, trees can be represented as real values of which consecutive digits correspond to the single entries. In such a setting, merging of context would correspond to a concatenation of the single representations, i.e. a label $l(v)$ and representation strings s_1 and s_2 for the two subtrees would result in the prefix representation $l(v)s_1s_2$ of the entire tree. This operation is no longer a simple addition of real numbers but rather complex; therefore, the approach is not very efficient in practice. However, since the operation is no longer commutative, trees can be reliably represented with a generalization of MSOM in principle.

We have introduced four different context models:

(1) a reference to the neuron itself as proposed by TKM,
(2) a reference to the whole map activation as proposed by RecSOM,
(3) a reference to the winner index as proposed by SOMSD, and
(4) a reference to the winner content as proposed by MSOM.

Of course, the choice of the context is crucial in this setting. As we will discuss in the following, this choice has consequences on the representation capabilities of the models and on the notion of similarity induced on the structures. One difference of the models, however, can be pointed out already at this point: the models differ significantly with respect to their computational complexity. For standard SOM the storage capacity required for each neuron is just n, the dimensionality of the single entries. For recursive unsupervised models, the storage size is (k is 1 for sequences and 2 for trees)

(1) only n, no further memory requirement for TKM,
(2) $n + k \cdot N$, the number of neurons for RecSOM,
(3) $n + k \cdot q$, the lattice dimension for SOMSD,

(4) $n + k \cdot n$, the entry dimension for MSOM.

RecSOM is very demanding, because neural maps usually contain of hundreds or thousands of neurons. SOMSD is very efficient, because q is usually small like $q = 2$ or $q = 3$. The storage requirement of MSOM is still reasonable, although usually larger than the storage requirement of SOMSD.

2.3 Training

How are these models trained? Each neuron is equipped with a weight and one or two context vectors which have to be adapted according to the given data. A general approach taken by RecSOM, SOMSD, and MSOM is Hebbian learning, i.e. a direct transfer of the original SOM update rule to both weights and contexts. We assume that the similarity measures d_W and d_R are differentiable. Neighborhood cooperation takes place during learning, thus we specify a neighborhood structure

$$nh : \{1, \ldots, N\} \times \{1, \ldots, N\} \to \mathbb{R}$$

of the neurons. As already mentioned, often the neighborhood structure is given by a low-dimensional rectangular lattice. Having presented a vertex v with label $l(v)$ and subtrees $ch_1(v)$ and $ch_2(v)$, the winner $I(v)$ is computed, and Hebbian learning is conducted by the update formulas

$$\triangle \mathbf{w}_i = -\eta(nh(i, I(v))) \cdot \frac{\partial d_W(\mathbf{w}_i, l(v))}{\partial \mathbf{w}_i} \, ,$$

$$\triangle \mathbf{c}_i^1 = -\eta(nh(i, I(v))) \cdot \frac{\partial d_R(\mathbf{c}_i^1, R_{ch_1(v)}))}{\partial \mathbf{c}_i^1} \, ,$$

$$\triangle \mathbf{c}_i^2 = -\eta(nh(i, I(v))) \cdot \frac{\partial d_R(\mathbf{c}_i^2, R_{ch_2(v)}))}{\partial \mathbf{c}_i^2} \, ,$$

where η is a positive learning rate function which is largest for winner and its immediate neighbors, e.g. the values of a Gaussian function $\eta(x) = c_1 \cdot \exp(-x^2/c_2)$, c_1 and c_2 being positive constants. $R_{ch_j(v)}$ ($j = 1, 2$) is the interior representation of the context, i.e. $R_{ch_j(v)} = \mathit{rep}(\tilde{d}_1(ch_j(v)), \ldots, \tilde{d}_N(ch_j(v)))$. If the squared Euclidean metric is applied, the partial derivatives can be substituted by the terms $(\mathbf{w}_i - l(v))$ and $(\mathbf{c}_i^j - R_{ch_j(v)}))$ (ignoring constants), respectively. This way, the familiar Hebb terms result, if representations are considered constant, i.e. weights and contexts are moved towards the currently presented input signal and the currently computed context in the map.

This learning rule has been used for the models RecSOM, SOMSD, and MSOM, and it leads to topological ordering of the neurons on the map. Since TKM uses contexts only implicitly, no adaptation of the contexts takes place in this case, and only standard SOM training of the weights is applied after each recursive step. An alternative learning rule has been proposed for a very similar model, the recurrent SOM (RSOM), which yields better results than TKM (Koskela et al., 1998a). Later we will discuss why this is the case and why the learning rule of TKM only leads to suboptimal results.

The question of interest is whether the heuristically motivated Hebbian learning rule can be justified from a mathematical point of view. It is well known that the original SOM learning dynamic does not possess a cost function in the continuous case, which makes the mathematical analysis of learning difficult (Cottrell, Fort, and Pagès, 1994). However, a cost function for a modified learning rule, which relies on a slightly different notion of the winner, has been proposed. This modification can be formally transferred to our case (Heskes, 2001):

$$E = \frac{1}{2} \sum_{i,v} \delta(i,v) \sum_{j=1}^{N} \eta(nh(i,j)) \tilde{d}_j(v) ,$$

where the sum is over all neurons i and vertices v in the considered pattern set, and where δ indicates the (modified) winner

$$\delta(i,v) = \begin{cases} 1 & \text{if } \sum_{j=1}^{N} \eta(nh(i,j))\tilde{d}_j(v) \text{ is minimum} \\ 0 & \text{otherwise} \end{cases}$$

The winner is the neuron for which the above average distance is minimum. Can the modified Hebbian learning rule for structures be interpreted as a stochastic gradient descent of this cost function? Does the learning converge to optimum stable states of E? These questions have been solved in (Hammer et al., 2004): in general, Hebbian learning is not an exact gradient descent of this cost function, but it can be interpreted as an efficient approximate version which disregards contributions of substructures to E.

Alternatives to the original SOM have been proposed which differ with respect to the implemented lattice structure. Specific lattices might be useful for situations in which the data manifold is too complex to be faithfully mapped to a simple Euclidean lattice. For the standard case, different methodologies for dynamic lattice adaptation or for the detection of topological mismatches have been proposed (Bauer and Pawelzik, 1992; Bauer and Villmann, 1997; Villmann et al., 1997). For the structured case discussed here we consider three alternative lattice models .

Vector quantization:

Simple vector quantization (VQ) does not use a lattice at all; instead, only the winner is adapted at each training step. The learning rule is obtained by choosing the neighborhood function as

$$nh(i,j) = \begin{cases} 1 & \text{if } i = j \\ \\ 0 & \text{otherwise} \end{cases}$$

This modified neighborhood definition can be obviously integrated into the Hebbian learning rules also for structure processing SOMs. Hence, we can transfer simple VQ to the structured case. Note, however, that semantic problems arise when VQ is combined with the context of SOMSD: internally, SOMSD refers to structure representations by locations on the map, and it refers to the distance of those locations by recursive winner computation. Topology preservation is not accounted for by the training rule of VQ, and the position of the winner has no semantic meaning: close indices of neurons do not imply that the represented structures are similar. As a consequence, the similarity measure d_R used for SOMSD yields almost random values when it is combined with VQ. Therefore, faithful structure representations cannot be expected in this scenario. For the other context models VQ can be used, because the topology of the map is not referred to in these context models. Note that the main problem of standard VQ, which is an inappropriate learning for slovenly initialized neurons, transfers to the case of structures.

Neural Gas:

Another alternative to SOM is the topology representing network called neural gas (NG) (Martinetz, Berkovich, and Schulten, 1993; Martinetz and Schulten, 1993). A prominent advantage of NG is that a prior lattice does not need to be specified, because a data optimal topological ordering is determined in each training step. The learning rule of NG can be obtained by substituting the learning rate $\eta(nh(i, I(v)))$

for a given vertex v by

$$\eta(\mathit{rk}(i, v)) \ ,$$

where $\mathit{rk}(i, v)$ denotes the rank of neuron i when ordered according to the distance from a given vertex v of a tree, i.e.

$$\mathit{rk}(i, v) = |\{j \in \{1, \ldots, N\} \mid \tilde{d}_j(v) < \tilde{d}_i(v)\}| \ .$$

As beforehand, $\eta(x)$ is a function which is maximum for small values $x \geq 0$, e.g. $\eta(x) = c_1 \cdot \exp(-x/c_2)$. During learning the update results in optimal topological ordering of the neurons. Neighborhood-based visualization of the data is no longer possible, but NG can serve as an efficient and reliable clustering and preprocessing algorithm which unlike VQ is not sensitive to the neuron initialization. We can include this alternative dynamic neighborhood directly into the Hebb terms for structure processing models: the learning rate is substituted by the term introduced above. However, like in the case of VQ, it can be expected that the combination of NG with the context of SOMSD does not yield meaningful representations since no semantic meaning is connected to its distance measure d_R. The other context models can be combined with NG in a straight forward manner.

Hyperbolic SOM:

A third alternative grid structure is realized by the hyperbolic SOM (HSOM) proposed in (Ontrup and Ritter, 2001; Ritter, 1999). The HSOM implements its lattice by a regular triangulation of the hyperbolic plane. Since projections of the hyperbolic plane into the standard Euclidean plane are possible in form of a ‘fish-eye’ view, easy visualization is maintained. The main difference between a hyperbolic grid and a standard Euclidean lattice is an exponentially increasing neighborhood size. The number of neighbors of a neuron can increase exponentially as a function of the distance between neurons on the lattice, whereas the neighborhood growth of regular lattices in an Euclidean space follows a power law. A hyperbolic neighborhood structure can be obtained in Hebbian learning by setting the neighborhood function $nh(i, j)$ to the distance of lattice points computed in the hyperbolic space. Obviously, this is also possible for structure processing models. In addition, the lattice structure is accounted for by the training algorithm; therefore, reasonable behavior can be expected for the SOMSD context, if the 2-dimensional Euclidean context coordinates are just replaced by their 2-dimensional hyperbolic counterparts and if the corresponding distance measure is used.

Tab. 1 summarizes the possibilities to combine lattice structures and context models.

Tab. 1 about here

3 Representation of structures

The chosen context model is the essential part of general unsupervised structure processing networks introduced above. It determines the notion of structure similarity: data structures which yield similar internal representations are located at similar positions of the map. Having introduced general recursive unsupervised models, several concrete context models, and possibilities of training, we turn to a mathematical investigation of these approaches now. Thereby, we focus on the fundamental aspect how structures are internally presented by the models. First, we investigate the encoding schemes used by the models. We study explicit characterizations of the encoding mechanism and we investigate the closely related question of metrics on the space of trees induced by these models. Afterwards, we turn to the question of the model capacities, i.e. the question how many structures can be represented using certain resources. Interestingly, one can go a step further and investigate the possibility to represent tree automata by SOMSD. Finally, we have a short look at the issues of noise tolerance and topology preservation.

3.1 Encoding

The proposed context models can be divided into two different classes: TKM and MSOM represent context in the weight space W, whereas RecSOM and SOMSD extend the representation space to a characteristic of the activation profile of the entire map: the activity profile of the map itself and the (compressed) location of the previous winner.

Representation in weight space

We turn to the representation in weight space first. Since TKM and MSOM have been proposed for sequences and a generalization to tree structures is not obvious, we focus on the representation of sequences. For notational simplicity, we assume that all presented values are elements of just one sequence of arbitrary length, i.e. an input signal $\mathbf{s}^t$ is uniquely characterized by the time point t of its presentation. For TKM, one can explicitly compute the weight vector $\mathbf{w}_i$ with maximum response to $\mathbf{s}^t$ (Koskela et al., 1998b). If d_W is the squared Euclidean distance, then the optimal weight vector for entry $\mathbf{s}^t$ has the form

$$\mathbf{w}_i = \frac{\sum_{j=0}^{t-1}(1-\alpha)^j \mathbf{s}^{t-j}}{\sum_{j=0}^{t-1}(1-\alpha)^j} \ .$$

This yields a fractal encoding known as Cantor set. The parameter α of the dynamic determines the influence of the current entry relative to the context. As can be seen from this explicit representation of the optimal weight vector, α also defines the internal representation of sequences. In particular, this parameter determines whether the representation is unique or whether different sequences are mapped to the same optimal values. This explicit representation of the weight with optimal response allows to assess the similarity induced on sequences: sequences are mapped to neighboring neurons of the map (with similar weights) if the associated terms $(\sum_{j=0}^{t-1}(1-\alpha)^j \mathbf{s}^{t-j})/(\sum_{j=0}^{t-1}(1-\alpha)^j)$ for two sequences are similar. Thus, sequences are similar if their most recent entries are similar, where the importance of the entries decreases by the factor $(1-\alpha)$ for each further step into the past.

This explicit representation of the weights with optimal response points to another important issue: the optimal weights according to this context model are usually not found during Hebbian training for TKM, because these weights do not constitute a fixed point of the learning dynamic. It can thus be expected that TKM finds appropriate representations of structures only in very limited situations, which has been pointed out already in (Koskela et al., 1998b). An alternative learning rule is given by the recurrent SOM (RSOM), a modification of TKM (Koskela et al., 1998a; Koskela et al., 1998b). For RSOM, distance vectors are integrated rather than only scalar distances, and the recursive learning rule is modified in such a way that the internal representation is adapted towards the vector of integrated distances as discussed in (Koskela et al., 1998b; Varsta et al., 2001). However, the tight coupling of internal representation and the winner dynamic through α remains.

