UNIVERSITÀ DI PISA

DIPARTIMENTO DI INFORMATICA

TECHNICAL REPORT: TR-03-04

A general framework for self-organizing structure processing neural networks

Barbara Hammer, Alessio Micheli, and Alessandro Sperduti

February 14, 2003

ADDRESS: Via Buonarroti 2, 56127 Pisa, Italy. TEL: +39 050 2212700 FAX: +39 050 2212726

A general framework for self-organizing structure processing neural networks *

Barbara Hammer[†], Alessio Micheli[‡], and Alessandro Sperduti[§]

February 14, 2003

Abstract

Self-organization constitutes an important paradigm in machine learning with successful applications e.g. for data- and web-mining. However, so far most approaches have been proposed for data contained in a fixed and finite dimensional vector space. We will focus on extensions for more general data structures like sequences and tree structures in this article. Various extensions of the standard self-organizing map (SOM) to sequences or tree structures have been proposed in the literature: the temporal Kohonen map, the recursive SOM, and SOM for structured data (SOMSD), for example. These methods enhance the standard SOM by recursive connections. We define in this article a general recursive dynamic which enables the recursive processing of complex data structures based on recursively computed internal representations of the respective context. The above mechanisms of SOMs for structures are special cases of the proposed general dynamic, furthermore, the dynamic covers the supervised case of recurrent and recursive networks, too. The general framework offers a uniform notation for training mechanisms such as Hebbian learning and the transfer of alternatives such as vector quantization or the neural gas algorithm to structure processing networks. The formal definition of the recursive dynamic for structure processing unsupervised networks allows the transfer of theoretical issues from the SOM literature to the structure processing case. One can formulate general cost functions corresponding to vector quantization, neural gas, and a modification of SOM for the case of structures. The cost functions can be compared to Hebbian learning which can be interpreted as an approximation of a stochastic gradient descent. We derive as an alternative the exact gradients for general cost functions which can be computed in several ways similar to well-known learning algorithms for supervised recurrent and recursive networks. We afterwards discuss general design issues which should be taken into account when choosing internal representations and which limit the applicability of specific choices in concrete situations: We investigate the necessary cardinality of the set of internal representations, the tolerance with respect to noise, and the necessary of globality of processing. Finally, we discuss the issue of topology preservation of structure processing self-organizing maps and derive an explicit metric for a specific situation for SOMSD.

1 Introduction

Neural networks constitute a particularly successful approach in machine learning which allows to learn an unknown regularity given a set of training examples. Learning tasks neural networks can deal with might be supervised or unsupervised; i.e. outputs or classes for the data points might be available and the network has to learn how to assign given input data appropriately to the correct class in the supervised case or, in the unsupervised case, no prior information about a valid separation into classes is known and the network has to extract useful information and reasonable classes from data by itself. Naturally, the latter task is more difficult because the notion of 'useful information' depends on the context. Results are often hard to evaluate automatically and they are to be validated by experts in the respective field. Nevertheless,

*The research has been done while the first author was visiting the University of Pisa. She would like to thank the groups of Pisa and Siena for their warm hospitality during her stay.

†Research group LNM, Department of Mathematics/Computer Science, University of Osnabrück, Germany. *E-mail: hammer@informatik.uni-osnabrueck.de*

‡Dipartimento di Informatica, Università di Pisa, Pisa, Italia. *E-mail: micheli@di.unipi.it*

§Dipartimento di Matematica Pura ed Applicata, Università di Padova, Padova, Italia. *E-mail: sperduti@math.unipd.it*

the task of unsupervised information processing occurs in many areas of applications were explicit teacher information is not yet available such as data- and Web-mining, bioinformatics, or text categorization, to name just a few topics. Most approaches for supervised or unsupervised learning in the context of neural networks deal with finite dimensional vectors as inputs. Data are given by sequences, trees, or graphs in interesting areas of application such as time-series prediction, speech processing, bioinformatics, chemistry, or theorem proving. Hence data are to be preprocessed appropriately in these cases such that important features are integrated in a simple vector representation of data. Preprocessing is domain dependent and time consuming. Moreover, loss of information is often inevitable. Hence effort has been done to derive neural methods which can deal with structural data directly.

In the supervised scenario, various successful approaches have been developed: Supervised recurrent neural networks constitute a well established approach for modeling sequential data e.g. in language processing or time series prediction [14, 15]. They can naturally be generalized to so-called recursive networks such that more complex data structures, tree structures and directed acyclic graphs, can be dealt with [12, 42]. Since symbolic terms possess a representation as a tree, this generalization has successfully been applied in various areas where symbolic or hybrid data structures arise such as theorem proving, chemistry, image processing, or natural language processing [1, 2, 8, 11]. The training method for recursive networks is a straightforward generalization of standard backpropagation through time[41, 42]. Moreover, important theoretical investigations from the field of feedforward and recurrent neural networks have been transferred to recursive networks [11, 13, 19].

Unsupervised learning as alternative important paradigm for neural networks has been successfully applied in data mining and visualization (see [29]). Since additional structural information is often available in possible applications of self-organizing maps (SOMs), a transfer of standard unsupervised learning methods to sequences and more complex tree structures would be valuable. Several approaches enlarge SOM to sequences: SOM constitutes a metric based approach. Hence it can be applied directly to structural data if data are represented such that an appropriate metric for the respective structural data is defined and a notion of how to adapt within the space given by structural data can be found. This has been proposed e.g. in [16, 26, 43]. Various approaches alternatively enlarge SOM by recurrent dynamics such as leaky integrators or more general recurrent connections which allow the recursive processing of sequences. Examples are the temporal Kohonen map (TKM) [7], the recursive SOM (RecSOM) [45, 46, 47], or the approaches proposed in [9, 24, 25, 31]. The SOM for structured data (SOMSD) [17, 18, 41] constitutes a recursive mechanism capable of processing tree structured data and thus also sequences in an unsupervised way. Alternative models for unsupervised time series processing use for example hierarchical network architectures. An overview of important models can be found e.g. in [3].

We will here focus on models based on recursive dynamics for structural data and we will derive a generic formulation of recursive self-organizing maps. We will propose a general framework which transfers the idea of recursive processing of complex data for supervised recurrent and recursive networks to the unsupervised scenario. This general framework covers TKM, RecSOM, SOMSD, and the standard SOM as has already been shown in a short version in [20]. The methods share the basic recursive dynamic but they differ in the way in which structures are internally represented by the neural map. An appropriate choice of the form of internal representations allows to recover the respective mechanism. Moreover, the dynamic of supervised recurrent and recursive networks can be integrated in the general framework as well. The approaches reported in [9, 24, 31] can be simulated with slight variations of parts of the framework. Hence we obtain a uniform formulation which allows a straightforward investigation of possible learning algorithms and theoretical properties of several important approaches proposed in the literature for SOMs with recurrence.

The reported models are commonly trained with Hebbian learning. The general formulation allows to formalize Hebbian learning in a uniform manner and to immediately transfer alternatives like the neural gas algorithm [33] or vector quantization to the approaches. For standard vector-based SOM and alternatives like neural gas, Hebbian Learning can be (approximately) interpreted as a stochastic gradient descent method on an appropriate error function [22, 33]. One can uniformly formulate analogous cost functions for the general framework for structural self-organizing maps and investigate the connection to Hebbian learning. It turns out that Hebbian learning can be interpreted as an approximation of a gradient mechanism where contributions of substructures are discarded. The exact gradient mechanism includes recurrent neural network training as a special case and explicit formula comparable to backpropagation through time or real time recurrent learning [36] can be derived for the unsupervised case. This gives some hints to understand the dynamics of unsupervised network training.

Another benefit of the general formulation lies in the possibility of comparing the above methods with respect to their capacity. Note that the above methods have been applied to monitor time series, language, or image data. A first evaluation criterion of structure processing SOMs, the depth of a quantizer, has been proposed in [46]. However, a clear and uniform measure of the success of unsupervised processing of structured data which is adequate for all reasonable situations is not yet available. Hence the success of the approaches is often measured on supervised classification tasks or, alternatively, experts rate the visualization obtained from the neural maps. The reported results for the above concrete models proposed in the literature seem very promising, however, a general theoretical framework for a deeper insight is helpful. This allows us to formulate general demands like the representation capacity and test these properties for the specific choices of representation. As an example, we will investigate the error tolerance, the representation capability, and the notion of topology and topology preservation within the general dynamics. It is possible to prove that specific choices of the internal representation of context, the part which distinguishes the several specific approaches, are more appropriate for structure processing than others. These investigations are first steps towards a general theory of unsupervised recurrent and recursive networks.

We will now define the general framework formally and show that SOMs with recursive dynamics as proposed in the literature can be recovered as special cases of the general framework. We show how Hebbian learning can be formulated within this approach. We propose alternative training methods based on energy functions, such that popular methods in the field of unsupervised learning can be directly transferred to this general framework and supervised and unsupervised training methods can be related. Afterwards we formulate general demands on the internal representation of context and investigate the noise tolerance, the representation capacity, and the notion of topology preservation. We finally consider the induced metric on the structural domain and derive an explicit formula for a special case of SOMSD.

2 Structure processing self-organizing maps

We first clarify a notation: the term 'self-organizing map' used in the literature refers to both, the paradigm of a neural system which learns in a self-organizing fashion, and the specific and very successful self-organizing map which has been proposed by Kohonen [29]. In order to distinguish between these two meanings, we refer to the specific architecture proposed by Kohonen by the shorthand notation SOM. If we speak of self-organization, the general paradigm is referred to. The SOM as proposed by Kohonen is a biologically motivated neural network which learns a topological representation of a data distribution from examples with Hebbian learning. Assume data are taken from the real-vector space $\mathbb{R}^n$ equipped with the Euclidian metric $\|\cdot\|$. The SOM exists of a set of neurons $\mathcal{N} = \{n_1, \ldots, n_N\}$ together with a neighborhood structure of the neurons $nh : \mathcal{N} \times \mathcal{N} \to \mathbb{N}$. This is often determined by a regular lattice structure, i.e. neurons n_i and n_j are direct neighbors with $nh(n_i, n_j) = 1$ if they are directly connected in the lattice. A two-dimensional lattice offers the possibility of easy visualization which is used e.g. for applications in data mining [32]. Each neuron n_i is equipped with a weight $w_i \in \mathbb{R}^n$ which represents the corresponding region of the data space. Given a set of training patterns $\mathcal{T} = \{a_1, \ldots, a_d\}$ in $\mathbb{R}^n$, the weights of the neurons are adapted by Hebbian learning including neighborhood cooperation such that the weights w_i represent the training points $\mathcal{T}$ as accurately as possible and the topology of the neurons in the lattice matches the topology induced by the data points. The precise learning rule is very intuitive:

repeat: choose $a_i \in \mathcal{T}$ at random
compute the neuron n_{i_0} with minimum distance $\|a_i - w_{i_0}\|^2 = \min_j \|a_i - w_j\|^2$
adapt for all j: $w_j := w_j + \eta(nh(n_j, n_{i_0}))(a_i - w_j)$

where $\eta(nh(n_j, n_{i_0}))$ is a learning rate which is maximum for the winner $n_j = n_{i_0}$ and decreasing for neurons n_j which are not direct neighbors of the winner n_{i_0}. Often, it is of the form $\exp(-nh(n_j, n_{i_0}))$ possibly with additional constant factors involved in the term. The incorporation of topology in the learning rule allows to adapt the winner and all neighbors in each training step. After training, the SOM is used with the following dynamic: given a pattern $a \in \mathbb{R}^n$, the map outputs the winner, i.e. the neuron with smallest distance $\|a - w_j\|^2$ or its weight, respectively. This allows to identify a new data point with an already learned prototype. Starting from the winning neuron, browsing through the map in order to find similar known data is possible.