Now we investigate the MSOM and show that MSOM yields the same internal encoding as TKM and RSOM, but unlike TKM, this encoding is a stable fixed point of the training dynamic.

Theorem 2 *Assume that an MSOM with recursive winner computation*

$$\tilde{d}_i(t) = \alpha \cdot d_W(\mathbf{w}_i, \mathbf{s}^t) + \beta \cdot d_W(\mathbf{c}_i, \gamma \mathbf{w}_{I(t-1)} + (1-\gamma)\mathbf{c}_{I(t-1)})$$

is given, whereby d_W is the standard squared Euclidean metric. We set the initial context to $\mathbf{w}_{I(0)} := \mathbf{c}_{I(0)} := \mathbf{0}$. Assume that a sequence with entries $\mathbf{s}^i$ for $i \geq 1$ is presented. If enough neurons are available and if the vectors $\sum_{j=1}^{t} \gamma(1-\gamma)^{j-1}\mathbf{s}^{t-j}$ are pairwise different for each t, then the choice of the weight and context of the winner for time point t by

$$\mathbf{w}_{\mathrm{opt}(t)} = \mathbf{s}^t, \quad \mathbf{c}_{\mathrm{opt}(t)} = \sum_{j=1}^{t-1} \gamma(1-\gamma)^{j-1}\mathbf{s}^{t-j}$$

constitutes a stable fixed point of the learning dynamic, if neighborhood cooperation is neglected, i.e. for late stages of the training.

PROOF. We assume that for each sequence entry t a separate winner neuron is available. One can see by induction over t that the above choice for $\mathbf{w}_{\mathrm{opt}(t)}$ and $\mathbf{c}_{\mathrm{opt}(t)}$ yields optimal response for sequence entry t:

For $t = 1$ we find

$$d_{\mathrm{opt}(1)}(1) = \alpha d_W(\mathbf{s}^1, \mathbf{s}^1) + \beta d_W(\mathbf{0}, \mathbf{0}) = 0.$$

For larger t, we find

$$d_{\mathrm{opt}(t)}(t) = \alpha d_W(\mathbf{s}^t, \mathbf{s}^t) +$$

$$\beta d_W\left(\sum_{j=1}^{t-1} \gamma(1-\gamma)^{j-1}\mathbf{s}^{t-j}, \gamma \cdot \mathbf{s}^{t-1} + (1-\gamma) \cdot \sum_{j=1}^{t-2} \gamma(1-\gamma)^{j-1}\mathbf{s}^{t-1-j}\right)$$

by induction. Since the second argument of d_W yields $\sum_{j=1}^{t-1} \gamma(1-\gamma)^{j-1}\mathbf{s}^{t-j}$, the distance is 0, and thus the response of a neuron with weight $\mathbf{s}^t$ and context $\mathbf{c}_{\mathrm{opt}(t)}$ is optimum. Obviously, 0 can only be achieved for this choice of weights, because we assume that the context values are pairwise different.

Since the above terms are continuous with respect to the parameters $\mathbf{w}_i$ and $\mathbf{c}_i$, we can find a positive value ϵ in such a way that a neuron with weight and context within an ϵ-range of the optimal choice is still the winner for the sequence entry. Now we show that every setting in which all weights and contexts are at most ϵ away from the optimal points converge to the optimal values. The optimal context for entry t is denoted by $\mathbf{c}_{\mathrm{opt}(t)}$, the optimal weight is $\mathbf{s}^t$. We neglect neighborhood cooperation; thus, only the winner $I(t)$ is updated. The distance of the updated value from the optimum is computed by

$$\|\mathbf{w}_{I(t)} + \eta(\mathbf{s}^t - \mathbf{w}_{I(t)}) - \mathbf{s}^t\| = (1-\eta)\|\mathbf{w}_{I(t)} - \mathbf{s}^t\|.$$

Since $\eta \in (0,1)$, the weight $\mathbf{w}_{I(t)}$ converges exponentially fast to $\mathbf{s}^t$. The distance of the optimal context value from the context updated in one step is

$$\|\mathbf{c}_{I(t)} + \eta \cdot ((\gamma \mathbf{w}_{I(t-1)} + (1-\gamma)\mathbf{c}_{I(t-1)}) - \mathbf{c}_{I(t)}) - \mathbf{c}_{\mathrm{opt}(t)}\|$$

$$\leq (1-\eta)\|\mathbf{c}_{I(t)} - \mathbf{c}_{\mathrm{opt}}(t)\| + \eta\|(\gamma \mathbf{w}_{I(t-1)} + (1-\gamma)\mathbf{c}_{I(t-1)}) - \mathbf{c}_{\mathrm{opt}}(t)\|.$$

We can expand the optimal context for the previous time step; then the second summand becomes

$$\eta\|\gamma \mathbf{w}_{I(t-1)} + (1-\gamma)\mathbf{c}_{I(t-1)} - \gamma \mathbf{s}^{t-1} - (1-\gamma)\mathbf{c}_{\mathrm{opt}(t-1)}\|$$

$$\leq \eta\gamma\|\mathbf{w}_{I(t-1)} - \mathbf{s}^{t-1}\| + \eta(1-\gamma)\|\mathbf{c}_{I(t-1)} - \mathbf{c}_{\mathrm{opt}(t-1)}\|.$$

$\mathbf{w}_{I(t-1)}$ converges exponentially fast to $\mathbf{s}^{t-1}$ as we have just seen, thus the first term gets arbitrarily small. We can assume that the first term is smaller than $\|\mathbf{c}_{I(t-1)} - \mathbf{c}_{\mathrm{opt}(t-1)}\|/2$. Assuming further that all contexts $\mathbf{c}_{I(t)}$ are at most ϵ apart from the optimal values, we get the overall bound

$$(1-\eta)\epsilon + \eta(\gamma\epsilon/2 + (1-\gamma)\epsilon) = \epsilon(1-\eta\gamma/2).$$

Thus, a single update step for all contexts decreases the distance from the optimal values by the factor $(1-\eta\gamma/2)$. Hence iterative adaptation yields convergence to the optimal values. $\square$

MSOM converges to the encoding induced by RSOM and TKM: the sequences are represented by means of the exponentially weighted summation of their entries. Sequences of which the most recent entries are identical are mapped to similar codes. Unlike TKM, this encoding is a fixed point of the MSOM dynamic. Hebbian learning converges to this encoding scheme as one possible stable fixed point. The additional parameter γ in MSOM allows to control the fading parameter of the encoding scheme. The internal context representation is thus controlled independently of the parameter α which controls the significance of contexts with respect to weights during training and which has an effect on the training stability. During training, it is usually necessary to focus on the current entries first, i.e. to set α to a comparably large value and to allow convergence according to the current entries of the map for time series. Gradually, the contexts are taken into account by decreasing α during training until an appropriate context profile is found. In the domain of time series with potentially deep input structures, this control strategy is necessary, because weights usually converge faster than contexts, and instability might be observed if α is small at the beginning of training. Since γ is not affected by such a control mechanism, the optimal codes for MSOM remain the same during this training. For RSOM, the codes would change by changing α.

Representation in the space of neurons

We turn to the encoding induced by RecSOM and SOMSD now: obviously, the encoding space R is correlated with the number of neurons and it increases with the map growth. How sequences or tree structures are encoded, this is explicitly given in the definition of SOMSD: the location of the winner neuron stands for one structure. Thus, a map with N neurons provides different codes for at most N structural classes. For RecSOM, the representation is similar, because the activity profile is considered. The winner is the location of the activity profile with smallest distance $\tilde{d}_i(v)$, i.e. largest value $\exp(-\tilde{d}_i(v))$. More subtle differentiation might be possible, because real values are considered. Therefore the number of different codes is not restricted.

What are the implications of this representation for tree structures? Two structures are considered similar if their representation is similar. More precisely, every trained map induces a pseudometric d_P on structures where

$$d_P(v_1, v_2) := d_R(\textit{rep}(\tilde{d}_1(v_1), \ldots, \tilde{d}_N(v_1)), \textit{rep}(\tilde{d}_1(v_2), \ldots, \tilde{d}_N(v_2)))$$

measures the distance of the internal representations of vertices v_1 and v_2 related to given tree structures. This is a pseudometric if d_R itself is a pseudometric. Can this similarity measure be formulated explicitly for a given map? In other words, a similarity measure on the tree structures is desired that does not refer to the recursively computed representation of trees. Under certain assumptions, we can find an approximation for the case of SOMSD as follows:

Theorem 3 *Assume that d_W and d_R are pseudometrics. Assume that a trained SOMSD is given such that the following two properties hold:*

(1) The map has granularity ϵ, i.e. for each triple $(l(v), R_{ch_1(v)}, R_{ch_2(v)})$ of a vertex label and representations R of subtrees which occurs during a computation a neuron i can be found with weight and contexts closer than ϵ to that triple, i.e. $d_W(l(v), \mathbf{w}_i) \leq \epsilon$, $d_R(R_{ch_1(v)}, \mathbf{c}_i^1) \leq \epsilon$, and $d_R(R_{ch_2(v)}, \mathbf{c}_i^2) \leq \epsilon$.

(2) The neurons on the map are ordered topologically in the following sense: for each two neurons i and j and their weights and contexts $(\mathbf{w}_i, \mathbf{c}_i^1, \mathbf{c}_i^2)$ and $(\mathbf{w}_j, \mathbf{c}_j^1, \mathbf{c}_j^2)$ the distance of the neurons in the lattice can be related to the distance of its contents, i.e. an ϵ' and positive scaling terms α', β' can be found such that $|d_R(i,j) - (\alpha' d_W(\mathbf{w}_i, \mathbf{w}_j) + \beta' d_R(\mathbf{c}_i^1, \mathbf{c}_j^1) + \beta' d_R(\mathbf{c}_i^2, \mathbf{c}_j^2))| \leq \epsilon'$.

Then

$$|d_P(v_1, v_2) - d_T(v_1, v_2)| \leq$$

$$(\epsilon' + 2\alpha'\epsilon(1 + 2\beta/\alpha) + 4\beta'\epsilon(\alpha/\beta + 2)) \cdot \frac{1 - (2\beta')^{H+1}}{1 - 2\beta'} ,$$

where H is the minimum of the height of v_1 and v_2, and the pseudometric d_T is given by

$$d_T(v, \xi) = d_T(\xi, v) = d_R(\mathbf{r}_\xi, I^p(v))$$

for the empty tree. $I^p(v)$ denotes the position of the winner of v and $\mathbf{r}_\xi$ denote the *representation of the empty vertex ξ, and*

$$d_T(v_1, v_2) =$$

$$\alpha' d_W(l(v_1), l(v_2)) + \beta' d_T(ch_1(v_1), ch_1(v_2)) + \beta' d_T(ch_2(v_1), ch_2(v_2))$$

for nonempty vertices v_1 and v_2.

PROOF. The proof is carried out by induction over the height of the vertices v_1 and v_2. If v_1 or v_2 is empty, the result is immediate. Otherwise, we find

$$\begin{aligned}
&|d_P(v_1, v_2) - d_T(v_1, v_2)| \\
&= |d_R(I^p(v_1), I^p(v_2)) - \alpha' d_W(l(v_1), l(v_2)) \\
&\quad - \beta' d_T(ch_1(v_1), ch_1(v_2)) - \beta' d_T(ch_2(v_1), ch_2(v_2))| \\
&\leq \epsilon' + \alpha' |d_W(\mathbf{w}_{I(v_1)}, \mathbf{w}_{I(v_2)}) - d_W(l(v_1), l(v_2))| \\
&\quad + \beta' |d_R(\mathbf{c}^1_{I(v_1)}, \mathbf{c}^1_{I(v_2)}) - d_T(ch_1(v_1), ch_1(v_2))| \\
&\quad + \beta' |d_R(\mathbf{c}^2_{I(v_1)}, \mathbf{c}^2_{I(v_2)}) - d_T(ch_2(v_1), ch_2(v_2))| =: (*) \,.
\end{aligned}$$

$I(v_1)$ is winner for v_1, thus the weights of this neuron are closest to the current triple $(l(v_1), I^p(ch_1(v_1)), I^p(ch_2(v_1)))$. Since we have assumed granularity ϵ of the map, the weighted distance computed in this step can be at most $\epsilon(\alpha + 2\beta)$. Thus, $d_W(\mathbf{w}_{I(v_1)}, l(v_1)) \leq \epsilon(\alpha + 2\beta)/\alpha$, and $d_R(\mathbf{c}^1_{I(v_1)}, I^p(ch_1(v_1))) \leq \epsilon(\alpha + 2\beta)/\beta$ and the same holds for the second child. An analogous argumentation applies to v_2.

Using the triangle inequality we obtain

$$
\begin{aligned}
(*) \leq\ & \epsilon' + 2\alpha'\epsilon(1 + 2\beta/\alpha) + 4\beta'\epsilon(\alpha/\beta + 2) \\
& + \beta'|d_R(I^p(ch_1(v_1)), I^p(ch_1(v_2))) - d_T(ch_1(v_1), ch_1(v_2))| \\
& + \beta'|d_R(I^p(ch_2(v_1)), I^p(ch_2(v_2))) - d_T(ch_2(v_1), ch_2(v_2))| \\
\leq\ & \epsilon' + 2\alpha'\epsilon(1 + 2\beta/\alpha) + 4\beta'\epsilon(\alpha/\beta + 2) \\
& + 2\beta'(\epsilon' + 2\alpha'\epsilon(1 + 2\beta/\alpha) + 4\beta'\epsilon(\alpha/\beta + 2\epsilon)) \cdot \tfrac{1-(2\beta')^H}{1-2\beta'} \\
=\ & (\epsilon' + 2\alpha'\epsilon(1 + 2\beta/\alpha) + 4\beta'\epsilon(\alpha/\beta + 2\epsilon)) \cdot \tfrac{1-(2\beta')^{H+1}}{1-2\beta'}\ .
\end{aligned}
$$

This concludes the proof. $\square$

Note that the term $(1 - (2\beta')^{H+1})$ is smaller than 1 for β' smaller than 0.5. Then we find a universal bound independent of the height of the tree by just substituting the numerator by 1. β' is usually smaller than 0.5, because it is used to scale the contents of the neurons in such a way that they can be compared with the indices. From the approximation of d_P by d_T one can see that SOMSD emphasizes (locally) the topmost part of given trees: the induced metric provides a weighting of the previously computed distances by a factor β' for regions of the map where Theorem 3 holds.

For which regions of the map do the conditions (1) and (2) of Theorem 3 hold? In particular, how should α' and β' be chosen? (1) is satisfied if the number of neurons is large enough to cover the space sufficiently densely. For only sparsely covered regions, different structures are mapped to the same winner locations and local contortions of the metric can be expected. Condition (2) refers to an appropriate topology preservation and metric preservation of the map. The distances between neurons in the lattice and their contents have to be related. As we will discuss later, topology preservation of a map refers to the fact that the index ordering on the map is compatible with the data. During training, topology preservation is accounted for as much as possible, but we will see later that fundamental problems arise if the training examples are dense. However, neighborhood cooperation ensures that at least locally a topologically faithful mapping is achieved. Theorem 3 indicates that d_T describes the local behavior of the map within patches of topologically ordered structures. Contortions might occur, e.g. at the borders of patches corresponding to structures with different height, depending on the choice of β.

Condition (2), however, is stronger than topology preservation: not only the ordering, but also the relative distances have to be compatible, whereby α' and β' quantify possible contortions. It should be mentioned that α' and β' are not identical to the parameters α and β used for training. Instead, α' and β' are values which have to be determined from the trained map. Since the pairs (α,β) and (α',β') both determine the relevance of the root of a tree structure compared to the children, it can be expected that their relations are roughly of the same order. Since the training dynamic of SOMSD is rather complex, however, we cannot prove this claim. Apart from possible topological mismatches, another issue contributes to the complexity of the representation analysis: the standard SOM follows the underlying data density but with a magnification factor different from 1 (Claussen and Villmann, 2003; Ritter and Schulten, 1986). The magnification factor specifies the exponent of the relation of the underlying data distribution and the distribution of weights in the neural maps. The magnification factor 1 indicates that the two distributions coincide. A different magnification indicates that the map emphasizes certain regions of the underlying density. This behavior accumulates in recursive computations making the exact determination of the magnification factor difficult for the recursive SOMSD. Therefore, the similarity d_P for SOMSD differs from d_T described above in the sense that d_P reflects the statistical properties of the data distribution and it might focus on dense regions of the space. Nevertheless, d_T delivers important insights into the induced similarity measure in principle.

3.2 Capacity

Having discussed the local representation of structures within the models and the induced similarity measure on tree structures, we now turn to the capacity of the approaches. A first question is whether the approaches can represent any given number of structures provided that enough neurons are available. We use the following definition:

Definition 4 *A map* represents *a set of structures T, if for every vertex v in T a different winner $I(v)$ of the map exists.*

For the considered context models we get the immediate result:

Theorem 5 *SOMSD and RecSOM can represent every finite set T of tree structures provided that the number of neurons N equals at least the number of vertices in T. TKM and MSOM can represent every finite set T of sequence elements of different time points provided that the number of neurons is at least the number of time points in T.*

PROOF. Assume that the number of vertices in T is the number of neurons N. Then the neurons of a SOMSD representing all vertices can be recursively constructed over the height of the vertices. For a leaf v, a neuron with weight $l(v)$

and contexts $\mathbf{r}_\xi$ is the winner. For other vertices v the weight should be chosen as $l(v)$ and the contexts as $I^p(ch_1(v))$ and $I^p(ch_2(v))$, respectively. For a RecSOM, a similar construction is possible. Here, the contexts for a non-leaf vertex are chosen as $(\exp(-\tilde{d}_1(ch_i(v)), \ldots, \exp(-\tilde{d}_N(ch_i(v))))$ for $i = 1, 2$. Since the winner is different for each substructure, these vectors yield also different values.

Assume that a finite set of sequences is given; N denotes the number of sequence entries. For TKM, the weight with optimal response for $\mathbf{s}^t$ has the form $\sum_{j=0}^{t-1}(1 - \alpha)^j \mathbf{s}^{t-j} / \sum_{j=0}^{t-1}(1 - \alpha)^j$. Since only a finite number of different time points $\mathbf{s}^i$ is given in the set, one can find a value $\alpha \in (0, 1)$ such that these optimal vectors are pairwise different. Then, α and the corresponding weights yield N different winners for TKM. For MSOM, we have to choose the weights for the winner $I(t)$ of $\mathbf{s}^t$ as $\mathbf{s}^t$ and the context as $\sum_{j=1}^{t-1} \gamma(1 - \gamma)^{j-1} \mathbf{s}^{t-j}$ where γ is chosen to produce pairwise different contexts. $\Box$

Thus, SOMSD and RecSOM can in principle represent every finite set of tree structures if enough neurons are given. Codes are embedded, and TKM and MSOM can in principle represent every finite set of sequences if enough neurons are available. We have shortly discussed alternatives to extend TKM and MSOM to tree structures. For MSOM, a possibility using prefix notation of trees exist in principle. For TKM, only a very limited extension has been proposed so far. We want to accompany this observation by a general theoretical result which shows that approaches comparable to TKM are fundamentally limited for tree structures: TKM is extremely local in the sense that the recursive computation depends only on the current neuron itself but not on the rest of the map, i.e. the context is quite restricted. Now we show that *all* local approaches are restricted with respect to their representation capabilities for tree structures if they are combined with a standard

Euclidean lattice.

Definition 6 *Assume that a structure processing map with neurons* $1, \ldots, N$ *is given. The* transition function *of neuron* i *is the function* $D_i : W \times \mathbb{R}^N \times \mathbb{R}^N \to \mathbb{R}$,

$$(\mathbf{x}, \mathbf{r}_1, \mathbf{r}_2) \mapsto \alpha d_W(\mathbf{w}_i, \mathbf{x}) + \beta d_R(\mathbf{c}_i^1, \mathbf{r}_1) + \beta d_R(\mathbf{c}_i^2, \mathbf{r}_2)$$

which is used to compute the recursive steps of $\tilde{d}_i(v)$.

A transition function is local *with respect to a given set of neurons* S*, if the function* D_i *does only depend on coefficients* $(\mathbf{r}_1)_j$ *and* $(\mathbf{r}_2)_j$ *of* $\mathbf{r}_1$ *and* $\mathbf{r}_2$ *for which neuron* j *is contained in* S*. A neural map is local* with degree k corresponding to a given neighborhood topology*, if every transition function* D_i *is local with respect to* S_i^k*, where* S_i^k *refers to all neighbors of degree at most* k *of neuron* i *in the given neighborhood structure.*

In each recursive step, local neural maps refer to a local neighborhood of the current neuron within the given lattice structure. The TKM is local with degree 0, since the recursive computation depends only on the neuron itself.

Theorem 7 *Assume that the dimension* q *of a Euclidean lattice and the degree* k *of locality are fixed; further assume that distances are computed with finite precision, i.e. values* $\tilde{d}_i(v)$ *are element of* $\{0, \ldots, D\}$ *for a* $D \in \mathbb{N}$*. Then a set of trees* T *exists for which no unsupervised structure processing network with the given lattice structure and locality degree* k *can represent* T*, independent of the number of neurons in the map.*

PROOF. Consider trees with binary labels, height t, and a maximum number of vertices. There exists at least $2^{2^{t-1}}$ such trees which are of the same structure but possess different labels. This set of trees is denoted by T_t. Every neural map which represents T_t must possess at least $2^{2^{t-1}}$ neurons i with different activations, i.e. different functions $v \mapsto \tilde{d}_i(v)$ for vertices v in T_t.

Consider the transition function D_i assigned to a neuron i. If the neural map is local and the distance computation is done with finite precision, D_i constitutes a mapping of the form

$$D_i : \{0,1\} \times \{1,\dots,D\}^{(2k+1)^q} \times \{1,\dots,D\}^{(2k+1)^q} \to \{1,\dots,D\}$$

for binary labels of the tree, because a neighborhood of size k contains at most $(2k+1)^q$ neurons of which the activations are elements of the finite set $\{1,\dots,D\}$. For combinatorial reasons, there exist at most $K := D^{2D^{(2k+1)^q}D^{(2k+1)^q}}$ different functions D_i, which is constant for fixed D, k, and q.

The function $\tilde{d}_i$ which computes the distance of neuron i is a combination of several transition functions D_i. In the last recursive step only neuron i contributes. For the last but one step also the k neighbors might contribute. In the step before, all k neighbors of these neighbors might take influence, which corresponds to all neighbors of degree at most $2k$ of neuron i. Hence, for the whole computation of trees in T_t at most all neighbors of degree tk contribute. The choice of these tk transition functions uniquely determines the value of $\tilde{d}_i$ for all root vertices in T_t, because the trees in T_t have the same structure. The number of different combinations is $K^{(2tk+1)^q}$, which is smaller than $2^{2^{t-1}}$ for sufficiently large t. Thus, if we choose t large enough, not all trees can be represented by a map. This limiting result is independent of the number of neurons. $\square$

This combinatorial argument does not rely on the specific form of the transition function D_i. It holds because the number of different functions which can be achieved by combining neighboring neurons in the lattice does not increase at the same rate as the number of different tree structures in terms of the height, i.e. recursion depth of the computation. This holds for every lattice which neighborhood structure obeys a power law. For alternatives, such as hyperbolic lattices, the situation might change. Nevertheless, the above argumentation is interesting, because it allows to derive general design criteria for the transition function *rep*, criteria that are based on the general recursive dynamics: global transition functions are strictly more powerful than local functions.

Another interesting question investigated for supervised recurrent and recursive networks is which functions they can implement, if infinite input sets are applied. Classical recurrent and recursive computation models in computer science are for example: definite memory machines, finite automata, tree automata, or Turing machines. Their relation to recurrent and recursive networks has been a focus of research (Carrasco and Forcada, 1993; Gori, Küchler, and Sperduti, 1999; Hammer and Tino, 2003; Kilian and Siegelmann, 1996; Omlin and Giles, 1996). Here we show that SOMSD can implement tree automata. These considerations do not take issues of learning into account, and the map which implements a given automaton is neither topologically ordered nor achieved as a result of Hebbian learning. Instead, we focus on the capability to represent automata in principle.

Definition 8 *A (bottom-up)* tree automaton *over a finite alphabet $A = \{a_1, \ldots, a_{|A|}\}$ consists of a set of states $S = \{s_1, \ldots, s_{|S|}\}$, with initial state s_1, an accepting state $s_{|S|}$, and a transfer function $\delta : A \times S \times S \to S$. Starting at the initial state, a vertex v of a tree with labels in A is mapped to a state by recursive application of* *the transfer function* δ*:* $v \mapsto \delta(l(v), s_{ch_1(v)}, s_{ch_2(v)})$*, where* $s_{ch_1(v)}$ *and* $s_{ch_2(v)}$ *are the states onto which the children are mapped. Empty children are mapped to the initial state* $s_\xi = s_1$*. A tree is accepted by a given tree automaton, if the root vertex of the tree is mapped to the accepting state.*

Tree automata constitute canonical extensions of finite automata to tree-structured input data, and they have applications e.g. in parsing, logic programming, term rewriting, and in linguistics (Gécseg and Steinby, 1984). These automata can be simulated by appropriate SOMSD networks in the sense that a neuron in the map can be specified as the winner for exactly the trees which are accepted by a given tree automaton. Thereby, a delay is to be introduced, i.e. one step of the tree automaton corresponds to two recursive steps of the neural map. Formally, a delay can be achieved by a simple transformation of a given structure: we choose a fixed element a_e which is not yet used by the tree automaton and substitute every vertex v in a given tree by two vertices v_1 with $l(v_1) = l(v)$ and children $ch_1(v_1) = v_2$, $ch_2(v_1) = \xi$, and v_2 with $l(v_2) = a_e$ and children $ch_1(v_2) = ch_1(v)$, $ch_2(v_2) = ch_2(v)$. We denote this extension of a given tree t by t^e. Then the following holds:

Theorem 9 *Assume that a tree automaton over an alphabet* A *with states* S *is given. Then a SOMSD can be found with two-dimensional Euclidean lattice structure and* $O(|A| \cdot |S|^3)$ *involved neurons and one specific neuron* n_e *such that for every given tree* t *over* A *the following holds:* t *is accepted by the automaton* $\iff$ *the winner for* t^e *is* n_e*.*

PROOF. For simplicity, we assume that the symbols in A are the numbers $\{1, \ldots, |A|\}$. Choose the extension symbol a_e as $|A| + 1$. The symbols are embedded in

$W = \mathbb{R}$ where the squared Euclidean metric d_W is used. The states are denoted by $s_1, \dots s_{|S|}$, and again I^p refers to the position of neuron I in the map. We construct a recursive neural map with $|S|$ neurons (say $n_1, \dots, n_{|S|}$) such that for all $a \in A$ the following holds:

$$\delta(a, s_i, s_j) = s_k \iff$$

$$k = \operatorname{argmin}_l(\alpha d_W(\mathbf{w}_l, a_e) + \beta d_R(\mathbf{c}_l^1, I^p) + \beta d_R(\mathbf{c}_l^2, r_\xi)) \; ,$$

$$\text{where } I = \operatorname{argmin}_l(\alpha d_W(\mathbf{w}_l, a) + \beta d_R(\mathbf{c}_l^1, i^p) + \beta d_R(\mathbf{c}_l^2, j^p)) \; .$$

In other words, if we carry out two recursive steps of distance computation in the neural map starting at the contexts given by the neurons i and j and if we read the symbol a and afterwards the special symbol a_e, then we get neuron k as winner. This means that a delayed recursive computation step in the neural map corresponds to one operation of the transfer function δ in the tree automaton. Hence, a recursive distance computation for a delayed tree t^e by the map corresponds to the recursive application of δ to t by the tree automaton, i.e. exactly the trees accepted by the automaton have the winner $n_e := n_{|S|}$.