Popular alternative self-organizing algorithms are vector quantization (VQ) and the neural gas algorithm (NG) [34]. VQ aims at learning a representation of the data points without topology preservation. Hence no neighborhood structure is given in this case and the learning rule adapts only the winner in each step:

repeat: choose $a_i \in \mathcal{T}$ at random
compute the neuron n_{i_0} with minimum distance $\|a_i - w_{i_0}\|^2 = \min_j \|a_i - w_j\|^2$
adapt $w_{i_0} := w_{i_0} + \eta(a_i - w_{i_0})$

where $\eta > 0$ is a fixed learning rate. NG does not priorly fix a topology of the neurons since a wrong choice may yield topological defects hence an inappropriate representation of data. Rather, neighborhood is defined posteriorly through training data. The recursive update reads as

repeat: choose $a_i \in \mathcal{T}$ at random
compute the distances $\|a_i - w_j\|^2$
adapt for all j: $w_j := w_j + \eta(rk(i,j))(a_i - w_j)$

where $rk(i,j)$ denotes the rank of neuron n_j according to the distances $\|a_i - w_j\|^2$, i.e. the number of neurons n_k such that $\|a_i - w_k\|^2 < \|a_i - w_j\|^2$. $\eta(rk(i,j))$ is a function with maximum at 0 and decreasing values for larger numbers, e.g. $\eta(rk(i,j)) = \exp(-rk(i,j))$ possibly with additional constant factors. I.e. the closest neuron is adapted most, all other neurons are adapted according to their distance to the given data point. Hence the respective order of the neurons with respect to a given training point determines the neighborhood. Eventually, those neurons become neighbored which are the neurons closest to at least one data point. One can introduce a perfect though no longer regular lattice after training in this way [33, 34].

There exist various possibilities of extending self-organizing maps such that they can deal with more complex structures instead of simple vectors in $\mathbb{R}^n$. The article [3] provides an overview of self-organizing networks which have been proposed for processing spatio-temporal patterns. Naturally, a very common way of processing structured data with self-organizing mechanisms relies on adequate preprocessing of sequences such that they are represented through features in a finite dimensional vector space. Then standard pattern recognition methods for simple vectors can be used. A simple way represents sequences with a truncated and fixed dimensional time-window of data. Since this method often yields to large dimensions, SOM with the standard Euclidian metric suffers from the curse of dimensionality and methods which adapt the metric as proposed for example in [21, 27, 40] are advisable. Hierarchical and adaptive preprocessing methods which involve SOMs at various levels can be found e.g. in the WEBSOM approach for document analysis [32]. Since self-organizing algorithms can immediately be transformed to arbitrary metrical structures instead of the standard Euclidian metric, one can alternatively define a complex metric adapted to structured data instead of complex preprocessing data. SOMs equipped with the edit distance constitute one example [16]. Data structures might be contained in a discrete space instead of a real vector space in these approaches. In this case one has to additionally specify how the weights of neurons can be adapted. If the edit distance is dealt with, one can perform a limited number of operations which transform the actual weight of the neuron to the given data structure, for example. Some methods propose the representation of complex data structures in SOM with complex patterns, e.g. the activation of more than one neuron. Activation patterns arise through recursive processing with appropriate decaying and blocking of activations like in SARDNET [25]. Alternative methods within these lines can be found in [3].

We are here interested in approaches which equip the SOM with additional recursive connections which make dynamic processing of a given recursive data structure possible. In particular, no prior metric or preprocessing is chosen but the similarity of structures evolves through the recursive comparison of the single parts of the data. Various approaches within this line have been proposed in the literature. Most of them deal with sequences of real vectors as input data. One approach has been proposed for more general tree structures and thus also sequences as a special case. We shortly summarize important dynamics within these lines in the following. Thereby, sequences over $\mathbb{R}^n$ with entries a_1 to a_t are denoted by $[a_1, \ldots, a_t]$. $[\,]$ denotes the empty sequence. Trees with labels in $\mathbb{R}^n$ are denoted in a prefix notation by $a(t_1, \ldots, t_k)$ where a denotes the label of the root, t_1 to t_k the subtrees. The empty tree is denoted by ξ.

Temporal Kohonen Map

The temporal Kohonen map proposed by Chappell and Taylor [7] extends the SOM by recurrent self-connections of the neurons such that the neurons act as leaky integrators. Given a sequence $[a_1, \dots, a_t]$, the activation of neuron n_i with weight w_i is computed as

$$\sum_{j=1}^{t} \alpha \cdot (1-\alpha)^{(t-j)} \cdot \|a_j - w_i\|^2$$

where $\alpha \in (0,1)$ is a constant which determines the integration of context information if the winner is computed. This formula has the form of a leaky integrator which integrates previous activations of neuron n_i given the entries of the sequence. Hence a neuron becomes the winner if its weight is close to the given data point and, in addition, the exponentially weighted distance of previous entries of the sequence from the weight is small. Obviously, an alternative recursive formalization of the activation $d_i(j)$ of neuron i after the j th time step is given by $d_i(j) = \alpha\|a_j - w_i\|^2 + (1-\alpha)d_i(j-1)$ where $d_i(0) := 0$, Training of the TKM is performed in the reference [7] with Hebbian learning after each time step, i.e. the weights w_i are adapted in each time step with respect to the actual input a_j according to the standard SOM update rule. Thereby, the winner is computed as the neuron with the least activation according to the above leaky integration formula.

The recurrent SOM (RSOM) as defined in the article [31], for example, uses a similar dynamic. However, it integrates the directions of deviations of the weights. Hence the activation $d_i(j)$ which is now an element of $\mathbb{R}^n$ is recursively computed by $d_i(0) = 0$, $d_i(j) = \alpha(a_j - w_i) + (1-\alpha)d_i(j-1)$. The winner is determined as the neuron with smallest integrated distance, i.e. smallest $\|d_i(j)\|^2$. Naturally, this procedure stores more information than the weighted distance of the TKM. Again, learning can be done with Hebbian learning after each time step like in the TKM. In [31] an alternative direct training method is proposed for both, SOM and TKM, which analytically solves the condition that at an optimum in the weight space the derivative of the quantization error is zero. This quadratic equation can be solved analytically which yields solutions for optimum weights w_i. Apart from sequence recognition tasks, these models have been successfully applied for learning motion-directivity sensitive maps as can be found in the visual cortex [10].

Recursive SOM

The recursive SOM (RecSOM) has been proposed by Voegtlin [45, 47] as a mechanism for sequence prediction which recursively processes the symbols based on the already computed context. Each neuron is equipped with a weight $w_i \in \mathbb{R}^n$ and, additionally, with a context vector $c_i \in \mathbb{R}^N$ which stores an activation profile of the whole map, indicating, in which sequential context the vector w_i should arise. Given the sequence $[a_1, \dots, a_t]$, the activation of neuron n_i at time step j is defined as $d_i(0) = 0$ and

$$d_i(j) = \alpha \cdot \|a_j - w_i\|^2 + \beta \cdot \|(\exp(-d_1(j-1), \dots, \exp(-d_N(j-1))) - c_i\|^2$$

where $\alpha, \beta > 0$ are constants. Hence the respective symbol is compared with the weights w_i. In addition, the already computed context, i.e. the activation of the whole map in the previous time step, has to match with the context c_i of neuron n_i such that the neuron becomes the winner. The comparison of contexts is done involving the exponential function $\exp$ in order to avoid numerical explosion. If this exponential transform was not included, the activation $d_i(j)$ could become very huge because the distances with respect to all of the N components of the context accumulate. The RecSOM has been applied for sequence recognition and prediction of the Mackey-Glass time series or sequences induced by a randomized automaton, respectively. Training has been done with Hebbian learning in these cases for both, the weights w_i of the neurons and their contexts c_i. I.e. if a winner n_i has been computed recursively, the parameters (w_i, c_i) of this neuron are adapted towards the actual input a_j and the computed context, i.e. the activity of the map if the predecessors of a_j are processed; the neighbors of n_i are adapted accordingly with a smaller learning rate. The RecSOM shows a good capability of differentiating input sequences; i.e. the winners represent different sequences which have been used for training [45, 47].

SOM for structured data

The SOM for structured data (SOMSD) has been proposed for processing of labeled trees with fixed fan-out k [41]. The specific case $k = 1$ covers sequences. It is assumed that the neurons are arranged on a regular d-dimensional lattice structure. Hence neurons can be enumerated by tuples $\vec{\imath} = (i_1, \ldots, i_d)$ in $\{1, \ldots, N_1\} \times \ldots \times \{1, \ldots, N_d\}$ where $N_i \in \mathbb{N}$ with $N_1 \cdot \ldots \cdot N_d = N$. Each neuron $n_{\vec{\imath}}$ is equipped with a weight $w_{\vec{\imath}}$ and k context vectors $c_1^{\vec{\imath}}, \ldots, c_k^{\vec{\imath}}$ in $\mathbb{R}^d$. They represent the context of a processed tree given by the k subtrees and the indices of their respective winners. The index $I(t)$ of the winning neuron given a tree t with root label a and subtrees $t_1, \ldots, t_k$ is recursively defined by

$$
\begin{aligned}
I(\xi) &= (-1, \ldots, -1) \\
I(a(t_1, \ldots, t_k)) &= \operatorname{argmin}_{\vec{\imath}}\{\alpha \cdot \|a - w_{\vec{\imath}}\|^2 + \beta \cdot \|I(t_1) - c_1^{\vec{\imath}}\|^2 + \ldots + \beta \cdot \|I(t_k) - c_k^{\vec{\imath}}\|^2 \mid \vec{\imath}\}
\end{aligned}
$$

Where $t = a(t_1, \ldots, t_k)$. Hence the respective context of a label in the tree is given by the winners for the k subtrees. Hereby, the empty tree is represented by an index not contained in the lattice, $(-1, \ldots, -1)$. The weights of the neurons consist of a compact representation of trees with prototypical labels and prototypical winner indices which represent the subtrees. Starting from the leaves, the winner is recursively computed for an entire tree. Even for SOMSD, Hebbian learning is applied. Starting from the leaves, the index $\vec{\imath}$ of the winner for the several subtrees is computed. The attached weights $(w_{\vec{\imath}}, c_1^{\vec{\imath}}, \ldots, c_k^{\vec{\imath}})$ are moved into the direction $(a, I(t_1), \ldots, I(t_k))$ after each recursive processing step, where $I(t_i)$ denotes the winning index of the subtree t_i of the currently processed part of the tree. The neighborhood is updated into the same direction with corresponding smaller learning rate.

This method has been used for the classification of pictures represented through tree structured data [17, 18]. During learning, a representation of the pictures in the SOMSD emerges which groups pictures described by similarly structured trees with similar labels together. In the reported experiments, the synthetically designed pictures correspond to ships, houses, and policemen with various shapes and colors. SOMSD is capable of grouping objects of the same category, e.g. houses, together. Within the clusters a differentiation with respect to the involved labels like in the standard SOM can be observed. Based on this clustering a good classification accuracy of the objects can be obtained attaching appropriate classes to the nodes in the SOMSD [17].

3 A general framework for the dynamic

The idea for a general framework for these models consists in the observation that all models share the basic recursive dynamics. They differ in the way how tree structures are internally represented with respect to the recursive processing and the weights of neurons. The above models deal with either sequences over a real vector space or labeled trees with fan-out k. For simplicity, we will here focus on binary trees with labels in some set $\mathcal{W}$. The generalization to trees with fan-out k, in particular sequences which are trees with fan-out 1 will be immediate. We allow an arbitrary set $\mathcal{W}$ instead of a real vector space to include the case of discrete labels such as symbolic data. Denote by $\mathcal{W}^\#$ the set of binary trees with labels in $\mathcal{W}$. We use a prefix notation if the single parts of a tree should be made clear; $a(t_1, t_2)$ refers to the tree with root label a and subtrees t_1 and t_2. As above, the empty tree is denoted by ξ. How can these type data be processed in an unsupervised fashion? We here define the basic ingredients and the basic dynamic or functionality of the map. I.e. given a trained map and a new input tree, how is the winner for this tree computed? Based on this general dynamic which includes as a special cases the above mechanisms TKM, RecSOM, and SOMSD, a canonic formulation of Hebbian learning will be derived in the next section.
The processing dynamic is based on the following main ingredients which define the general SOM for structured data or GSOMSD, for short:

1. The set of labels $\mathcal{W}$ together with a similarity measure $d_{\mathcal{W}} : \mathcal{W} \times \mathcal{W} \to \mathbb{R}$.

2. A set of formal representations $\mathcal{R}$ for trees together with a similarity measure $d_{\mathcal{R}} : \mathcal{R} \times \mathcal{R} \to \mathbb{R}$. Denote the priorly fixed formal representation of the empty tree ξ by r_ξ. The adequate formal representation of any other trees will be computed via the GSOMSD.

3. A set of neurons $\mathcal{N}$ of the self organizing map which we assume to be enumerated with 1, …, $N = |\mathcal{N}|$, for simplicity. A weight function $\mathcal{L} = (\mathcal{L}_0, \mathcal{L}_1, \mathcal{L}_2) : \mathcal{N} \to \mathcal{W} \times \mathcal{R} \times \mathcal{R}$ attaches a weight to each neuron. This consists for each neuron in the map of a triple consisting of a prototype label and two formal representations.

4. A representation mapping *rep* : $\mathbb{R}^N \to \mathcal{R}$ which maps a vector of activations of all neurons into a formal representation.

The idea behind these ingredients is as follows: A tree can be processed iteratively; starting from the leaves, it can be compared with the information contained in GSOMSD until the root is reached. In each step of the comparison the context information resulting from the processing of the two subtrees should be taken into account. Hence a neuron weighted with (w, c_1, c_2) in the map is a proper representation of the entire tree $a(t_1, t_2)$ if the first part of the weight attached to the neuron, w, represents a properly, and c_1 and c_2 represent correct contexts, i.e. they correspond to t_1 and t_2. w and a can be compared using the similarity measure $d_\mathcal{W}$ directly. c_1 and c_2 are formal descriptions of the contexts, which could be some pointers or some reduced description for the context and which should be compared with t_1 and t_2, respectively. For this purpose, t_1 and t_2 can be iteratively processed yielding a context, i.e. an activation of all neurons in the map. This activation can be transferred to a formal description of the context via the representation mapping *rep*. The comparison of the output with c_i is then possible using the metric $d_\mathcal{R}$ on formal representations.

This informal description can be formalized with the following definition which allows to compute recursively the *recursive similarity* or activation $\tilde{d}$ of a node n_i given a tree $a(t_1, t_2)$:

$$\tilde{d}(a(t_1, t_2), n_i) = \alpha \cdot d_\mathcal{W}(a, \mathcal{L}_0(n_i)) + \beta \cdot d_\mathcal{R}(R_1, \mathcal{L}_1(n_i)) + \beta \cdot d_\mathcal{R}(R_2, \mathcal{L}_2(n_i))$$

where

$$R_j = \begin{cases} r_\xi & \text{if } t_j = \xi \\ \mathit{rep}(\tilde{d}(t_j, n_1), \ldots, \tilde{d}(t_j, n_N)) & \text{otherwise} \end{cases}$$

for $j = 1, 2$ and $\alpha, \beta > 0$ are weighting factors. The choice of α and β determines the importance of a proper context in comparison to the correct root label of the representation. The set $\mathcal{R}$ enables a precise notation of how trees are internally stored in the map. The function *rep* constitutes the interface which maps activity profiles to internal representations of trees. The recursive similarity gives the activation of all neurons for an input tree. Applying *rep*, we can obtain a formal representation of the tree. In order to use the resulting map for example for information storing and recovering, we can determine a neuron with highest responsibility, i.e. a neuron which fires or the winner, given the input tree. This could be the neurons with highest or lowest recursive similarity, depending on the meaning of $\tilde{d}$, e.g. depending on the fact whether dot products or distances are used for the computation of the similarities. A picture which explains the processing in one recursive step can be found in Fig. 1.
Note the following

1. The labels of the input trees might come from a proper subset of $\mathcal{W}$, e.g. if discrete data are processed, and the neuron's representation lies in a real vector space. $d_\mathcal{W}$ is often the squared standard Euclidian metric or induced by any other metric; however, at this point, we do not need special properties of $d_\mathcal{W}$.

2. The formal representations $\mathcal{R}$ should represent trees in a compact way, e.g. in a finite dimensional vector space. As an example, they could be chosen simply as the index of the neuron with best recursive similarity if a tree is processed with a trained self organizing map. The idea behind is that a connectionistic distributed representation of a tree emerges from the processing of the tree with the map. The formal description could be a shorthand notation, a pointer for this activity profile. Based on this assumption we can compare trees via comparing their formal descriptions.

3. The neurons can be seen as prototypes for trees, i.e. their root label and their subtrees. The latter are represented by their formal representations. Note that we did not introduce any lattice or topological structure of the neurons at this point. The definition of a topology of the self organizing map does not affect the recursive similarity of neurons given a tree. However, a topological structure might be used for training or it might prove beneficial for specific applications such as browsing in topological representations of the given data space via the map.

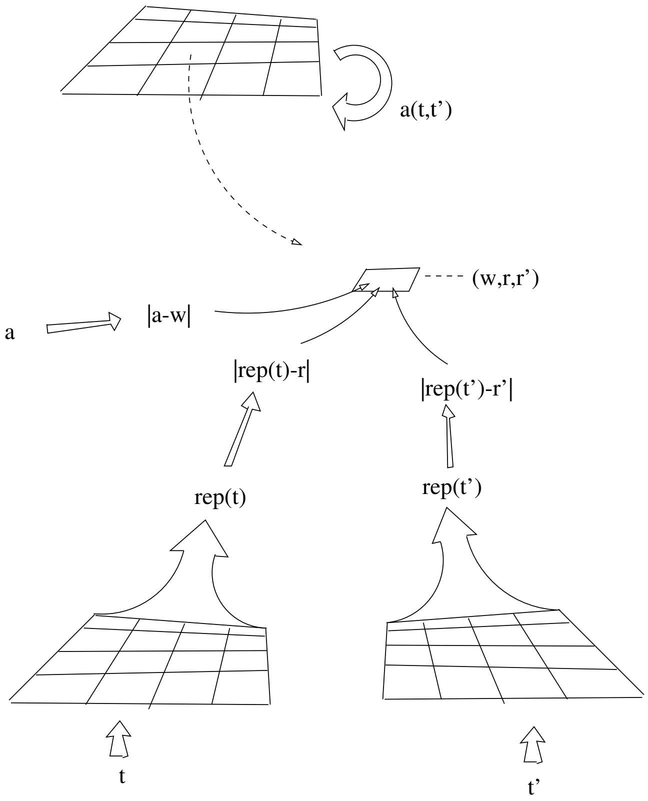

Figure 1: One recursive processing step: given a tree $a(t, t')$, the distance from a neuron weighted with (w, r, r') is computed weighting the distances of a from w, and the distances of r and r', respectively, from the representations of t and t'. These latter representations can be obtained recursively processing the trees and applying *rep* to the obtained activity profile of the map.

4. The mapping *rep* maps the recursively processed trees which yield an activity profile of the neurons into a formal description of trees. It might be simply the identity or some compactification, for example. However, it is no longer a symbolic tree representation but something similar to connectionistic descriptions based on the activation of the GSOMSD.

Obviously, the GSOMSD can easily be defined for trees with fan-out k where $k \neq 2$. In particular, the case of sequences, i.e. $k = 1$ is then included. The weights of the neurons then change to

3. The weight function is of the form $\mathcal{L} = (\mathcal{L}_0, \mathcal{L}_1, \ldots, \mathcal{L}_k) : \mathcal{N} \to \mathcal{W} \times \mathcal{R}^k$, i.e. k contexts are attached to the neurons corresponding to the fan-out k instead of 2.

The recursive processing reads as

$$\tilde{d}(a(t_1, \ldots, t_k), n_i) = \alpha \cdot d_{\mathcal{W}}(a, \mathcal{L}_0(n_i)) + \beta \cdot d_{\mathcal{R}}(R_1, \mathcal{L}_1(n_i)) + \ldots + +\beta \cdot d_{\mathcal{R}}(R_2, \mathcal{L}_k(n_i))$$

where

$$R_j = \begin{cases} r_\xi & \text{if } t_j = \xi \\ rep(\tilde{d}(t_j, n_1), \ldots, \tilde{d}(t_j, n_N)) & \text{otherwise} \end{cases}$$

This abstract definition captures the above approaches of structure processing self organizing networks:

Theorem 3.1 *GSOMSD captures the dynamic of SOM, TKM, RecSOM, and SOMSD via appropriate choices of* $\mathcal{R}$*.*

Proof. For SOM, we choose $\mathcal{R} = \emptyset$. Then no context is available and hence only the respective labels of the root of the trees, or elements in $\mathcal{W}$, respectively, are taken into account.
For the TKM, the ingredients are as follows:

1. Define $\mathcal{W} = \mathbb{R}^n$. $d_\mathcal{W}$ is the squared Euclidian metric.

2. $\mathcal{R} = \mathbb{R}^N$ explicitely stores the activation of all neurons if a tree is processed, N denotes the number of neurons. The similarity measure $d_\mathcal{R}$ is here given by the dot product. The representation for the empty tree ξ is the vector $r_\xi = (0, \ldots, 0)$.

3. The weights of the neurons have a special form: in neuron n_i with $\mathcal{L}(n_i) = (w, c_1, c_2)$ the first parameter may be an arbitrary value in $\mathbb{R}^n$. The contexts $c_1 = c_2$ coincide with the unit vector which is one at position i and 0, otherwise. Hence the context represented in the neurons is of a particular simple structure. One can think of the context as a focus: the neuron looks only at its own activation if processing a tree, it doesn't care about the global activation produced by the tree. Mathematically, this is implemented via the dot product $d_\mathcal{R}$ of the context with the unit vector stored by the neuron.