Now we construct this neural map. The lattice of neurons is a two-dimensional Euclidean grid. The number of neurons will be specified later. d_R is the standard squared Euclidean distance. Since the map is two-dimensional, we can identify each neuron of the map by a location in $\mathbb{R}^2$. For convenience, we use the index of the neuron or its two-dimensional representation in the following. The representation of empty vertices is $\mathbf{r}_\xi = (-1, -1)$.

We introduce two types of neurons: a neuron for each state s_i of the automaton and a neuron for each triple $(a_j, s_l, s_m) \in A \times S \times S$. The role of the latter neuron is to become winner if the input for its transition function corresponds to this triple. For every state s_i, a finite number of triples can be identified which yield s_i under δ. Thus, it suffices to choose the weights of the neuron corresponding to s_i in such a way that it is the best matching unit for all neurons corresponding to these triples.

More precisely: For every state s, a neuron n_s is fixed. Its weight is a_e and its second context is $(-1,-1)$. The value of the first context will be defined later. These neurons are located at arbitrary positions of the map. In addition, for every triple $(a_j, s_l, s_m) \in A \times S \times S$ a neuron is introduced with weight a_j and with contexts corresponding to the map locations of the neurons which represent s_l and s_m respectively. The neuron associated with the triple (a_j, s_l, s_m) becomes winner for an input if and only if the input corresponds to a_j and the winner indices of the neurons are corresponding to s_l and s_m. We can arrange these neurons on a two-dimensional lattice in such a way that the neurons corresponding to triples leading to identical s_i under δ form clusters on the map. More precisely, by possibly introducing idle units we can arrange the neurons on the map such that we find points $\mathbf{p}_1, \ldots, \mathbf{p}_{|S|}$ in $\mathbb{R}^2$ with the following property: the location of a neuron associated with the triple (a_j, s_l, s_m) is closest to $\mathbf{p}_i$ if and only if $\delta(a_j, s_l, s_m) = s_i$. Since any related cluster consists of at most $|A| \cdot |S|^2$ neurons, we need at most $O(|A| \cdot |S|^3)$ neurons to arrange all neurons in clusters on the map. Now we can define the context for neurons corresponding to states in the map: choose the first context of the neuron n_i representing state s_i by $\mathbf{p}_i$. Hence, n_i becomes winner if and only if the previous neuron is located in a cluster representing triples that yield s_i under δ.

A delayed computation within this map simulates the transition function δ of the given automaton. $\square$

As a consequence, SOMSD can simulate tree automata. However, it cannot be expected that these tree automata are learned with the proposed Hebb rule which induces a rather smooth topological neuron ordering. As shown beforehand, under the assumption of topology preservation, the induced metric for SOMSD focusing on the most recent entries of the presented structures has a Markovian flavor. Alternative learning rules might yield different solutions like tree automata discussed above.

3.3 Topology preservation

We have already mentioned the aspect of topology preservation of a trained map. This issue is important if the map is used for visualization of a high-dimensional data manifold. If the map preserves the topology, the resulting visualization allows to draw conclusions about the semantics of the data. Data at neighboring positions of the map correspond to similar data points in the original possibly high-dimensional space. Of course, topology preservation is not an issue if the data optimal adaptive lattice of neural gas is used. For a fixed lattice structure, however, such as a Euclidean lattice of the standard SOM or a hyperbolic lattice of HSOM, this issue is important. Hebbian training accounts for topology preservation as far as possible. If the data topology does not match the topology of the chosen lattice structure, topological deformations are unavoidable.

There has been a long debate on how topology preservation should be tested for a SOM with given lattice structure (Bauer and Pawelzik, 1992; Villmann et al., 1997). The rough picture for standard SOM is as follows: a data manifold with metric structure is represented in the map by the weights attached to the neurons. In addition, the neurons are arranged on a lattice with a second metric measuring the lattice distance of neurons. Topology preservation refers to the fact that two neurons are neighbors in the lattice if and only if the attached weights represent neighbors in the data manifold. Concrete possibilities to measure the degree of topology preservation are the topographic product (Bauer and Pawelzik, 1992) which might cause problems for a data manifold with large curvature, or the topographic function (Villmann et al., 1997) which is computationally more complex, but which can be applied to arbitrary data manifolds.

We will not go into details concerning the precise quantification of topology preservation. However, as a first observation, topology preservation for structure processing SOMs can be formally tested in the same way as for standard SOMs: for standard SOMs, weights of neurons should be similar if and only if their indices are close (however this can be measured). For a structure processing SOM we demand the same: the weights and contexts attached to neurons, which are usually just real vectors in Euclidean space, should be similar if and only if the neurons are neighbors in the lattice structure. The implications, though, of such a test are less clear for structured data. Unlike for standard SOM, the weights and contexts within a structure processing SOM are only compressed representatives of the underlying data manifold and, more essentially, the manifold of structures is not equipped with an explicit similarity measure in this case. Instead, the distance calculation d_W applied to single entries, the internal representations R with the distance measure d_R, and the recursive winner dynamics determine the notion of similarity on structures *implicitly*. We have already discussed for several context models how such induced explicit similarity measures look like. What implication has got topology preservation on these similarity measures?

One induced similarity measure introduced above compares two structures by the distances of their internal representations:

$$d_P(v_1, v_2) = d_R(\mathit{rep}(\tilde{d}_1(v_1), \ldots, \tilde{d}_N(v_1)), \mathit{rep}(\tilde{d}_1(v_2), \ldots, \tilde{d}_N(v_2)))$$

for two given vertices v_1 and v_2. However, we give two alternative similarity measures induced by a structure processing SOM:

$$d_I(v_1, v_2) = d_R(I^p(v_1), I^p(v_2))$$

measures the distance of the winners for v_1 and v_2 in the lattice, and

$$d_D(v_1, v_2) = \alpha d_W(l(v_1), l(v_2)) +$$

$$\beta d_P(ch_1(v_1), ch_1(v_2)) + \beta d_P(ch_2(v_1), ch_2(v_2))$$

'unfolds' the comparison of internal representations of structures by one step. d_I is the similarity measure induced by the *lattice* on the tree structures. d_D is the similarity measure induced by the *contents of the neurons*. On one hand, d_I is accounted for because of the chosen lattice. d_D, on the other hand, is accounted for because of neighborhood cooperation during training. Thus, topology preservation refers to the fact that these two similarity measures, the measure d_I which focuses on similar locations of neurons, and d_D which focuses on similar contents of neurons, yield the same ordering of structures. If the induced similarities are comparable, topological ordering takes place, and the structures similar according to these metrics are arranged at similar locations on the map.

The similarity d_P given above compares internal representations and it is residing between the two extremes of d_I and d_D, depending on the context model. For the SOMSD d_P and d_I are identical, because the internal representation of context is the winner. For the context model given by RecSOM they are still similar, because the activity profile of the whole map focuses on the winner location. For MSOM d_P is similar to d_D, whereby the merge parameter γ is used to weight the distance of the actual entry and the context. If topology preservation takes place, the three similarity measures induce the same topological ordering and we can consider any of the induced similarity measures d_D, d_I, and d_P for structures to assess the underlying semantic visualized within the SOM.

We would like to add a comment on the question whether the topology preservation discussed above can be achieved for structure processing SOMs. In other words, the indices of neurons and the tuples containing the neurons' weights and contexts should arrange in the same order. It can be best seen for SOMSD, that this task might be problematic. For SOMSD, the contexts are given by the indices of the winners, and we have thus to arrange elements in the index set R in the same way as elements in $W \times R \times R$. This is obviously not possible, if the space R is densely covered, because the dimension of the latter product set is always larger than the dimension of the former one. This dilemma corresponds to the intuitively obvious problem that it is impossible to map the full potentially infinite-dimensional space of structures faithfully to the only finite-dimensional space of the interior representations. In applications, though, the set of structures is often only sparsely covered, and topology preservation can be still achieved then.

3.4 Noise tolerance

As a last mathematical issue on properties of recursive unsupervised maps we consider the noise tolerance of context models. In recursive computations there is always the risk of noise accumulation, and information within the structures cannot be securely maintained over a large number of recursive steps. The sensitivity of the models with respect to noise depends on the choice of *rep*. One obvious demand for *rep* is that it has got a limited codomain. Otherwise, the recursive distance computation might accumulate an arbitrarily large value during the processing of big structures. For this reason, the damping transformation $x \mapsto \exp(-x)$ has been added to RecSOM.

The noise tolerance of a chosen context model can be estimated by the term

$$d_R(\mathit{rep}(x_1, \ldots, x_n), \mathit{rep}(x_1 + \eta_1, \ldots, x_n + \eta_n)) =: (*) \ ,$$

η_i being independent and identically distributed noise with expected value zero and with finite variance. Assume that d_R is the squared Euclidean distance. If we had chosen *rep* as the identity we would obtain

$$(*) = N \cdot \eta_1^2 \ ,$$

with N being the number of neurons. The distance d_R is used in every recursive step of accumulation. If *rep* was the identity, noise would accumulate in the distance computation, because it would be multiplied by the factor N in each recursive step. For RecSOM we get

$$(*) = \sum_i \exp(-2 \cdot z_i) \cdot \eta_i^2 \ ,$$

where z_i is a value between x_i and $x_i + \eta_i$. Usually, a large number of components x_i will have large values corresponding to large distances of a considered structure from the neurons. For these coefficients the factor $\exp(-2 \cdot z_i)$ is small; thus, virtually all of these summand can be dropped. As a consequence, only the few summands corresponding to the winner and to similar neurons will effectively contribute to the noise. Hence, the noise does not scale with the dimensionality in this case, but it is suppressed because of the exponential weighting. This way also comparably high-dimensional vectors can be dealt with.

For SOMSD the situation is even better: the representation function *rep* has got a discrete output space. Therefore, it is error correcting if a clearly expressed activity profile of the map is present and if $(*)$ yields 0 in most cases. The noise tolerance is almost independent of the number of neurons for SOMSD and the computation is very robust. Since the representation function of MSOM is also defined via the winner, the same error correcting behavior takes place. However, noise can accumulate also in these settings, and the term $(*)$ can become non-vanishing. Since it is contained in a discrete alphabet, the noise influence is large then, making learning potentially unstable, as argued e.g. in (Voegtlin, 2002).

4 Experiments

Now we demonstrate the behavior of the models in two experiments. There is a double motivation for the experiments: on the one hand, we want to compare the models directly and demonstrate differences of their capacity and their applicability. On the other hand, we want to accompany our theoretical results with concrete pictures to demonstrate the obtained representation type and similarity measure.

4.1 Sequences

First, we report results for sequence processing recursive models as most models have been originally proposed for sequences. A direct comparison of the approaches is difficult because of the nature of unsupervised learning: no explicit objective is stated (such as the classification error for supervised models) which could be used for benchmarking. Therefore, we introduce a possible evaluation criterion of unsupervised models: the quantization error, i.e. the coverage of the data space by a given model. This measure quantifies for each neuron the deviation of elements of the data set in its receptive field relative to the mean specialization of the neuron. If the calculated value is small, the data space is optimally covered by the neurons according to the underlying data distribution. For sequence processing models, an interesting measure for the ability to capture the temporal structure of the data is the temporal quantization error for a number of steps back in the past.

Definition 10 *Assume that a sequence processing neural map and an input sequence* $(\ldots, \mathbf{s}^{t-1}, \mathbf{s}^t, \mathbf{s}^{t+1}, \ldots)$ *are given,* $\mathbf{s}^t$ *denoting the value for time step* t.[1] *For a neuron* i *we refer by* win_i *to the number of time steps in which neuron* i *becomes winner, i.e.* $win_i = |\{j \mid I(j) = i\}|$. *The mean activation of neuron* i *for time step* t *into the past is the vector*

$$\mathbf{a}_i(t) = \sum_{j:I(j)=i} \mathbf{s}^{j-t}/win_i \,.$$

This sum averages the past (t *steps backwards) over all events for which* i *becomes winner. The* temporal quantization error *of neuron* i *for time step* t *in the past is defined as*

$$e_i(t) = \left(\sum_{j:I(j)=i} \left\| \mathbf{s}^{j-t} - \mathbf{a}_i(t) \right\|^2 / win_i \right)^{1/2} .$$

If this quantization error is small, i.e. the average has only small deviation, then the activation of neuron i *as winner shows correlation with the past events at step* t.

[1] For simplicity, we assume that all data are given in one time series.

The temporal quantization error *for the map for time step t is defined as the average*

$$e(t) = \sum_{i=1}^{N} e_i(t)/N \ ,$$

N denoting the number of neurons.

It can be expected that the temporal quantization error for time step 0 is small: standard SOM minimizes this error. For increasing t, that is going back time, a small temporal quantization error of the map indicates that the neurons specialize not only on the current entry but also on the history, i.e. adaptation to the temporal context, which is only implicitly accounted for by the recursive processing, clearly takes place.

We extend an experiment proposed by Voegtlin (Voegtlin, 2002) for the RecSOM to all context models. The considered time series is the real-valued chaotic Mackey-Glass time series with a dynamic given by

$$\frac{dx}{d\tau} = bx(\tau) + \frac{ax(\tau - d)}{1 + x(\tau - d)^{10}} \ ,$$

with $a = 0.2$, $b = -0.1$, and $d = 17$. The presented values result from sampling at discrete time steps $\triangle t = 3$. Results are reported for different maps, each containing 100 neurons. The considered models are trained by presenting $1.5 \cdot 10^5$ values of the time series starting at random positions. The models are evaluated by their temporal quantization error for 30 steps in the past.

The considered models are the simple SOM with two-dimensional Euclidean lattice without recurrence, standard NG without recurrence, the recurrent SOM (RSOM) (as explained beforehand, this is based on the same local neighborhood model as TKM, but training is better adapted to the temporal dynamic with respect to optimal weight adaptation), RecSOM and SOMSD with two-dimensional Euclidean lattice, SOMSD with hyperbolic lattice (HSOMSD), and MSOM combined with the data optimal lattice of neural gas. To emphasize the lattice type chosen for MSOM, we refer to this combination by MNG. Results for SOM, RSOM, and RecSOM are taken from Voegtlin (Voegtlin, 2002). For regular lattice structures, neighborhood cooperation is given by a Gaussian function with initial neighborhood range 10 which is multiplicatively decreased during training. The results taken from Voegtlin refer to a rectangular 10×10 Euclidean lattice. The implementation for SOMSD and HSOMSD uses a general circular triangular meshing as described in (Strickert and Hammer, 2003b), which yields a Euclidean lattice if six neighbors per neuron are chosen, and it yields a hyperbolic lattice structure for seven neighbors per neuron.

Crucial parameters of the distance computation are the weighting factors α and β for the context and the current entry. Generally, context influence has to be small for time series to avoid instabilities of the model: for time series, structures have a large depth which might lead to instabilities and temporal accumulations of context changes. In the reported cases, the parameters are chosen as $1 - \alpha = \beta = 0.1$ for RSOM, and $\alpha = 2$ and $\beta = 0.06$ are chosen for RecSOM. For the other recursive models, only the ratio α/β is relevant because of the winner computation within the recursive context. For SOMSD and HSOMSD, β is initialized with 0 (no context influence) to allow the weights to converge, and the context influence is increased to 3% during training. For MSOM, two parameters have to be specified: the influence γ of context within encoding, which is set to 0.5 for a balanced mixing of current weight and context. The context influence within the recursive winner computation is adaptively determined during MSOM training to give optimal values, as introduced in (Strickert and Hammer, 2003a). This parameter is controlled by the entropy of the activity of the neurons: a small entropy of the activity of a neuron indicates a high specialization, thus the context influence might be increased to also achieve a specialization in time. If the entropy of a neuron increases, its specificity gets smaller and it is potentially ‘overloaded’, thus the context influence is decreased to allow a stabilization of the dynamic.

The training results are depicted in Fig. 1. As already mentioned, results indicated by $(*)$ are taken from Voegtlin (Voegtlin, 2002). The temporal quantization error of the map for 30 time steps in the past is shown. The most recent entry is on the left side of the graphs. SOM and NG do not take temporal context into account during training; correspondingly, the temporal quantization error is small for the current entry used for training, and it gets worse with increasing number of time steps into the past. The resulting curves reflect the inherent regularities of the time series: since the input is smooth, the first step backwards has still a small quantization error. Then oscillations according to the quasi-periodicity of the Mackey-Glass time series can be observed. The shape of the two graphs for NG and SOM is similar. Interestingly, NG allows a much better compression than SOM, although neither of the approaches accounts for a temporal context. Since this behavior is a bit surprising, we have reimplemented the standard SOM training scenario using a triangular mesh embedded in Euclidean space and parameters comparable to the NG setting. The quantization error of this triangular SOM was a bit smaller than the results reported by Voegtlin due to the denser mesh, but in principle, the difference to NG remained. This behavior can be explained by the fact that the lattice topology of NG is not restricted, and the data space can be optimally covered by this approach. RSOM yields anti-cyclic behavior compared to SOM and NG, whereby the quantization error is not better except for the recent past (up to four steps back). The current entry is not learned at all, which can be explained by the fact that the con-

Fig. 1 about here

text influence of the last two steps might disrupt the actual distance in such a way that the current entry becomes almost irrelevant. The value for three time steps back is optimum then. Afterwards, degradation can be observed. In comparison, RecSOM shows a lower error curve. Obviously, the current entry is better learned than for RSOM. In addition, the curve is only very weakly oscillating and context is clearly represented by this model. The quantization error increases for past values corresponding to accumulated noise and fading information. SOMSD also prevents oscillations and leads to an even smaller error than RecSOM although it relies on a compressed and much more efficient context model. In particular, past events are more reliably represented, which can be explained by the fact that computation noise does not accumulate in this model because the context computation relies only on the winner. Thereby, the immediate past represented by SOMSD is competitive to RecSOM. This region of the graph can be improved if more appropriate lattice structures instead of an Euclidean lattice are used. Obviously, both HSOMSD using a more complex hyperbolic lattice structure and MNG using the data optimal lattice structure of NG with the previous winner contents as context model show the best performance in this setting. Hence, this first experiment demonstrates that the models yield different results in practice, and it shows that more complex lattice structures and more complex context models allow a better coverage of the data space.

We make a further step and experimentally verify our findings about the internal representation of structures and induced metrics. Since the models rely on different lattice structures and on different internal representations, the maps cannot be compared directly with respect to this issue. To give a hint of the emerging internal representations, we exemplarily investigate the output of MNG and SOMSD. MNG uses the data optimal dynamic lattice corresponding to the nearest neighbor rela-

Fig. 2 about here.

tionships. Usually, the neuron arrangement cannot be visualized directly due to the high dimension of the input space; however, weights and contexts of neurons are elements of the vector space of the sequences entries which are scalar real numbers in our case. Thus, we can plot the contents of each neuron as a point in the two-dimensional plane given by the product space of weights and contexts. In Fig. 2, the contents of the neurons of a trained MNG-map are depicted. Although the contexts have been initialized by zeros, the figure illustrates how space for sequence representations is uniformly covered after training. The MNG space is used optimally in this example, and Hebbian learning converges to an appropriate encoding of structures corresponding to a fractal encoding, as has been stated by Theorem 2.

SOMSD uses a regular lattice structure for which a canonic visualization is obtained. Fig. 3 shows the mean activation of all neurons within a Euclidean circular triangular meshing for the past 30 time steps in combination with the bounds defined by the standard deviations. For five neurons (four neurons in the center of the map, and one neuron at the lower left corner), the activation is not well expressed and the variance is large. This observation corresponds to local contortions within the map, i.e. to regions where topologically faithful mapping is violated. For all other neurons, the variance of their activation is small at the immediate past, i.e. the left side of the graph, and it increases to the right, as already indicated by the global quantization error of the map in Fig. 1. The neurons are ordered in a topological way: the graphs which correspond to the activation of neighboring neurons differ only slightly, and a smooth phase shift of the represented curves can be observed for different paths in the map. Thus, topology preservation takes place with respect to the metric found in Theorem 3.

Fig. 3 about here

4.2 Tree structures

We add an experiment which demonstrates the applicability of recursive unsupervised models to tree structures and which illustrates the explicit metric given in Theorem 3. Our focus lies on the possibility of visualization and on the investigation whether the results are meaningful for different choices of the weighting parameters α and β. We restrict ourselves to SOMSD for the following reasons: since SOMSD relies on a regular lattice structure, visualization is easily possible, in contrast to the dynamic NG lattice. In comparison to RecSOM, SOMSD is much faster. RecSOM would need a storage capacity of $k \cdot N$ for each neuron, k being the fan-out of trees, and N the number of neurons of the map, compared to just $k \cdot 2$ for SOMSD with two-dimensional lattice. Since SOMs make ultimate use of distance calculations, the training time of SOMSD is faster by a factor in the order N compared to RecSOM. As discussed beforehand, TKM has only very limited capacity for tree structures, and a generalization of MSOM to tree structures is possible, but not very natural. Therefore, SOMSD is considered as the most appropriate model for dealing with tree structures. Note that extensive experiments for SOMSD for a different data set have already been reported in (Hagenbuchner, Sperduti, and Tsoi, 2003).

The data set used for the experiments contains company logos which are described as tree structures. The considered cases are a subset of the data provided in the article (Francesconi et al., 1997). We pick the 8 different logos shown in Fig. 4. Each logo is disturbed by rotation and blurring, as reported in (Francesconi et al., 1997), in order to simulate typical copying artifacts. This way we obtain 200 examples for each logo, which are randomly distributed in the training set and the test set. The pictures have been transformed into trees by contour analysis: a virtual surrounding

Fig. 4 about here.

Fig. 5 about here

Tab. 2 about here

rectangle forms the root of the tree and the contours within the image form the children. A contour becomes the child of another one if the latter encloses the previous one, as exemplarily depicted in Fig. 5. Each single contour is described by a 12-dimensional real vector which includes relevant features such as the contour area given by the number of pixels, the outer boundary, the number of enclosed pixels, the diameter, and characteristics related to the curvature of the contour. Additionally, the trees have been post-processed to ensure a maximum number of 7 children per vertex. For this reason, vertices which describe small contours are merged, and vertices with many children are split as described in (Francesconi et al., 1997). We use 100 examples per class for training, and 100 examples per class for testing. The mean, the maximum, and the minimum number of vertices in the trees of a specific class as well as the tree heights are reported in Tab. 2.

For training we use the standard Hebb algorithm and the training program described in (Hagenbuchner, Sperduti, and Tsoi, 2003). [2] We train a SOMSD with rectangular 2-dimensional lattice structure and 900 neurons. The initial neighborhood size is 15. 100 training examples corresponding to 12549 vertices are presented in 800 cycles. We fix the weighting term α to 1 and use different values for β. Since the maximum height of trees in the given data set is 6, β can be chosen much larger than in the case of sequences. Numerical instabilities do not occur because of the limited (and small) heights. The larger β, the more does the model focus on the structure of the trees and to characteristics within the entire tree compared to the characteristics of only the root vertex.

We report the classification accuracy on the test set by the comparison with the

Tab. 3 about here.

[2] We would like to thank Markus Hagenbuchner for providing the SOMSD-training program.

winner neurons labeled according to the training set (Tab. 3) for different values of β, with $\beta \in \{0.5, 1.5, 3.5, 5\}$. These neuron labels are determined as the majority class of root vertices for which the neuron becomes winner on the training set. Classes for the trees from the test set are obtained from the map by the class of the neuron which is winner for the root. Since not all neurons become winner for a root vertex of the training set this labeling is only partial, and the classes of several trees in the test set therefore remain unknown. As can be seen in Tab. 3, less than 15% of the trees are misclassified with this approach, where training itself takes place in a completely unsupervised fashion. Trees which are mapped to the same winner have a semantic similarity with respect to the given classes. In our case, the classification is best for $\beta = 3.5$ for which enough information is passed from the leaves to the root. The differences of the achieved classification accuracy, however, are not subject to much fluctuation in these experiments. We did not take much effort to optimize the learning parameters with respect to the classification result, because our focus is the investigation of unsupervised learning of tree structures in principle. Of course, the classification accuracy could be improved if additional supervised mechanisms were included, such as techniques from learning vector quantization for the optimization of classification boundaries (Kohonen, 1995). At this point, however, we are not interested in classification but in unsupervised data visualization. The classification accuracy thus serves only as a hint that semantically meaningful clusters arise in this setting.

In Fig. 6 we depict the winners of root vertices of the map depending on the class of the trees. The neurons are arranged in the rectangular lattice, determined by the neighborhood structure. The maps obtained for $\beta = 0.5$ and for $\beta = 5$ are not fundamentally different from the reported maps for $\beta = 1.5$ and $\beta = 3.5$. One can detect several clusters on the map which correspond to classes of logos: for

Fig. 6 about here.

$\beta = 1.5$, class C forms a cluster in the lower right corner, followed by (from right to left) class P, class Q, class B, and — partially overlapping — classes 9, and 3. Class 2 is scattered at the left side of the picture. From visual inspection of the logos one can say that the logos P and Q are similar (both just consist of words), logos 9 and M are similar (both consist of a circle which surrounds other simple and sparse contours), and B and C are similar (both consist of a non-intersecting arrangement of compact forms). On the map we find P and Q at neighboring locations (for $\beta = 1.5$ and 3.5), 9 and M at neighboring positions (for $\beta = 3.5$), and also B and C are not that far apart (for $\beta = 1.5$ and 3.5). The arrangement visualizes similarities between logos which one would expect from the raw pictures. Of course, this fact depends on the way in which the logos are represented as tree structures. Both characteristics the number of tree vertices and tree heights are reported in Tab. 2 which points out differences and similarities of the tree representations: classes B, C, P, and Q are represented by flat trees, whereas 3 is deeper. Classes Q, 2, and 3 consist of at least 10 neurons, whereas the maximum number of vertices for 9 and B is 6. The classes overlap partially and neurons are ordered in different arrangements for different choices of β according to the emphasis put on the context. The neurons are also arranged differently for different runs with the same parameter β, because topological faithful mapping is not possible which leads to erratic unfoldings of the map caused by suboptimal phase transitions during the training.