4. *rep* is simply the identity, no further reduction takes place.

Then the recursive dynamic reads as $\tilde{d}(a(t_1, t_2), n_i)) = \alpha \cdot \|a - \mathcal{L}_0(n_i)\|^2 + \beta \cdot \tilde{d}(t_1, n_i) + \beta \cdot \tilde{d}(t_2, n_i)$ for $t_1, t_2 \neq \xi$. If we choose $\beta = 1 - \alpha$ and consider the case of sequences, this is just a leaky integrator as defined above. An analogous dynamic results for tree structures which does not involve global context processing but focuses on the activation of a single neuron. The winner can be computed as neuron with smallest value $\tilde{d}$.
For RecSOM, we define:

1. Define $\mathcal{W} = \mathbb{R}^n$. $d_\mathcal{W}$ is the squared Euclidian metric.

2. Define $\mathcal{R} = \mathbb{R}^N$ where N is the number of neurons in the self organizing map. $d_\mathcal{R}$ is the squared Euclidian metric. Here the formal description of a tree is merely identical to the activation of the neurons in the self organizing map when processing the tree. No reduction takes place. The representation for the empty tree ξ is the origin $(0, \ldots, 0)$.

3. The neurons are organized on a lattice in this approach; however, this does not affect the processing dynamics.

4. Choose *rep* $=$ *rep*$_R$ where *rep*$_R(x_1, \ldots, x_N) = (\exp(-x_1), \ldots, \exp(-x_N))$. The exponential function is introduced merely for stability reasons. On the one hand, it avoids the similarities to blow up during recursive processing; on the other hand, it scales the similarities in a nonlinear fashion such that small similarities become close to zero. This has the side effect of suppressing noise in these values which could disturb the computation due to the large dimensionality of $\mathcal{R}$. However, no information is gained or lost by the addition of the exponential function.

These definitions yield to the recursive formula $\tilde{d}(a(t_1, t_2), n_i) = \alpha \cdot \|a - \mathcal{L}_0(n_i)\|^2 + \beta \cdot \|R_1 - \mathcal{L}_1(n_i)\|^2 + \beta \cdot \|R_2 - \mathcal{L}_2(n_i)\|^2$ where $R_j = (\exp(-\tilde{d}(t_j, n_1)), \ldots, \exp(-\tilde{d}(t_j, n_N)))$ for $j = 1, 2$ and $t_1, t_2 \neq \xi$. If we restrict to the corresponding fan-out 1, we get the dynamic of RecSOM as introduced above. The winner is again computed as neuron with smallest value $\tilde{d}$.
We choose for SOMSD:

1. Define $\mathcal{W} = \mathbb{R}^n$, $d_\mathcal{W}$ is the squared Euclidian metric.

2. $\mathcal{R}$ is the real vector space which contains the set of indices of neurons in the self organizing map. If the neurons lie on a d-dimensional lattice, this equals $\mathbb{R}^d$. $d_\mathcal{R}$ is the squared Euclidian distance of these points. In addition, the representation of the empty tree ξ is a singular point r_ξ outside the lattice, e.g. $(-1, \ldots, -1)$.

3. The neurons have a topological structure; they lie on a d-dimensional lattice. The weighting attaches appropriate values in $\mathcal{W} \times \mathcal{R} \times \mathcal{R}$ to all neurons which are trained by Hebbian learning. Note that the above enumeration $1, \ldots, N$ of neurons which we assumed for the neurons can be substituted by an enumeration based on this lattice with elements $\vec{\imath} \in \{1, \ldots, N_1\} \times \ldots \times \{1, \ldots, N_d\}$.

4. We choose *rep* $=$ *rep*$_S$ as follows: the input domain of *rep*$_S$, the set $\mathbb{R}^N$ where N denotes the number of neurons, can be identified with $\mathbb{R}^{N_1 \times \ldots \times N_d}$. *rep*$_S$ maps a vector of similarities to the index of the neuron with best similarity. *rep*$_S(x_{(1,\ldots,1)}, \ldots, x_{(N_1,\ldots,N_d)}) = \vec{\iota}$ such that $x_{\vec{\iota}}$ is the smallest input. Note that this need not be unique. We assume here that there is given an ordering on the neurons. In the case of multiple optima, we assume implicitly that the first optimum is chosen.

With this definition we obtain for nonempty trees t_1 and t_2 the general formula for the recursive similarity $\tilde{d}(a(t_1, t_2), n_i) = \alpha \cdot \|a - \mathcal{L}_0(n_i)\|^2 + \beta \cdot \|R_1 - \mathcal{L}_1(n_i)\|^2 + \beta \cdot \|R_2 - \mathcal{L}_2(n_i)\|^2$ where R_1 and R_2 are the indices of the neurons which are the winner for t_1 and t_2, respectively. The winner of the map can again be computed as the neuron with smallest $\tilde{d}$. □

Note that in all of the above examples $\tilde{d}$ corresponds to a distance which has been computed within the GSOMSD. This is the standard setting which can be found in the standard SOM, too: given an input, the neurons compute an activation which measures their distance from the respective input. The winner can afterwards be determined as the neuron with smallest distance. Note that this interpretation as distance is common, but we could more generally interprete the similarity $\tilde{d}$ as the activation of the neurons given an input signal. Then general supervised recurrent and recursive networks are included in the above framework, too, as can be seen as follows: Various different dynamics for discrete time recurrent neural networks (RNN) have been established in the literature [35]. Most of the models can be simulated within the following simple dynamic as proved in the reference [19]: Assume sequences with entries in $\mathbb{R}^n$ are dealt with. The N neurons $n_1, \ldots, n_N$ are equipped with weights $w^i = (w_0^i, w_1^i) \in \mathbb{R}^{n+N}$. Denote by $\tanh$ the hyperbolic tangent. Then the activation of the neurons $\tilde{D} \in \mathbb{R}^N$ given a sequence $[a_1, \ldots, a_t]$ can be computed as

$$
\begin{aligned}
&\tilde{D}([\,]) = (0, \ldots, 0),\\
&\tilde{D}([a_1, \ldots, a_t]) =\\
&(\tanh(w_0^1 \cdot a_t + w_1^1 \cdot \tilde{D}([a_1, \ldots, a_{t-1}])), \ldots, \tanh(w_0^N \cdot a_t + w_1^N \cdot \tilde{D}([a_1, \ldots, a_{t-1}])))
\end{aligned}
$$

where '$\cdot$' denotes the dot product of vectors. Often, a linear function is added in order to obtain the desired output from $\tilde{D}$. RNNs can be generalized to so-called recursive neural networks (RecNN) which can deal with tree structures. We here only introduce the dynamic for binary trees with labels in $\mathbb{R}^n$. Neurons are weighted with $w = (w_0^i, w_1^i, w_2^i) \subset \mathbb{R}^{n+N+N}$. The activation $\tilde{D} \in \mathbb{R}^N$ can recursively be computed by

$$
\begin{aligned}
&\tilde{D}(\xi) = (0, \ldots, 0),\\
&\tilde{D}(a(t_1, t_2)) =\\
&(\tanh(w_0^1 \cdot a + w_1^1 \cdot \tilde{D}(t_1) + w_2^1 \cdot \tilde{D}(t_2)), \ldots, \tanh(w_0^N \cdot a_t + w_1^N \cdot \tilde{D}(t_1) + w_2^N \cdot \tilde{D}(t_2))))
\end{aligned}
$$

where again often a linear function is added to obtain the final output. Obviously, RecNNs can be formulated within the dynamics of GSOMSDs, choosing $\alpha = \beta = 1$ and choosing in the recursive computation the following ingredients:

1. $\mathcal{W} = \mathbb{R}^n$, $d_{\mathcal{W}}$ is the dot product.

2. $\mathcal{R} = \mathbb{R}^N$ stores the nettoinput of all neurons, $d_{\mathcal{R}}$ is the dot product, $r_\xi = (0, \ldots, 0)$.

3. the N neurons are equipped with the weights $\mathcal{L}_0(n^i) = w_0^i$, $\mathcal{L}_1(n^i) = w_1^i$, and $\mathcal{L}_2(n^i) = w_2^i$.

4. *rep*$(x_1, \ldots, x_N) = (\tanh(x_1), \ldots, \tanh(x_N))$.

Then we obtain the so-called nettoinput of the neurons as recursive similarity $\tilde{d}$, i.e. $\tanh(\tilde{d}) = \tilde{D}$ where $\tanh$ denotes component-wise application of the hyperbolic tangent. Hence if we substitute the computation of the final winner by the function $\tanh$ and a possibly added further linear function, we obtain the processing dynamic of recurrent and recursive networks within the GSOMSD dynamic.

We could even allow more general activations of the neurons than simple one-dimensional vectors which need not correspond to distances. Then a dimensionality l of the activation vector of the neurons is fixed and $\tilde{d} \in \mathbb{R}^l$ and $d_{\mathcal{R}}$ and $d_{\mathcal{W}}$ are l-dimensional, too. The sum $\alpha \cdot d_{\mathcal{W}} + \beta \cdot d_{\mathcal{R}} + \beta \cdot d_{\mathcal{R}}$ denotes scalar multiplication and addition of vectors. This setting allows to model RSOM, for example, which, alternatively to the TKM, stores the leaky integration over the directions instead of the leaky integration of distances. We obtain the analogon of RSOM for binary trees if we choose $\mathcal{R} = (\mathbb{R}^N)^l$, $\mathcal{L}_1(n_i) = \mathcal{L}_2(n_i) = (e_i, \ldots, e_i)$, e_i denoting the i th unit vector in $\mathbb{R}^N$, $d_\mathcal{R}((r_1^1, \ldots, r_l^1), (r_1^2, \ldots, r_l^2)) = (r_1^1 \cdot r_1^2, \ldots, r_l^1 \cdot r_l^2)$, *rep* as the identity, and $d_\mathcal{W}(a_1, a_2) = a_1 - a_2$. Then the recursive dynamic becomes $\tilde{d}(a(t_1, t_2), n_i) = \alpha(a - \mathcal{L}_0(n_i)) + \beta\tilde{d}(t_1, n_i) + \beta\tilde{d}(t_2, n_i)$. For fan-out 1 and $\beta = (1 - \alpha)$, this is the recursive formula of RSOM.

Of course, a more general computation of the combination of similarities than a simple weighted sum makes sense, too. Then the computation $\alpha d_\mathcal{W} + \beta d_\mathcal{R} + \beta d_\mathcal{R} : \mathcal{W}^2 \times \mathcal{R}^2 \times \mathcal{R}^2 \to \mathbb{R}^l$ is substituted by some mapping $\Phi : \mathcal{W}^2 \times \mathcal{R}^2 \times \mathcal{R}^2 \to \mathbb{R}^l$ which maps the inputs given by the actual processed tree and the weights attached to a neuron to the new activation of this neuron. Such more general mechanism could for example model the so-called extended Kohonen feature map (EKFM) or the self-organizing temporal pattern recognizer (SOTPAR) [9, 24]. The EKFM further equips the SOM with weighted connections w_{ij} from neuron i to j which are trained with some form of temporal Hebbian learning. Given an input sequence, a neuron n_i can only become active, if it is the neuron with smallest distance from the actual pattern, and, in addition, the weighted sum of all activations of the previous time step weighted with the weights w_{ji} is larger than a certain threshold θ_i. It might happen that no winner is found if the sequence to be processed is not known. This corresponds to the fact that the neuron with smallest distance does not become active due to a small support from the previous time step. This computation can be modeled within the proposed dynamics if $\tilde{d}$ is two-dimensional for each neuron, the first part storing the actual distances, the second part storing the fact, whether the neuron finally becomes active, and an appropriate choice of Φ. SOTPAR activates at each step the neuron with minimum distance from the actual processed label, where all neurons which lie in the neighborhood of a recently activated neuron have a benefit, i.e. they tend to become winner more easily. This is modeled storing a bias within each neuron which is initialized with 0 and which is adapted if a neighbored neuron or the neuron itself becomes winner, and which gradually decreases over time to 0. This bias is subtracted from the actual distance in each time step, such that the effective winner is the neuron with smallest distance and largest bias. Again, this dynamic can easily be modeled within the above framework if the states of the neurons are two-dimensional, storing the respective distance and the actual bias of each neuron, and if Φ is chosen appropriately. However, we will in the following focus on the more specific dynamics where the states of the neurons are one-dimensional and correspond to a distance or similarity, and where Φ has the specific form of a weighted combination of the metrics or similarity measures $d_\mathcal{R}$ and $d_\mathcal{W}$, as defined above. Moreover, if we describe the consequences of the general dynamic and respective learning algorithms for concrete settings, we will mainly focus on SOMSD and RecSOM adapted for binary trees. These two methods both rely on reasonable contexts, the winner, or the exponentially transformed activation of the entire map, respectively. We will see later that local contexts as used in TKM, for example, are not sufficient for the task of storing trees of arbitrary depth. SOMSD and RecSOM can be seen as two prototypical mechanisms with a different degree of information compression: SOMSD only stores the winner, whereas RecSOM stores an activity profile which, due to the exponential transform, focuses on the winners, too.

4 Hebbian learning

In order to find an appropriate neural map, we assume that a finite set of training data $\mathcal{T} = \{T_1, \ldots, T_d\} \subset \mathcal{W}^\#$ is given. We would like to find a neural map such that the training data is represented as accurately as possible by the neural map. In the following, we discuss training methods which adapt the weights of the neurons, i.e. the triples (x, r_1, r_2) attached to the neurons. That means, we assume that only the function $\mathcal{L}$ can be changed during training, for simplicity. Naturally, other parts could be adaptive as well such as the weighting terms α and β, the computation of formal representations *rep*, or even the similarities $d_\mathcal{W}$ and $d_\mathcal{R}$. Note that some of the learning algorithms discussed in the following can immediately be transferred to the adaptation of additional parameters involved in e.g. *rep*.

There are various ways of learning in self organizing maps. One of the simplest and most intuitive learning paradigm is Hebbian learning. It has the advantage of simple update formulas and it does not require any further properties of e.g. the functions involved in $\tilde{d}$. However, it constitutes at a first glance a heuristic. A mathematical investigation of the learning behavior is often difficult. Alternatives can be found if an objective of the learning process is explicitely formulated in terms of a cost function which is to be optimized. Then gradient descent methods or other optimization techniques such as a genetic algorithm can be applied. A gradient descent requires that the involved functions are differentiable and that $\mathcal{W}$ and $\mathcal{R}$ are real-vector spaces. We will discuss Hebbian learning first and adapt several cost functions for self organizing maps to our approach, afterwards.

We would like to find a proper representation of all trees T_i of the training set $\mathcal{T}$ and all its subtrees in the neural map. Since the computation of the recursive similarities for a tree T_i uses the representation of all its subtrees, it can be expected that an adequate representation of T_i can only be found in the neural map if all subtrees of T_i are properly processed, too. Therefore we assume that for each tree contained in $\mathcal{T}$ all its subtrees are contained in $\mathcal{T}$, too. This property of $\mathcal{T}$ is essential for effective Hebbian learning. It can be expected that this property is beneficial for learning based on an cost function as well. Therefore we assume implicitly in the following that the training set $\mathcal{T}$ is enlarged to contain all subtrees of each $T_i \in \mathcal{T}$. Hebbian learning characterizes the idea of iteratively making the weights of the neuron which fires in response to a specific input more similar to the input. For this purpose some notation of 'moving into the direction of' is required. We assume the following two functions to be given:

- A function $mv_{\mathcal{W}} : \mathcal{W} \times \mathcal{W} \times \mathbb{R} \to \mathcal{W}, (w_1, w_2, \eta) \mapsto mv_{\mathcal{W}}(w_1, w_2, \eta)$ which allows an adaptation of the weights w_1 stored in the first position of a neuron into the direction of w_2 provided by training up to a degree η. If $\mathcal{W}$ is a real vector space, this function is usually just given by the addition of vectors: $mv_{\mathcal{W}}(w_1, w_2, \eta) = w_1 + \eta(w_2 - w_1)$. If discrete weights are dealt with which cannot properly be embedded in a real vector space, this can consist of a discrete decision:

$$mv_{\mathcal{W}}(a, b, \eta) = \begin{cases} a & \text{if } \eta \leq 0.5 \\ b & \text{otherwise} \end{cases}$$

  If a and b are compared using the edit distance, for example, a specific number of operations which transform a into b could be applied depending on the value of η, as proposed in [16] for the standard SOM.

- A function $mv_{\mathcal{R}} : \mathcal{R} \times \mathcal{R} \times \mathbb{R} \to \mathcal{R}, (r_1, r_2, \eta) \mapsto mv_{\mathcal{R}}(r_1, r_2, \eta)$ which allows us to adapt the formal representations in the same way. If the formal representations are contained in a real vector space as in SOMSD or the recursive SOM, for example, this is given by the addition of vectors, too. Alternatively, we can use discrete transformations if $\mathcal{R}$ consists of a discrete set.

Note that we can deal with discrete formal representations in this way, too. We first formulate Hebbian training for the analogon of SOM-training for the GSOMSD. Here an additional ingredient, the neighborhood structure or topology, is used: Assume there is given a neighborhood function

$$nh : \mathcal{N} \times \mathcal{N} \to \mathbb{N}$$

which measures the distance of two neurons with respect to the topology. Often, a lattice structure is fixed a priori and this lattice of neurons is spread over the data during training. It might consist of a two-dimensional lattice and the neurons are enumerated by $(1, 1)$, $(1, 2)$, …, (N_1, N_2) accordingly. For this two-dimensional lattice $nh(n_{(i_1,j_1)}, n_{(i_2,j_2)})$ could be the distance of the indices $|i_1 - i_2| + |j_1 - j_2|$, for example. However, the lattice structure could be more difficult such as a hexagonal grid structure or a grid with exponentially increasing number of neighbors [37]. As a program, Hebbian learning reads in this case as:

initialize the weights at random
repeat: choose some training pattern T
  for all subtrees $t = a(t_1, t_2)$ *in* T *in inverse topological order:*
    compute $\tilde{d}(a(t_1, t_2), n_i)$ *for all neurons* n_i
    compute the neuron n_{i_0} *with best similarity*
    adapt the weights of all neurons n_i *simultaneously:*
      $\mathcal{L}_0(n_i) := mv_{\mathcal{W}}(\mathcal{L}_0(n_i), a, \eta(nh(n_i, n_{i_0})))$
      $\mathcal{L}_1(n_i) := mv_{\mathcal{R}}(\mathcal{L}_1(n_i), R_1, \eta(nh(n_i, n_{i_0})))$
      $\mathcal{L}_2(n_i) := mv_{\mathcal{R}}(\mathcal{L}_2(n_i), R_2, \eta(nh(n_i, n_{i_0})))$

where

$$(*) \quad R_j = \begin{cases} r_\xi & \text{if } t_j = \xi \\ \mathit{rep}(\tilde{d}(t_j, n_1), \ldots, \tilde{d}(t_j, n_N)) & \text{otherwise} \end{cases}$$

for $j \in \{1, 2\}$ and $\eta : \mathbb{R} \to \mathbb{R}$ is a monotonically decreasing function. Often, $\eta(x) = \eta_0 \exp(-x/\sigma^2)$ where $\eta_0 > 0$ is a learning rate which is decreased in each step in order to ensure convergence; $\sigma > 0$ is a term which determines the size of the neighborhood which is taken into account. Usually, σ is decreased during training, too. The dependence $(*)$ requires to recompute the recursive similarity for all subtrees of $a(t_1, t_2)$ after changing the weights which is, of course, very costly. Therefore the recursive similarity $\tilde{d}$ is usually computed only once for each subtree of a tree and the values are uniformly used for the update. This approximation does not change training much for small learning rates. Hebbian learning has been successfully applied for both, SOMSD and RecSOM. See for example the literature [17, 18, 41, 45, 47] for the classification of time series or images, respectively.

We would like to point out that the scaling term $\eta(\mathit{nh}(n_i, n_{i_0}))$ might be substituted by a more general function η which depends not only on the winner but on the whole lattice, i.e. it gets as input the distances $\tilde{d}(a(t_1, t_2), n_i)$ of all neurons n_i. This more general setting covers updates based on the soft-min function or the ranking of all neurons, for example, such as the k-means algorithm.

Two alternatives to SOM proposed in the neural networks literature are vector quantization (VQ) and neural gas (NG) as introduced beforehand. We obtain an analogon for VQ in the case of structured data if we choose the neighborhood structure as

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

This gives a learning algorithm where sequences and trees can be stored because of the respective context, too; however, no topology preservation takes place. A lattice structure is not accounted for by the VQ algorithm. VQ would constitute an alternative to SOM training for the SOMSD and RecSOM. However, the metric $d_{\mathcal{R}}$ of the internal representations $\mathcal{R}$ in SOMSD yields random values which depend on the initialization of the map and the order of the presentation of the examples. The distance of indices of neurons in a given lattice which is not accounted for by the learning algorithm constitutes no useful information. Hence it can be doubted whether formal representation of trees arise if VQ is applied for SOMSD which make sense. In particular, it might happen that very similar trees have distinct internal representations with respect to $d_{\mathcal{R}}$ and vice versa. As a consequence of this argumentation one should treat the indices of neurons as orthogonal directions. Assume the neurons are enumerated by 1, …, N. Then the alternative $\mathit{rep}_V(x_1, \ldots, x_N) = (0, \ldots, 0, 1, 0, \ldots, 0)$ as the vector with entry 1 at place i iff $x_i \le x_j$ for all $j \ne i$ instead of rep_R has the effect of decoupling the indices. Introducing this representation into vector quantization for the SOMSD seems more promising. Note that we can easily apply a form of Hebbian learning to this representations. A fast computation can look like follows: we choose the similarity measure $d_{\mathcal{R}}$ as the dot product. We restrict $\mathcal{R} = \{r \in \mathbb{R}^N \mid \sum r_i = 1\}$. We can store the formal representations attached to a neuron in the form $(r_1, \ldots, r_N)/c$ where c is a global scaling factor for the components. Given a tree $a(t_1, t_2)$ such that neuron n_0 is the winner for $a(t_1, t_2)$ and the neurons n_1 and n_2 are the winners for t_1 and t_2, respectively, we adapt the weight $\mathcal{L}_1(n_0)$ and $\mathcal{L}_2(n_0)$ in the following way with Hebbian learning: the component representing n_1 and n_2, respectively, is increased by a fixed constant ϵ, the global scaling factors c for both representations are increased by ϵ, too. This has the effect of implicit normalization of the formal representations.