Because of variations in the structure of the single classes, the winner for trees of one class spread over the map. In Fig. 7 the winner locations of all vertices in the training set are shown depending on the depth of the vertices.[3] The number of vertices in the training set depending on its depth is 8788 for depth 1, 1823 for

Fig. 7 about here

[3] The depth of a vertex refers to the largest distance of the vertex from a leave, counting from 1, i.e. leaves have depth 1.

depth 2, 957 for depth 3, 548 for depth 4, 168 for depth 5, and 152 for depth 6. The majority of vertices has a depth of 1 and, correspondingly, a large area of the map represents leaves. These winner neurons form a compact cluster separated from the other neurons in the map. Also for depth 2 to 6 clusters can be observed which partially overlap. The clusters for depth 5 and depth 6 are small corresponding to the small number of vertices with this depth, and they are neighbors in both depicted maps. Winners for vertices of depths 2 to 4 are spread over the map, corresponding to different tree structures with this height which represent different logos. Note that there are several idle neurons in the maps, i.e. neurons which are never selected as winner for the training set. These neurons can be found particularly at cluster borders and between clusters which contain winners for different depth vertices. This effect can be explained by topological mismatches at these points of the map. Trees of the same structure with different contents can be faithfully mapped on a standard SOM. Trees with different structure are less similar (depending on the contents of the vertices, of course), and they are thus separated in the map. Idle nodes at the borders can be used to make the other winners more dissimilar, i.e. to separate the interior representations which otherwise would yield topological mismatches. SOMSD arranges the trees according to the structure and also according to the content as discussed for the explicit similarity measure derived in Theorem 3. By visual inspection, the resulting map clearly reflects the semantics given by the canonic similarity of the logos.

5 Conclusions

Unsupervised networks can be extended in a straight forward manner to recursive data structures such as sequences and tree structures, which has been demonstrated in this article. Concrete models implementing different notions of context have been realized in a unifying notation and investigated in detail. We have identified four main directions which correspond to models proposed in literature: a localized context like the one used for TKM, a reference to the whole information available in the map as taken for RecSOM, and a compression thereof, i.e. a reference to the previous winner location as realized by SOMSD, and a reference to the winner content as proposed for MSOM. As pointed out, the corresponding context models can be combined with different training algorithms and different lattice architectures.

We have formulated several criteria to compare the models with respect to practical applications:

(1) The complexity and storage capacity of the models: RecSOM is rather demanding, whereas the other three context models extend the requirements of standard SOM by only small additional vectors.
(2) The possibility to combine the models with different lattice structures: SOMSD relies on a fixed lattice of neurons for which a semantic meaning of the lattice structure is accounted for during training.
(3) The internal representation of structures: TKM and MSOM encode structures in weight space by fractal codes, whereas SOMSD and RecSOM extend the context representation to the space provided by the neurons in the lattice.
(4) The induced metric for Hebbian training: under idealized assumptions, all approaches induce metrics with Markovian flavor, i.e. they focus on the most recent entries of the considered structures and they include a distribution dependent fading memory over the whole structures.
(5) The representational capabilities for different types of structure: TKM and, more generally, localized approaches are fundamentally limited for tree struc-

tures. For all other models, every finite set of tree structures can be represented, at least in theory. In addition, we have shown that the representation ability of SOMSD transcends simple (finite memory) Markovian models in the sense that finite tree automata can be represented – although these might not be learned with Hebbian training.

(6) Noise tolerance: compression of noise is advisable and done by SOMSD, MSOM, and partially by RecSOM.

In addition, we have discussed the topic of topology preservation and we have directly compared the models in an experiment on time-series data. Clearly, different model capacities for the representation of temporal context can be observed. Based on this general overview and investigation of unsupervised structure learning, several directions of future research can be identified.

Here, we just focus on two interesting questions: Is it possible to design alternative learning algorithms which lead to better representations for the considered data? One could, for example, try to impose the structure of specified automata to a recursive map during training; thereby, appropriate regularization constraints must be applied because of formal reasons referring to the learnability of automata (Hammer and Steil, 2002). Such a regularization would also affect the arising metric structure. Alternatively, one could try to adapt the internal representation automatically to the given task for an optimal coverage of the data space. For a parameterized function *rep* (e.g. the mixture parameter γ of MSOM), the parameters could be set or adapted by global criteria such as the entropy.

Is it possible to extract information explicitly from the map and to find an appropriate representation of the regularities and of the temporal dependencies? As a first step into this direction, the U-matrix method for SOM has been extended to

SOMSD; with this approach simple finite memory Markov models have been successfully extracted from trained maps in the article (Strickert, Hammer, Blohm, 2004). The best representation of the temporal dependencies in a trained map could be further investigated as well as browsing strategies for the temporal context stored in a map.

These questions constitute the practical counterparts to the mathematical foundations of recursive unsupervised models. The demonstrations presented in this article concerning the applicability of the models are a first promising step towards the implementation of an efficient and powerful toolbox for the visualization and the analysis of complex non-vectorial data structures.

Acknowledgment

This work has been partially supported by MIUR grant 2002093941_004. We would like to thank two anonymous reviewers for valuable and detailed comments on an earlier version of this manuscript.

References

Baldi, P., Brunak, S., Frasconi, P., Pollastri, G., and Soda, G. (1999). Exploiting the past and the future in protein secondary structure prediction. *Bioinformatics*, 15(11).

Bianucci, A .M., Micheli, A., Sperduti, A., and Starita, A. (2000). Application of cascade correlation networks for structures to chemistry. *Journ. of Appl. Int.*, 12:117–146.

Barreto, G. and Araújo, A. (2001). Time in self-organizing maps: An overview of models. *Int. Journ. of Computer Research*, 10(2):139–179, 2001.

Barreto, G., Araújo, A., and Kremer, S.C. (2003). A Taxonomy for Spatiotemporal Connectionist Networks Revisited: The Unsupervised Case. *Neural Computation*, 15(6):1255-1320.

Bauer, H.-U. and Pawelzik, K.R. (1992). Quantifying the Neighborhood Preservation of Self-Organizing Feature Maps. *IEEE Transactions on Neural Networks*, 3(4):570–579.

Bauer, H.-U. and Villmann, T. (1997). Growing a Hypercubical Output Space in a Self–Organizing Feature Map. *IEEE Transactions on Neural Networks*, 8(2):218–226.

Bezdek, J.C., Hathaway, R.H., Sabin, M.J., and Tucker, W.T. (1987). Convergence theory for fuzzy c-means: counterexamples and repairs. *IEEE Transactions on Systems, Man, and Cybernetics* 17:873-877.

Carrasco, R.C. and Forcada, M.L. (2001). Simple strategies to encode tree automata in sigmoid recursive neural networks. *IEEE Transactions on Knowledge and Data Engineering* 13(2): 148-156.

Chappell, G. and Taylor, J. (1993). The temporal Kohonen map. *Neural Networks* 6:441-445.

Claussen, J. C. and Villmann, T. (2003). Magnification Control in Winner Relaxing Neural Gas. In M.Verleysen (ed.), *European Symposium on Artificial Neural Networks*, pages 93-98, D-side publications.

Costa, F., Frasconi, P., Lombardo, V., and Soda, G. (2003). Towards incremental parsing of natural language using recursive neural networks. *Applied Intelligence* 19(1-2).

Cottrell, M., Fort, J.C., and Pagès, G. (1994). Two or three things that we know about the Kohonen algorithm. In M.Verleysen (ed.), *European Sym-*

posium on Artificial Neural Networks, pages 235-244, D-side publications.

Diligenti, M., Frasconi, P., and Gori, M. (2003). Hidden tree Markov models for document image classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(4):519-523.

De Mauro, C., Diligenti, M., Gori, M., and Maggini, M. (2003). Similarity learning for graph-based image representations. *Pattern Recognition Letters* 24(8):1115-1122.

Erwin, E., Obermayer, K., and Schulten, K. (1992). Self-organizing maps, convergence properties, and energy functions. *Biological Cybernetics* 67(1):47–55.

Euliano, N.R. and Principe, J.C. (1999). A spatiotemporal memory based on SOMs with activity diffusion. In E.Oja and S.Kaski (eds.), *Kohonen Maps*, Elsevier.

Farkas, I. and Miikkulainen, R. (1999). Modeling the self-organization of directional selectivity in the primary visual cortex. In *Proceedings of the International Conference on Artificial Neural Networks*, pages 251-256, Springer.

Francesconi, E., Frasconi, P., Gori, M., Marinai, S., Sheng, J.Q., Soda, G., and Sperduti, A. (1997). Logo recognition by recursive neural networks, Second International Workshop on Graphics Recognition, GREC'97. Eds. Kasturi, R. and K. Tombre, LNCS, pp.104-117, Springer-Verlag.

Frasconi, P., Gori, M., Küchler, A., and Sperduti, A. (2001). A field guide to dynamical recurrent networks, in: J.F.Kolen, S.C.Kremer(eds.), *From Sequences to Data Structures: Theory and Applications*, pages 351-374, IEEE.

Frasconi, P., Gori, M., and Sperduti, A. (1998). A general framework for adaptive processing of data structures. *IEEE TNN*, 9(5):768–786.

Gärtner, T. (2003). A survey of kernels for structured data. *SIGKDD explorations*.

Gécseg, F. and Steinby, M. (1984). *Tree automata.* Akademiasi Kiado, Budapest.

Günter, S. and Buhnke, H. (2001). Validation indices for graph clustering. In: J.-M. Jolion, W.G. Kropatsch, and M. Vento (eds.), *Proceedings of the third IAPR-TC15 Workshop on Graph-based Representations in Pattern Recognition*, pages 229-238, Ischia, Italy.

Gori, M., Küchler, A., and Sperduti, A. (1999). On the implementation of frontier-to-root tree automata in recursive neural networks. *IEEE Transactions on Neural Networks*, 10.

Hagenbuchner, M., Tsoi, A.C., and Sperduti, A. (2001). A supervised self-organizing map for structured data. In N.Allison, H.Yin, L.Allinson, and J.Slack (eds.), *Advances in Self-Organizing Maps*, pages 21-28, Springer.

Hagenbuchner, M., Sperduti, A., and Tsoi, A.C. (2003). A Self-Organizing Map for Adaptive Processing of Structured Data. *IEEE Transactions on Neural Networks* 14:191-505.

Hammer, B. (2002). Recurrent networks for structured data - a unifying approach and its properties. *Cognitive Systems Research* 3(2), 145-165.

Hammer, B. (2000). *Learning with Recurrent Neural Networks*. LNCIS 254, Springer.

Hammer, B. and Jain, B.J. (2004). Neural methods for non-standard data. In M.Verleysen (ed.), *European Symposium on Artificial Neural Networks*, pages 281-292, D-side publications.

Hammer, B., Micheli, A., and Sperduti, A. (2002). A general framework for unsupervised processing of structured data. In M.Verleysen (ed.), *European Symposium on Artificial Neural Networks*, pages 389-394, D-side publications.

Hammer, B., Micheli, A., Sperduti, A., and Strickert, M. (2004). A general framework for unsupervised processing of structured data. *Neurocomputing* 57, 3-35.

Hammer, B. and Steil, J. (2002). Perspectives on learning with recurrent networks. In: M.Verleysen (ed.), *European Symposium on Artificial Neural Networks'2002*, D-side publications, 357-368.

Hammer, B. and Tino, P. (2003). Recurrent neural networks with small weights implement definite memory machines. *Neural Computation* 15(8): 1897-1929.

Hammer, B. and Villmann, T. (2002). Generalized relevance learning vector quantization. *Neural Networks* 15, 1059-1068.

Heskes, T. (2001). Self-organizing maps, vector quantization, and mixture modeling. *IEEE Transactions on Neural Networks*, 12:1299-1305.

Hoekstra, A. and Drossaers, M.F.J. (1993). An extended Kohonen feature map for sentence recognition. In S.Gielen and B.Kappen (eds.), *Proc. ICANN'93. International Conference on Artificial Neural Networks*, pages 404–407, Springer.

James, D.L. and Miikkulainen, R. (1995). SARDNET: a self-organizing feature map for sequences. In G.Tesauro, D.Touretzky, and T.Leen (eds.), *Advances in Neural Information Processing Systems 7*, pages 577-584, MIT Press.

Kangas, J. (1990). Time-delayed self-organizing maps. In: *Proc. IEEE/INNS*

Int. Joint Conj. on Neural Networks 1990, 2, pages 331-336.

Kaski, S. (2001). Bankruptcy analysis with self-organizing maps in learning metrics. *IEEE Transactions on Neural Networks*, 12:936-947.

Kaski, S., Kangas, J., and Kohonen, T. (1998). Bibliography of self-organizing maps, papers: 1981-1997. *Neural Computing Surveys*. 1, 102-350.

Kilian, J. and Siegelmann, H.T. (1996). The dynamic universality of sigmoidal neural networks. *Information and Computation*, 128.