The neural gas algorithm as explained above determines the respective neighborhood from the given training example and its distances to all neurons. Hence the update in the above algorithm given a tree t_j becomes

$$\begin{aligned} \mathcal{L}_0(n_i) &:= \mathit{mv}_{\mathcal{W}}(\mathcal{L}_0(n_i), a, \eta(\mathit{rk}(j, i))) \\ \mathcal{L}_1(n_i) &:= \mathit{mv}_{\mathcal{R}}(\mathcal{L}_1(n_i), R_1, \eta(\mathit{rk}(j, i))) \\ \mathcal{L}_2(n_i) &:= \mathit{mv}_{\mathcal{R}}(\mathcal{L}_2(n_i), R_2, \eta(\mathit{rk}(j, i))) \end{aligned}$$

where $\mathit{rk}(j, i)$ denotes the rank of the neuron n_i ordered according to the actual distance from the input, i.e. if the tree t_j is currently processed, $\mathit{rk}(j, i)$ denotes the number of neurons n_k such that $\tilde{d}(t_j, n_i) < \tilde{d}(t_j, n_k)$. $\eta(x)$ can for example have the form $\eta_0 \exp(-x/\sigma^2)$. This update makes sure, that neurons which have similar activation adapt the respective weights into the same direction. Finally, those neurons are neighbors according to this data driven lattice which constitute the first and second winner for at least one data point. Naturally, the resulting neighborhood structure is less regular and need not come from a lattice. It proposes an alternative to SOM with data oriented lattice. Like in the case of vector quantization, the usefulness of the representation $\mathcal{R}$ in SOMSD as the index of the winner and the choice of $d_{\mathcal{R}}$ as the distance in a priorly fixed lattice which is not taken into account by the learning algorithm for SOMSD can be doubted. Since no lattice is given a priori, the space of indices which constitutes $\mathcal{R}$ in the original approach consists of points which do not possess any prior metrical information. The distance between indices should be dynamically computed during the training algorithm according to the ranking with respect to the respective data point. This behavior can be obtained if we choose $\mathcal{R}$ as $\mathbb{R}^N$, $d_{\mathcal{R}}$ as the squared Euclidian metric, and we choose the representation function rep_N where the ith component of $rep_N(x_1,\ldots,x_n)$ is the number of x_j such that $x_j < x_i$, i.e.

$$rep_N(x_1,\ldots,x_n) = (rk(x_1),\ldots,rk(x_N))\,.$$

Hereby, $rk(x_i)$ denotes the number of x_k with $x_k < x_i$.

5 Learning as cost minimization

For SOM, NG, and VQ effort has been made to derive the Hebbian learning rules alternatively as stochastic gradient methods of appropriate cost functions for simple vector data. Assume a differentiable cost function E is given which can be written in the form $\sum_{a_i} E(a_i)$, a_i denoting a training pattern; a stochastic gradient descent with respect to weights w then has the form

initialize the weights at random
repeat: choose a training pattern a_i
update $w := w - \eta \cdot \dfrac{\partial E(a_i)}{\partial w}$

where $\eta > 0$ is the learning rate. Note that Hebbian learning has a similar form. Assumed we could prove that Hebbian learning obeys a stochastic gradient descent with appropriate choice of η like in the case of simple vectors, then a prior mathematic objective can be identified and alternative optimization methods, in particular global optimization methods such as simulated annealing can be used as an alternative. Moreover, there exist well-known guarantees for the convergence of a stochastic gradient descent if the learning rate η (which may vary over time) fulfills certain properties.

We first have a look at SOM, VQ, and NG for simple vectors. Denote the simple labels or vectors used for training by a_i. The weights of the map are denoted by w_j. The cost function of VQ for the case of simple vectors has the form

$$\frac{1}{2}\sum_i\sum_j \chi(a_i,w_j)\|a_i-w_j\|^2$$

where $\chi(a_i,w_j)$ denotes the characteristic function of the receptive field of w_j, i.e. it is 1 if n_j is the winner for a_i and 0 otherwise. Taking the derivatives with respect to the weights w_j gives the learning rule of VQ. Note, the derivatives of χ are thereby computed e.g. using the δ-function. We here ignore this point and provide a detailed derivation of the formulas including the borders of χ for the structured case. The cost function of NG is of the form [34]

$$\frac{1}{2}\sum_i\sum_j \eta(rk(i,j))\|a_i-w_j\|^2$$

where $rk(i,j)$ denotes the rank of neuron n_j if the neurons are sorted according to their distance from the actual data point. SOM itself does not possess an cost function which could be transferred to the continuous case, i.e. if a data distribution instead of a finite training set is considered. The article [22], for example, proposes a cost function for a slightly modified version of SOM:

$$\frac{1}{2}\sum_i\sum_j \chi_S(i,j)\sum_k \eta(nh(n_j,n_k))\|a_i-w_k\|^2$$

where here $\chi_S(i,j)$ denotes the characteristic function of the receptive field of the winner whereby a slightly modified definition of the winner is used:

$$\chi_S(i,j) = \begin{cases} 1 & \text{if } \sum_k \eta(nh(n_j,n_k))\|a_i - w_k\|^2 \leq \sum_k \eta(nh(n_{j'},n_k))\|a_i - w_k\|^2 \text{ for all } j' \\ 0 & \text{otherwise} \end{cases}$$

Hence the winner in the above cost function as well as the modified learning rule of SOM is not the neuron with smallest distance but the neuron with smallest averaged distance with respect to the local neighborhood.

Obviously, all of the above cost functions have the form

$$\sum_i f(\|a_i - w_1\|^2, \ldots, \|a_i - w_N\|^2).$$

This general scheme allows an immediate transfer of the above cost functions to the structured case: the term $\|a_i - w_j\|^2$ is substituted by the respective recursive similarity of the neurons given a tree structure. We again assume that $\mathcal{T} = \{T_1, \ldots, T_d\}$ is a given set of trees which are used for training. Since usually together with the trees all subtrees should be represented properly, i.e. its corresponding quantization error should be small, we assume that $\mathcal{T}$ is complete with respect to subtrees. Given a neural map for structured data with neurons n_i, the general cost function has the form

$$\sum_{t_i \in \mathcal{T}} f(\tilde{d}(T_i, \mathcal{N}))$$

where $\tilde{d}(T_i, \mathcal{N})$ denotes the vector $(\tilde{d}(T_i, n_1), \ldots, \tilde{d}(T_i, n_N))$. If f is chosen corresponding to the above cost functions of VQ, NG, or SOM, this measure yields an objective for the structured case. Taking the derivatives allows to obtain formulas for a stochastic gradient descent in the structured case. We will see that, unlike the case of simple data, the gradient descent methods for the cost functions of NG, SOM, and VQ differ from the respective Hebbian learning algorithms. Hence unlike the case of simple vectors, Hebbian learning as introduced above is only an approximation of a stochastic gradient descent on the respective cost functions of NG, VQ, or SOM.

We start computing the derivatives of the above general function $(*)$ with respect to the weights of the neurons. For this purpose, we assume that $\mathcal{W}$ and $\mathcal{R}$ are contained in a real vector space, and we here first assume that all involved functions including f, *rep*, $d_{\mathcal{R}}$, and $d_{\mathcal{W}}$ are differentiable. (They are not e.g. for SOMSD, we will discuss this point later.) Assume the derivative with respect to a weight $\mathcal{L}_I(n_J)$ for $I \in \{0, 1, 2\}$ and $J \in \{1, \ldots, N\}$ is to be computed. We find

$$\frac{\partial f(\tilde{d}(T_i, \mathcal{N}))}{\partial \mathcal{L}_I(n_J)} = \sum_{j=1}^{N} \frac{\partial f(\tilde{d}(T_i, N))}{\partial \tilde{d}(T_i, n_j)} \cdot \frac{\partial \tilde{d}(T_i, n_j)}{\partial \mathcal{L}_I(n_J)}$$

The first part depends on the respective cost function, i.e. the choice of f. It often involves the use of δ-functions for computing the derivatives because f is built of characteristic functions. The second component can be computed as follows: Define $\partial_i f(x_1, \ldots, x_n) := \partial f(x_1, \ldots, x_n)/\partial x_i$ for $i \leq n$ as a shorthand notation. If $f = (f_1, \ldots, f_m)$ is a vector of functions, then the term $\partial_i f(x_1, \ldots, x_n)$ denotes the vector $(\partial_i f_1(x_1, \ldots, x_n), \ldots, \partial_i f_m(x_1, \ldots, x_n))$. If x_i is a vector $(x_{i1}, \ldots, x_{im})$, we define $\partial_i f(x_1, \ldots, x_n) = (\partial f(x_1, \ldots, x_n)/\partial x_{i1}, \ldots, \partial f(x_1, \ldots, x_n)/\partial x_{im})$. As above, we use the abbreviation

$$R_j = \begin{cases} r_\xi & \text{if } t_j = \xi \\ rep(\tilde{d}(t_j, n_1), \ldots, \tilde{d}(t_j, n_N)) & \text{otherwise} \end{cases}$$

for $j \in \{1, 2\}$ and we denote the corresponding derivative with respect to a variable x by

$$\partial R_{j,x} = \begin{cases} 0 & \text{if } t_j = \xi \\ \sum_{l=1}^{N} \partial_l rep(\tilde{d}(t_j, n_1), \ldots, \tilde{d}(t_j, n_N)) \cdot \partial \tilde{d}(t_j, n_l)/\partial x & \text{otherwise} \end{cases}$$

for $j \in \{1,2\}$. Then we have

$$\begin{aligned}
\partial\tilde{d}(a(t_1,t_2),n_j)/\partial\mathcal{L}_0(n_J) &= \alpha\,\delta_{jJ}\,\partial_2 d_{\mathcal{W}}(a,\mathcal{L}_0(n_j)) && (1.1)\\
&+ \beta\,\partial_1 d_{\mathcal{R}}(R_1,\mathcal{L}_1(n_j))\cdot\partial R_{1,\mathcal{L}_0(n_J)} && (1.2)\\
&+ \beta\,\partial_1 d_{\mathcal{R}}(R_2,\mathcal{L}_2(n_j))\cdot\partial R_{2,\mathcal{L}_0(n_J)} && (1.3)
\end{aligned}$$

for the first components of the weights where $\delta_{ij} \in \{0,1\}$ is the Kronecker symbol with $\delta_{ij} = 1 \iff i = j$. '$\cdot$' denotes the dot product if $\mathcal{R}$ is contained in a vector space of dimension larger than one. For the formal representations attached to a neuron we find

$$\begin{aligned}
\partial\tilde{d}(a(t_1,t_2),n_j)/\partial\mathcal{L}_1(n_J) &= \beta\,\delta_{jJ}\,\partial_2 d_{\mathcal{R}}(R_1,\mathcal{L}_1(n_j)) && (2.1)\\
&+ \beta\,\partial_1 d_{\mathcal{R}}(R_1,\mathcal{L}_1(n_j))\cdot\partial R_{1,\mathcal{L}_1(n_J)} && (2.2)\\
&+ \beta\,\partial_1 d_{\mathcal{R}}(R_2,\mathcal{L}_2(n_j))\cdot\partial R_{2,\mathcal{L}_1(n_J)} && (2.3)
\end{aligned}$$

and

$$\begin{aligned}
\partial\tilde{d}(a(t_1,t_2),n_j)/\partial\mathcal{L}_2(n_J) &= \beta\,\delta_{jJ}\,\partial_2 d_{\mathcal{R}}(R_2,\mathcal{L}_2(n_j)) && (3.1)\\
&+ \beta\,\partial_1 d_{\mathcal{R}}(R_1,\mathcal{L}_1(n_j))\cdot\partial R_{1,\mathcal{L}_2(n_J)} && (3.2)\\
&+ \beta\,\partial_1 d_{\mathcal{R}}(R_2,\mathcal{L}_2(n_j))\cdot\partial R_{2,\mathcal{L}_2(n_J)} && (3.3)
\end{aligned}$$

Hence the derivatives can be computed recursively over the depth of the tree. Starting at the leaves, we obtain formulas for the derivatives of the respective activation with respect to the weights of the neurons. The complexity of this method is of order $NTWR$, N denoting the number of neurons, T the number of labels of the tree, W the dimensionality of the weights of each neuron, and R the dimensionality of $\mathcal{R}$.

Note that the first summands of the derivatives yield terms which occur in Hebbian learning, too: $\partial_2 d_{\mathcal{R}}(a,b)$ and $\partial_1 d_{\mathcal{R}}(a,b)$ coincide for the squared Euclidian metric with the terms $-2(a-b)$ or $2(a-b)$, respectively. Hence we get the original Hebbian learning rules if we only consider the first summands and involve the respective terms arising from f for NG, VQ, and SOM. This argumentation has two consequences: first, it offers the possibility of transferring the above rules to situations where discrete data are involved and hence $d_{\mathcal{W}}$ and $d_{\mathcal{R}}$ possibly not differentiable. The above argument proposes to substitute the derivative $\partial_2 d_{\mathcal{W}}$ in the above formulas by the function $mv_{\mathcal{W}}$ and the derivative $\partial_2 d_R$ by the function $mv_{\mathcal{R}}$ in order to obtain a heuristic training method for the general case where the labels or formal representations are not given by a real vector space.

Second, Hebbian learning is different from a stochastic gradient descent if structured data are involved. Hebbian learning disregards the contribution of the subtrees because it cuts the latter two summands in the above recursive formulas for the derivatives. Hence unlike the simple vector case where Hebbian learning and a stochastic gradient descent coincide, structured data causes differences for the methods. However, the factor β is usually smaller than 1; i.e. the weighting factor for the two additional terms in the above gradient formulas vanishes exponentially with the depth since it is multiplied in each recursive step by the small factor β. Therefore, Hebbian learning can be seen as an efficient approximation of the precise stochastic gradient descent.

Nevertheless, the formulation as cost minimization allows us to formulate supervised learning tasks within this framework, too. Assume a desired output y_i is given for each tree T_i, too. Then we can train a GSOMSD which approximately yields the output y_i given T_i if we minimize $\sum_i (y_i - \tilde{d}(T_i,\mathcal{N}))^2$ with a stochastic gradient descent. We have already shown that recurrent and recursive networks can be simulated within the GSOMSD dynamic. The introduction of the above error function introduces training methods for these cases, too. The respective formulas are in this case essentially equivalent to the well-known real time recurrent learning and its structured counterpart [36]. Here a Hebbian approximation would yield training with so-called truncated gradient. Note that the well-known problem of long-term dependencies, i.e. the difficulty to latch information through several recursive layers, arises in this context [6, 23].

Another direction can immediately be tackled in the same way: learning vector quantization (LVQ) [29, 30] constitutes a self-organizing supervised training method which allows to learn a prototype based clustering of data with Hebb-style learning rules. The approach [39] proposes a cost function for variants of LVQ and introduces so-called GLVQ. This method constitutes a very stable and intuitive learning method for which additional features like automatic metric adaptation have been developed [21]. Since all these methods rely on cost functions which only depend on the squared Euclidian distance of patterns from the weights of neurons, these methods can immediately be included in the above framework for structured data and above formulas be used for computing the respective derivatives.

The reference to RTRL of course proposes to relate the above method to another standard algorithm of recurrent and recursive network training, backpropagation through time or structure, respectively (BPTT and BTTS) [11, 36]. BPTT has a lower complexity with respect to the number of neurons. The idea of BPTT is to use bottlenecks of the computation, i.e. a parameterization of the cost function including few variables which can substitute the contribution of the activation of all neurons to the respective activation of the neurons in later time steps. Then using the chain rule, the error signals can be backpropagated much more efficiently. One possible bottleneck in the above computation is offered by the formal representation of trees. If $\mathcal{R}$ is low dimensional as in the case of SOMSD, we can compute the derivatives of the respective cost function with respect to the formal representations, first. The other interesting terms can be efficiently computed afterwards. We shortly sketch this alternative. If we use $\mathcal{R}$ as a bottleneck, the complexity of the method is only linear with respect to the number of neurons.

We here assume that the representation function *rep* is R-dimensional with components $\mathit{rep}_1, \ldots, \mathit{rep}_R$. Consider a tree T. For simplicity, we refer to the nodes in T via their labels a_i and we identify the subtree with root a_i and the root label in the notation. The two subtrees of a node a_i are denoted by $\mathit{left}(a_i)$ and $\mathit{right}(a_i)$, its parent node in T is denoted by $\mathit{parent}(a_i)$. We denote the formal representation of a tree a_i by $R(a_i)$ and its components by $R_j(a_i)$. Each weight $\mathcal{L}_i(n_j)$ of the GSOMSD is used multiple times in a recursive computation. We assume that there is given a corresponding number of identical copies $\mathcal{L}_i(n_j)(a_k)$, where $\mathcal{L}_i(n_j)(a_k)$ refers to the copy which is used in the last recursive step for computing $\tilde{d}(a_k, \mathcal{N})$. The part of the cost function which comes from labels in the tree T is denoted by $\mathcal{E}(T) = \sum_{a_k \in T} f(\tilde{d}(a_k, \mathcal{N}))$. The derivative with respect to $\mathcal{L}_i(n_j)$ can then be computed as the sum of all derivatives with respect to one copy $\mathcal{L}_i(n_j)(a_k)$. We find for one copy:

$$\frac{\partial \mathcal{E}(T)}{\partial \mathcal{L}_i(n_j)(a_k)} = \frac{\partial f(\tilde{d}(a_k, \mathcal{N}))}{\partial \tilde{d}(a_k, n_j)} \cdot \frac{\partial \tilde{d}(a_k, n_j)}{\partial \mathcal{L}_i(n_j)(a_k)} + \sum_l \frac{\partial \mathcal{E}(T)}{\partial R_l(a_k)} \cdot \frac{\partial R_l(a_k)}{\partial \mathcal{L}_i(n_j)(a_k)}$$

where $\partial f(\tilde{d}(a_k, \mathcal{N})) / \partial \tilde{d}(a_k, n_j)$ depends on the specific cost function, and the other terms yield

$$\frac{\partial \tilde{d}(a_k, n_j)}{\partial \mathcal{L}_i(n_j)(a_k)} = \begin{cases} \alpha \cdot \partial_2 d_{\mathcal{W}}(a_k, \mathcal{L}_0(n_j)) & \text{if } i = 0 \\ \beta \cdot \partial_2 d_{\mathcal{R}}(R(\mathit{left}(a_k)), \mathcal{L}_1(n_j)) & \text{if } i = 1 \\ \beta \cdot \partial_2 d_{\mathcal{R}}(R(\mathit{right}(a_k)), \mathcal{L}_2(n_j)) & \text{if } i = 2 \end{cases}$$

and

$$\frac{\partial R_l(a_k)}{\partial \mathcal{L}_i(n_j)(a_k)} = \begin{cases} \partial_j \mathit{rep}_l(\tilde{d}(a_k, \mathcal{N})) \cdot \alpha \cdot \partial_2 d_{\mathcal{W}}(a_k, \mathcal{L}_0(n_j)) & \text{if } i = 0 \\ \partial_j \mathit{rep}_l(\tilde{d}(a_k, \mathcal{N})) \cdot \beta \cdot \partial_2 d_{\mathcal{R}}(R(\mathit{left}(a_k)), \mathcal{L}_1(n_j)) & \text{if } i = 1 \\ \partial_j \mathit{rep}_l(\tilde{d}(a_k, \mathcal{N})) \cdot \beta \cdot \partial_2 d_{\mathcal{R}}(R(\mathit{right}(a_k)), \mathcal{L}_2(n_j)) & \text{if } i = 2 \end{cases}$$

$\partial \mathcal{E}(T) / \partial R_l(a_k)$ can be computed recursively starting from the root to the leaves via backpropagation as follows: $\partial \mathcal{E}(T) / \partial R_l(a_k) = 0$ for the root a_k of T. For other nodes a_k we find

$$\begin{aligned} & \frac{\partial \mathcal{E}(T)}{\partial R_l(a_k)} \\ = \; & \sum_s \frac{\partial f(\tilde{d}(\mathit{parent}(a_k), \mathcal{N}))}{\partial \tilde{d}(\mathit{parent}(a_k), n_s)} \cdot \beta \cdot \frac{\partial d_{\mathcal{R}}(R(a_k), \mathcal{L}_I(n_s))}{\partial R_l(a_k)} \\ & + \sum_s \frac{\partial \mathcal{E}(T)}{\partial R_s(\mathit{parent}(a_k))} \cdot \frac{\partial R_s(\mathit{parent}(a_k))}{\partial R_l(a_k)} \\ = \; & \sum_s \frac{\partial f(\tilde{d}(\mathit{parent}(a_k), \mathcal{N}))}{\partial \tilde{d}(\mathit{parent}(a_k), n_s)} \cdot \beta \cdot \frac{\partial d_{\mathcal{R}}(R(a_k), \mathcal{L}_I(n_s))}{\partial R_l(a_k)} + \\ & \sum_s \frac{\partial \mathcal{E}(T)}{\partial R_s(\mathit{parent}(a_k))} \cdot \sum_u \frac{\partial \mathit{rep}_s(\tilde{d}(\mathit{parent}(a_k), \mathcal{N}))}{\partial \tilde{d}(\mathit{parent}(a_k), n_u)} \cdot \beta \cdot \frac{\partial d_{\mathcal{R}}(R(a_k), \mathcal{L}_I(n_u))}{\partial R_l(a_k)} \end{aligned}$$

where $I \in \{1,2\}$ equals 1 iff a_k is the left subtree of $parent(a_k)$. The complexity of this backpropagation is of order $TR(W+NR)$.