Kohonen, T. and Sommervuo, P. (2002). How to make large self-organizing maps for nonvectorial data. Neural Networks 15(8-9):945-952.

Kohonen, T. (1997). *Self-organizing maps*. Springer.

Kohonen, T. (1995). Learning vector quantization. In M. Arbib, editor, *The Handbook of Brain Theory and Neural Networks*, pages 537–540. MIT Press.

Kohonen, T., Kaski, S., Lagus, K., and Honkela, T. (1996). Very large two-level SOM for the browsing of newsgroups. In C. von der Malsburg, W. von Seelen, J. C. Vorbrüggen, and B. Sendhoff (eds.), *Proceedings of ICANN96*, pages 269–274, Springer.

Koskela, T., Varsta, M., Heikkonen, J., and Kaski, K. (1998). Recurrent SOM with local linear models in time series prediction. In M.Verleysen (ed.), *6th European Symposium on Artificial Neural Networks*, pages 167–172, De facto.

Koskela, T., Varsta, M., Heikkonen, J., and Kaski, K. (1998). Time Series Prediction using Recurrent SOM with Local Linear Models. *Int. J. of Knowledge-Based Intelligent Engineering Systems* 2(1): 60-68, 1998.

Linde, Y., Buzo, A., and Gray, R. (1980). An algorithm for vector quantizer design. *IEEE Transactions on Communications* 28:84-95.

Martinetz, T., Berkovich, S., and Schulten, K. (1993). "Neural-gas" network for vector quantization and its application to time-series prediction. *IEEE-Transactions on Neural Networks* 4(4): 558-569.

Martinetz, T. and Schulten, K. (1993). Topology representing networks. *Neural Networks* 7(3):507-522.

Omlin, C.W., and Giles, C.L. (1996). Constructing deterministic finite-state automata in recurrent neural networks. *Journal of the ACM*, 43(6):937-972.

Kremer, S.C. (2001). Spatio-temporal connectionist networks: A taxonomy and review. *Neural Comp.*, 13(2):249–306.

Ontrup, J. and Ritter, H. (2001). Text categorization and semantic browsing with self-organizing maps on non-euclidean spaces. In: L.De Raedt and A.Siebes (eds.), *Proceedings of PKDD-01, 5th European Conference on Principles and Practice of Knowledge Discovery in Databases*, pp. 338–349, Springer, LNAI 2168.

Pollastri, G., Baldi, P., Vullo, A., and Frasconi, P. (2002). Prediction of protein topologies using GIOHMMs and GRNNs. In: *Advances of Neural Information Processing Systems 2002*.

Ritter, H. (1993). Parametrized self-organizing maps. In: S. Gielen, B. Kappen (Eds.), *Proc. ICANN'93 International Conference on Artificial Neural Networks*, Springer, London, UK, pp. 568-575.

Ritter, H. (1999). Self-organizing maps in non-euclidean spaces. In E. Oja and S. Kaski, editors, *Kohonen Maps*, pages 97–108. Springer.

Ritter, H., Martinetz, T., and Schulten, K. (1992). *Neural Computation and Self-Organizing Maps*. AddisonWesley, Reading.

Ritter, H. and Schulten, K. (1986). On the stationary state of Kohonen's self-organizing sensory mapping. *Biological Cybernetics*, 54(1):99-106.

Sinkkonen, J. and Kaski, S. (2002). Clustering based on conditional distribution in an auxiliary space. *Neural Computation*, 14:217-239.

Sperduti, A. (2001). Neural networks for adaptive processing of structured data. In: G.Dorffner, H.Bischof, K.Hornik (eds.), *ICANN'2001*, pages 5-12, Springer.

Sperduti, A. and Starita, A. (1997). Supervised Neural Networks for the Classification of Structures. *IEEE Transactions on Neural Networks*, 8(3):714-735.

Strickert, M. and Hammer, B. (2003). Neural Gas for Sequences, *WSOM'03*, 53-57.

Strickert, M. and Hammer, B. (2003). Unsupervised recursive sequence processing, in: M.Verleysen (ed.), *European Symposium on Artificial Neural Networks'2003*, D-side publications, 27-32.

Strickert, M., Hammer, B., and Blohm, S. (2004). Unsupervised recursive sequence processing. To appear in Neurocomputing.

Sturt, P., Costa, F., Lombardo, V., and Frasconi, P. (2003), Learning first-pass structural attachment preferences with dynamic grammars and recursive neural networks. *Cognition*, 88(2):133-169, 2003.

Varsta, M., Heikkonen, J., Lampinen, J., and Milán, J.del R. (2001). Temporal Kohonen map and recurrent self-organizing map: analytical and experimental comparison. *Neural Processing Letters*, 13(3):237-251.

Vesanto, J. (1997). Using the SOM and local models in time-series prediction. In: *Proc. Workshop on Self-Organizing Maps 1997*, pages 209-214.

Villmann, T., Der, R., Herrmann, M., and Martinetz, T. (1997). Topology

Preservation in Self–Organizing Feature Maps: Exact Definition and Measurement. *IEEE Transactions on Neural Networks*, 8(2):256–266.

Villmann, T., Merenyi, E., and Hammer, B. (2003). Neural maps in remote sensing image analysis. *Neural Networks* 16(3-4),389-403.

Voegtlin, T. (2000). Context quantization and contextual self-organizing maps. In: *Proc. Int. Joint Conf. on Neural Networks*, vol.5, pages 20-25.

Voegtlin, T. (2002). Recursive self-organizing maps. *Neural Networks* 15(8-9):979-992.

Voegtlin, T. and Dominey, P.F. (2001). Recursive self-organizing maps. In N.Allison, H.Yin, L.Allinson, and J.Slack (eds.), *Advances in Self-Organizing Maps*, pages 210-215, Springer.

Vullo, A. and Frasconi, P. (2003). A recursive connectionist approach for predicting disulfide connectivity in proteins. *Proceedings of the Eighteenth Annual ACM Symposium on Applied Computing (SAC 2003)*, Melbourne, FL.

Yao, Y., Marcialis, G.L., Pontil, M., Frasconi, P., and Roli, F. (2003). Combining flat and structured representations for fingerprint classification with recursive neural networks and support vector machines. *Pattern Recognition*, 36(2): 397-406.

List of Tables

List of Figures

	TKM	RecSOM	SOMSD	MSOM
regular lattice	x	x	x	x
no lattice	x	x		x
neural gas lattice	x	x		x
hyperbolic lattice	x	x	x	x

Table 1

Possibilities to combine the different context models with different lattice structures.

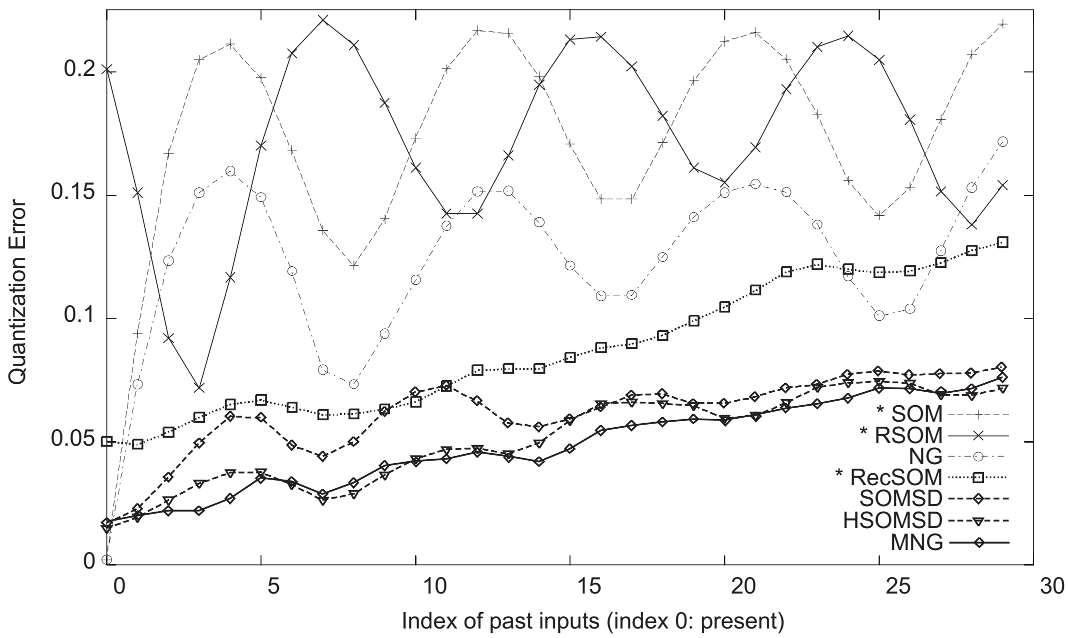

Fig. 1. Temporal quantization error of different unsupervised models for 30 steps into the past. SOM and NG (without context integration) can serve as a baseline since they represent the inherent pseudo-periodic structure of the time series. Results indicated by $*$ are taken from Voegtlin.

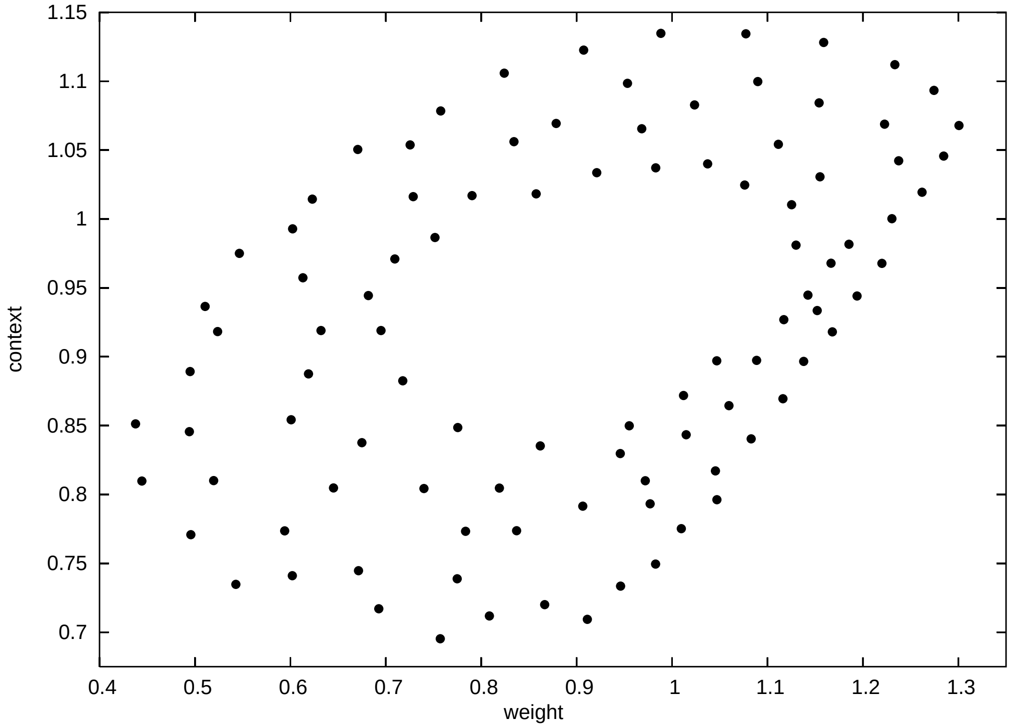

Fig. 2. Plot of weights and contexts of each neuron of MNG trained for the Mackey-Glass time series.

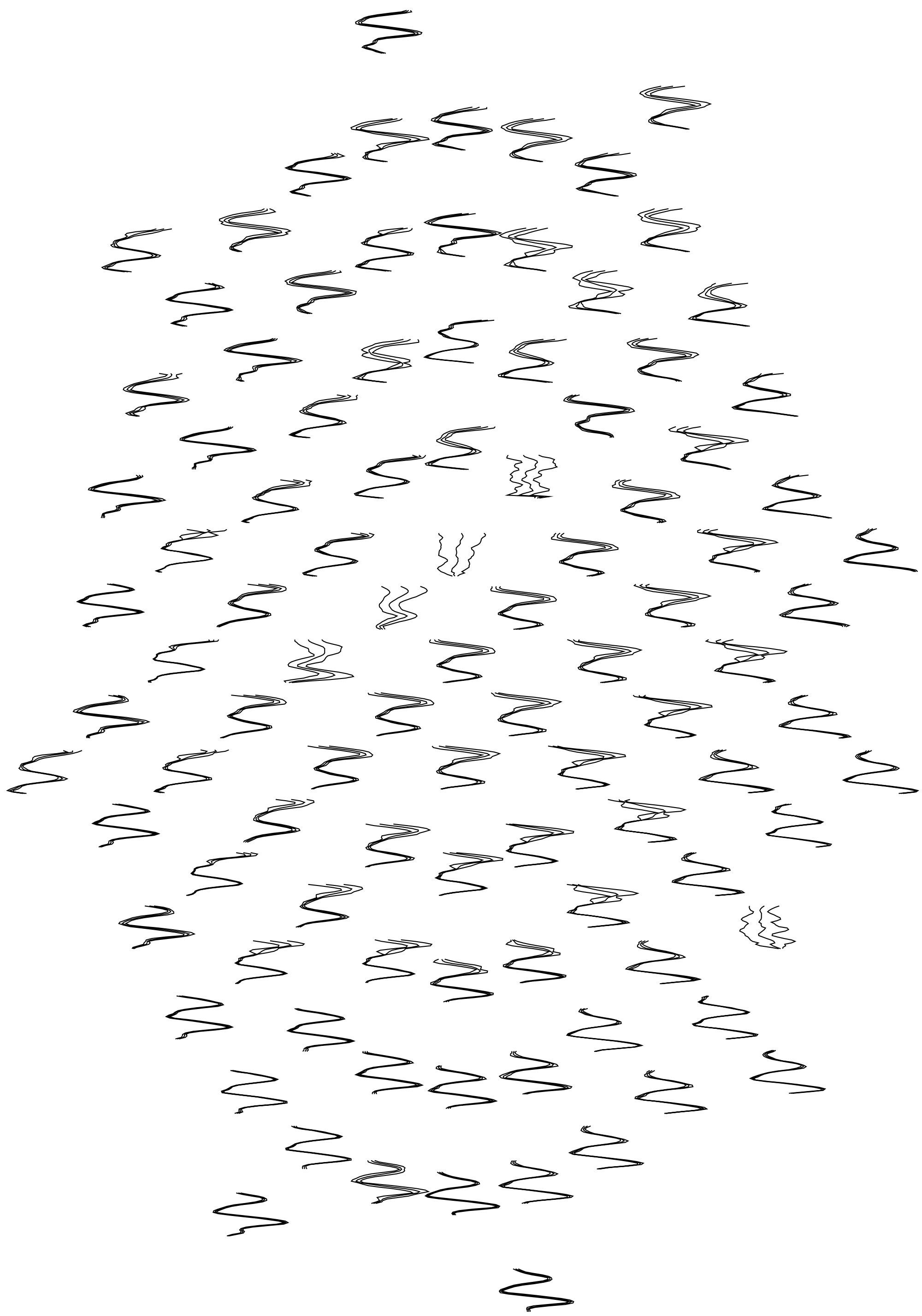

Fig. 3. Activation of the neurons of circular SOMSD with triangular meshing for the past 30 time steps. From left to right, the mean activation of time step t backward in time, $t \in \{0, \ldots, 29\}$ of time series for which the neurons become winner is depicted. In addition, an upper boundary and lower boundary are shown corresponding to the standard deviation of the value at time point t, $t \in \{0, \ldots, 29\}$, taken over all time series for which the neuron becomes winner. The map is given in landscape mode. The neurons are arranged on a triangular meshing whereby each neuron has 6 neighbors.

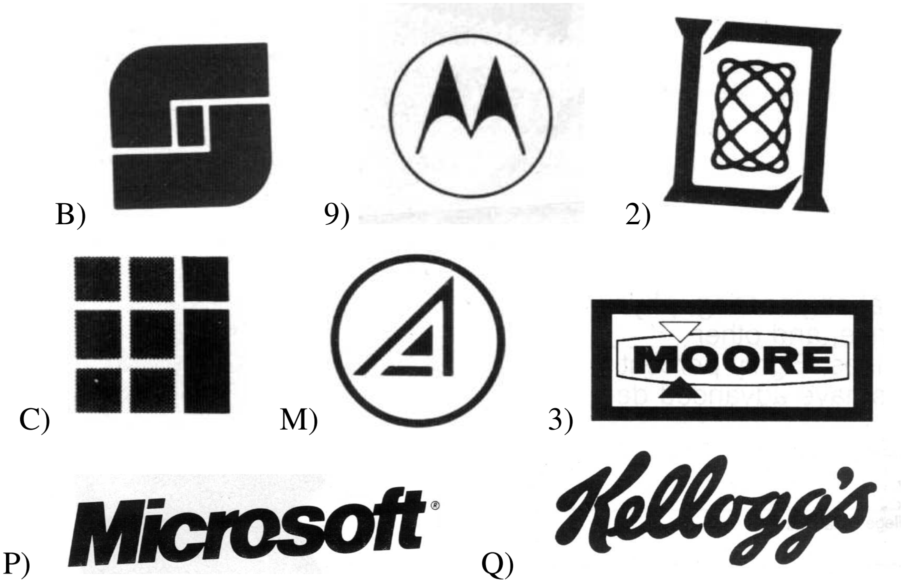

Fig. 4. Pictures of logos used to train the tree processing SOMSD.

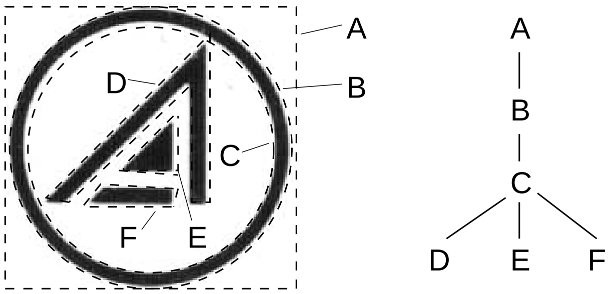

Fig. 5. Example of how a logo is represented as a tree structure based on contours.

class	9	M	B	C	P	Q	2	3
height								
min	2	2	1	1	2	2	2	4
max	4	4	2	2	2	2	4	6
mean	2.8	3	1.1	1	2	2	2.7	5.7
number of vertices								
min	3	3	2	6	8	10	18	10
max	6	8	6	11	12	15	23	16
mean	4.2	6	4	8.5	10	12.6	20.8	13.2

Table 2

Number of vertices and heights of the trees in the training set.

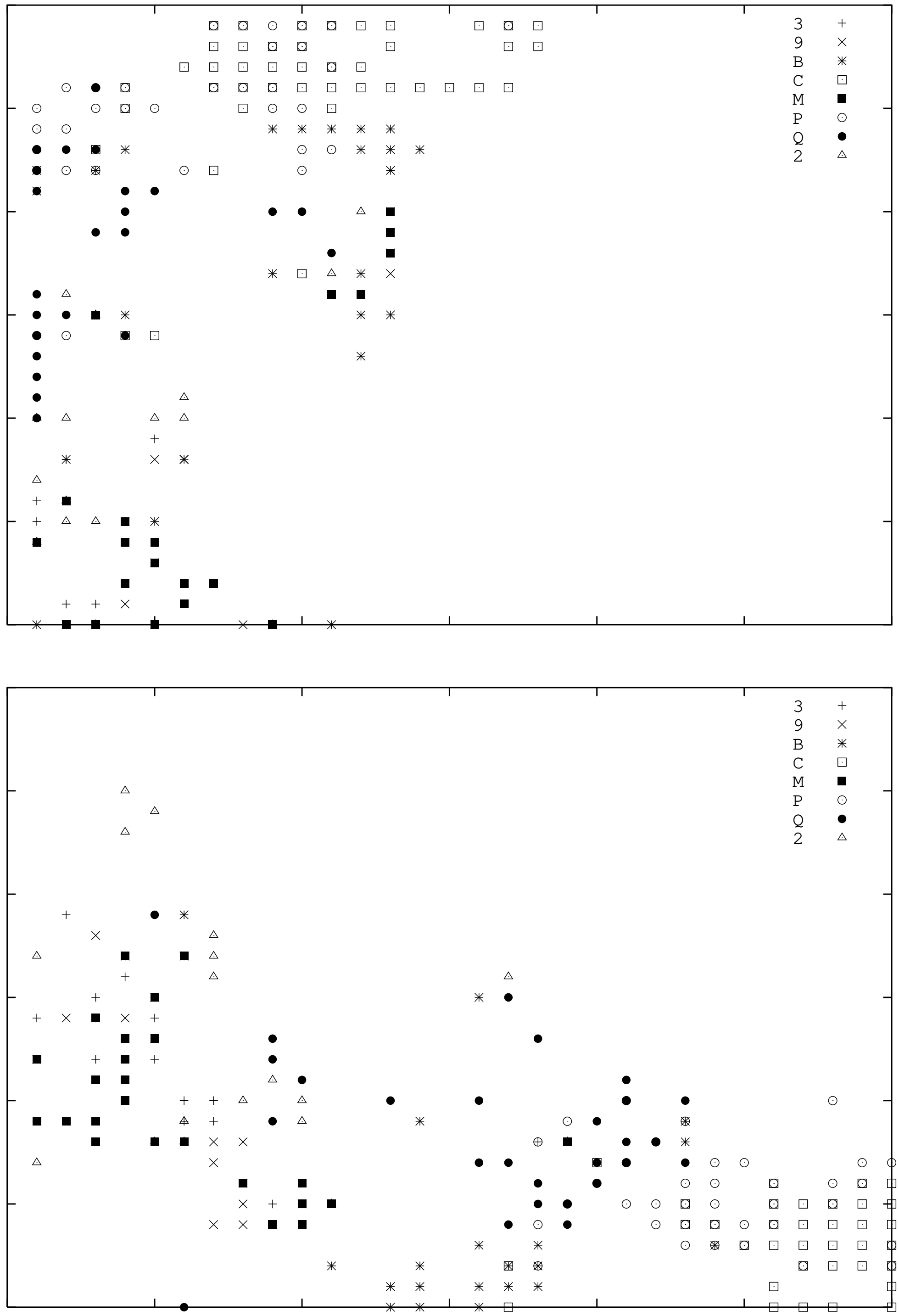

Fig. 6. Mapping of root vertices of the trees in the training set to the map arranged according to the classes for $\beta = 1.5$ (top) and $\beta = 3.5$ (bottom).

β	correct	wrong	unknown
0.5	82.91%	15.77%	1.33%
1.5	86.27%	10.89%	2.83%
3.5	86.36%	9.48%	4.16%
5	86.27%	10.27%	3.45%

Table 3

Classification results on the test set (for the root vertices) according to the label of the winner neuron. Labels are attached to winner neurons obtained by a majority vote from the training set. Unknown classification might result for neurons which do not become winner on the training set for a root vertex.

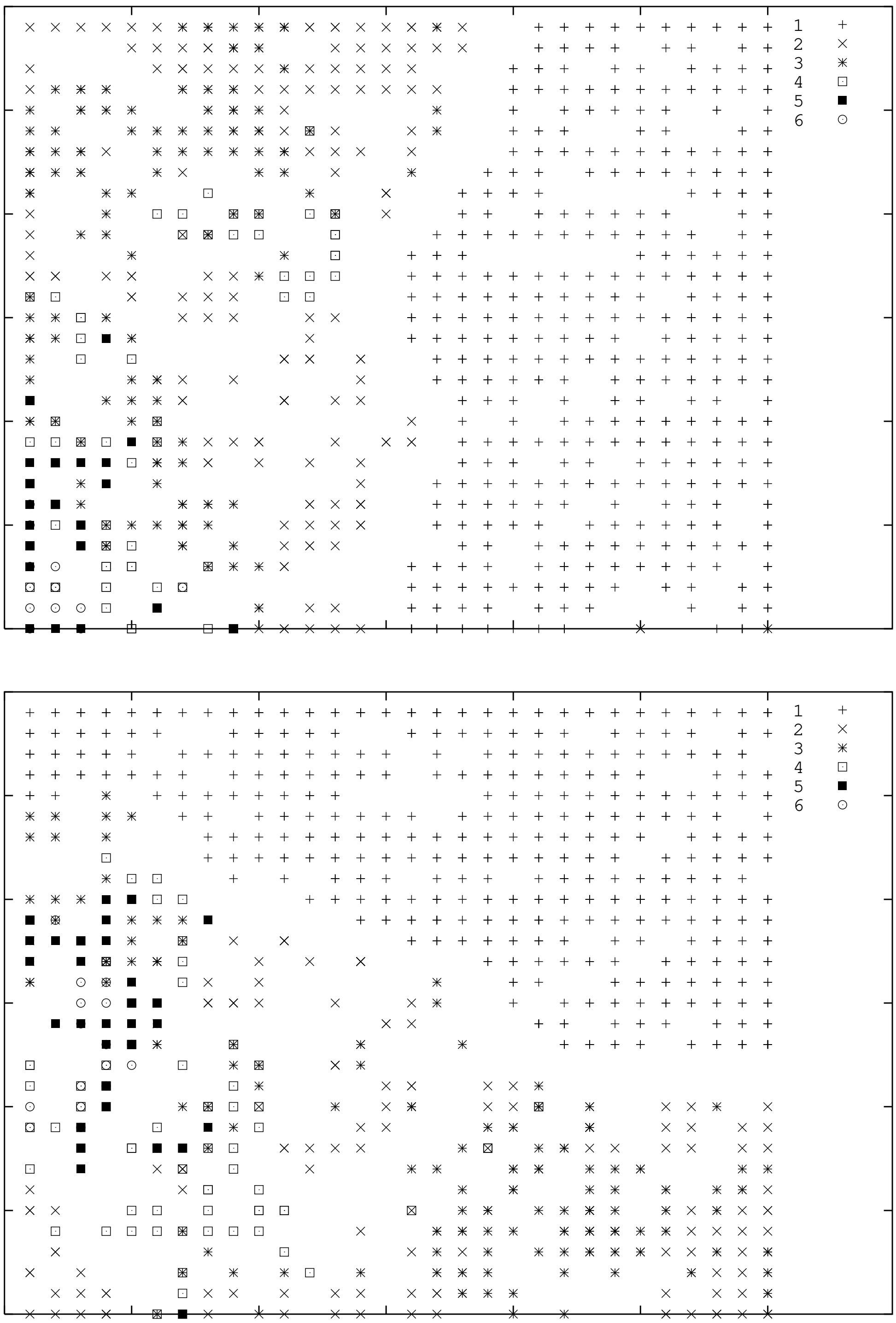

Fig. 7. Mapping of vertices of the trees in the training set to the map arranged according to the depth of the vertices for $\beta = 1.5$ (top) and $\beta = 3.5$ (bottom).