An alternative bottleneck corresponds to the nettoinputs of BPTT and, in the case of RNNs, gives the respective BPTT formula: we use the similarities given as output of $d_{\mathcal{R}}$ and $d_{\mathcal{W}}$ in the computation as bottleneck. Define

$$D_I(a_j,n_i) = \begin{cases} d_{\mathcal{W}}(a_j,\mathcal{L}_0(n_i)), & \text{if } I=0, \\ d_{\mathcal{R}}(R(\mathit{left}(a_j),\mathcal{L}_1(n_i)), & \text{if } I=1, \\ d_{\mathcal{R}}(R(\mathit{right}(a_j)),\mathcal{L}_2(n_i)), & \text{if } I=2 \end{cases}$$

Then we find

$$\frac{\partial\mathcal{E}(T)}{\partial\mathcal{L}_i(n_j)(a_k)} = \frac{\partial\mathcal{E}(T)}{\partial dR_i(a_k,n_j)} \begin{cases} \partial_2 d_{\mathcal{W}}(a_k,\mathcal{L}_0(n_j)), & i=0, \\ \partial_2 d_{\mathcal{R}}(R(\mathit{left}(a_k)),\mathcal{L}_1(n_j)), & i=1, \\ \partial_2 d_{\mathcal{R}}(R(\mathit{right}(a_k)),\mathcal{L}_2(n_j)), & i=2, \end{cases}$$

where the term $\partial\tilde{d}(a_k,n_j)/\partial\mathcal{L}_i(n_j)(a_k)$ is as above and the derivative of $\mathcal{E}(T)$ with respect to $D_i(a_k,n_j)$ can be computed via backpropagation as

$$\frac{\partial\mathcal{E}(T)}{\partial D_0(a_k,n_j)} = \frac{\partial f(\tilde{d}(a_k,\mathcal{N}))}{\partial\tilde{d}(a_k,n_j)}\cdot\alpha$$

and for $i \in \{1,2\}$:

$$\frac{\partial\mathcal{E}(T)}{\partial D_i(a_k,n_j)} = \frac{\partial f(\tilde{d}(a_k,\mathcal{N}))}{\partial\tilde{d}(a_k,n_j)}\cdot\beta + \sum_l \frac{\partial\mathcal{E}(T)}{\partial D_I(\mathit{parent}(a_k),n_l)}\beta\partial_1 d_{\mathcal{R}}(R(a_k),\mathcal{L}_I(n_l))\cdot\partial_j \mathit{rep}(\tilde{d}(a_k,\mathcal{N}))$$

where $I \in \{1,2\}$ is 1 iff a_k is the left child of $parent(a_k)$. This backpropagation is of order $T(W+N^2R)$. Note that for a representation function of the form $\mathit{rep}(x_1,\ldots,x_N) = (\mathit{rep}_1(x_1),\ldots,\mathit{rep}_N(x_N))$ the complexity further reduces because in this case the derivative $\mathit{rep}_i(x_1,\ldots,x_n)$ reduces to a vector with only one nonzero entry. Hence the complexity of the above two methods in this case reduces to $TR(W+N)$ and $T(W+N^2)$, respectively.

Energy function of VQ

The cost function of vector quantization reads in our case as

$$\mathcal{E}_V(T) := \frac{1}{2}\sum_{i=1}^{d}\sum_{j=1}^{N}\chi(T_i,n_j)\tilde{d}(T_i,n_j)$$

where

$$\chi(T_i,n_j) = \begin{cases} 1 & \text{if } \tilde{d}(T_i,n_j) \text{ is minimal} \\ 0 & \text{otherwise} \end{cases}$$

Hence $f(\tilde{d}(T_i,\mathcal{N}))$ has here the form $\frac{1}{2}\sum_j\chi(T_i,n_j)\tilde{d}(T_i,n_j)$. The only term in the above derivatives which is specific for f reads as $\partial f(\tilde{d}(a_k,\mathcal{N}))/\partial\tilde{d}(a_k,n_j)$. We here compute the term for VQ involving in particular the precise derivative of formulas for the borders of the receptive fields where δ-functions are to be used.

Note that

$$\chi(T_i,n_j) = \mathrm{H}\left(\sum_l \mathrm{H}(\tilde{d}(T_i,n_l)-\tilde{d}(T_i,n_j)) - N + 1.5\right)$$

where H is the Heaviside function with

$$H(x) = \begin{cases} 1 & \text{if } x>0 \\ 0 & \text{otherwise} \end{cases}$$

The derivative of H is the function δ which is a symmetric function with $\delta(x) \neq 0 \iff x = 0$. Hence we find

$$\frac{\partial f(\tilde{d}(T_i,\mathcal{N}))}{\partial \tilde{d}(T_i,n_l)} = \frac{1}{2}\sum_j \frac{\partial \chi(T_i,n_j)}{\partial \tilde{d}(T_i,n_l)} \cdot \tilde{d}(T_i,n_j) + \frac{1}{2}\sum_j \chi(T_i,n_j) \cdot \frac{\partial \tilde{d}(T_i,n_j)}{\partial \tilde{d}(T_i,n_l)}$$

The second term yields the standard Hebb factor of VQ, i.e. the contribution $\frac{1}{2}\chi(T_i,n_l)$ to the above general formulas for the derivatives. The first summand equals

$$\frac{1}{2}\sum_{j,k} \delta(\tilde{d}(T_i,n_k) - \tilde{d}(T_i,n_j)) \left(\frac{\partial \tilde{d}(T_i,n_k)}{\partial \tilde{d}(T_i,n_l)} - \frac{\partial \tilde{d}(T_i,n_j)}{\partial \tilde{d}(T_i,n_l)} \right) \cdot$$
$$\delta\left(\textstyle\sum_k \mathrm{H}(\tilde{d}(T_i,n_k) - \tilde{d}(T_i,n_j)) - N + 1.5\right) \tilde{d}(T_i,n_j)$$

Hence it vanishes because δ is symmetric, and the terms involving δ do not equal zero iff the role of $\tilde{d}(T_i,n_k)$ and $\tilde{d}(T_i,n_j)$ can be changed.

Therefore, we obtain the Hebbian update rules from a gradient descent, if we only take the summands (1.1), (2.1), and (3.1) into account and if we ignore all remaining summands (1.2), (1.3), (2.2), (2.3), (3.2), (3.3). Note that these latter summands contain at least one of the terms $\alpha^i\beta$ or α^{i+1} for $i \geq 1$, hence their contribution is small for small β. The precise update rule which we obtain from this gradient approach, disregarding the contributions of substructures, looks like follows if the Euclidian metric is used:

$$\begin{aligned}
\mathcal{L}_0(n_l) &:= \mathcal{L}_0(n_l) + \eta \cdot \alpha \cdot \chi(T_i,n_l)(a - \mathcal{L}_0(n_l)) \\
\mathcal{L}_1(n_l) &:= \mathcal{L}_1(n_l) + \eta \cdot \beta \cdot \chi(T_i,n_l)(R_1 - \mathcal{L}_1(n_l)) \\
\mathcal{L}_2(n_l) &:= \mathcal{L}_2(n_l) + \eta \cdot \beta \cdot \chi(T_i,n_l)(R_2 - \mathcal{L}_2(n_l))
\end{aligned}$$

where a denotes the current label and R_i denotes the representation of the subtrees.

Alternatively, it is possible to transfer the above three proposals of computing the precise gradients to the recursive SOM directly since the formal representation for trees lies in a real vector space and the involved functions are differentiable. Then the component number l of $\partial_l \mathit{rep}_R(x_1,\ldots,x_N)$ yields $-\exp(-x_l)$, all other components are zero.

For SOMSD, the above method cannot be applied directly: the involved function rep_S is not differentiable; moreover, it can be expected that rep_S should be substituted by rep_V as before, which encodes winners in a unary fashion instead of referring to a lattice. Although rep_V is not differentiable, it can be approximated up to any desired degree of accuracy with a differentiable function. We hence approximate using the soft-min function

$$\mathit{rep}_V(x_1,\ldots,x_N) \approx \left(\frac{\exp(-x_1/\gamma)}{\sum_l \exp(-x_l/\gamma)}, \ldots, \frac{\exp(-x_N/\gamma)}{\sum_l \exp(-x_l/\gamma)} \right)$$

where $\gamma > 0$ controls the quality of the approximation. The original function is recovered for $\gamma \to 0$ except for situations where the minimum is not unique. The derivative of component i with respect to x_j yields $\left(\exp(-x_j/\gamma)/\sum_l \exp(-x_l/\gamma)\right)^2/\gamma$ for $i \neq j$ and $\left(\exp(-x_i/\gamma)/\sum_l \exp(-x_l/\gamma)\right)^2/\gamma - \left(\exp(-x_i/\gamma)/\sum_l \exp(-x_l/\gamma)\right)/\gamma$ for $i = j$

Energy function of SOM

Simple vector quantization formalizes that the quantization error should be minimized. Adaptations which try to match the topology of the data and the lattice structure of the neuron require enhanced cost functions such that the degree of topology preservation is taken into account. The self organizing map of Kohonen does not possess a cost function which can be transferred to the continuous case. Hence we adapt a variation proposed in [22] which uses a slightly different notation of the winner. Then a cost function can be found such that a stochastic gradient descent for simple vectors yields a variation of Hebbian learning for SOM with a slightly different notation of the winner. This cost function when adapted to structured data has the form

$$\mathcal{E}_S(T) = \frac{1}{2}\sum_{i=1}^{T}\sum_{j=1}^{N} \chi_S(T_i,n_j) \sum_{k=1}^{N} \eta(nh(n_j,n_k))\tilde{d}(T_i,n_k)$$

where $nh : \mathcal{N} \times \mathcal{N} \to \mathbb{R}$ describes the neighborhood function of the lattice and $\eta(x) = \eta_0 \exp(-x/\sigma^2)$ with $\eta_0 > 0, \sigma > 0$ provides a scaling factor for the update of neighbors.

$$\chi_S(T_i, n_j) = \begin{cases} 1 & \text{if } \sum_{k=1}^N \eta(nh(n_j, n_k))\tilde{d}(T_i, n_k)^2 \text{ is minimal} \\ 0 & \text{otherwise} \end{cases}$$

determines the winner. Here the winner is the neuron with optimum weighted distance with respect to the neighborhood. I.e. the winner is not the closest neuron but the neuron with minimum averaged distance with respect to the neighborhood. Given a tree T_i, the above cost function induces the function f of the form

$$\frac{1}{2} \sum_j \chi_S(T_i, n_j) \sum_k \eta(nh(n_j, n_k))\tilde{d}(T_i, n_k)$$

The derivative with respect to $\tilde{d}(T_i, n_l)$ we are interested in in order to apply the above general formulas can be computed as

$$\frac{1}{2} \sum_j \frac{\partial \chi_S(T_i, n_j)}{\partial \tilde{d}(T_i, n_l)} \sum_k \eta(nh(n_j, n_k))\tilde{d}(T_i, n_k) + \frac{1}{2} \sum_j \chi_S(T_i, n_j) \sum_k \eta(nh(n_j, n_k)) \frac{\partial \tilde{d}(T_i, n_k)}{\partial \tilde{d}(T_i, n_l)}$$

The second term yields the usual SOM-update with slightly different notion of the winner as explained above. The contribution to the above formulas is $\frac{1}{2} \sum_j \chi_S(T_i, n_j)\eta(nh(n_j, n_l))$. The first term vanishes, as can be seen as follows: we again use δ-functions and the identity

$$\chi_S(T_i, n_j) = \mathrm{H}\left(\sum_l \mathrm{H}\left(\sum_k \eta(nh(n_l, n_k))\tilde{d}(T_i, n_k) - \sum_k \eta(nh(n_j, n_k))\tilde{d}(T_i, n_k) \right) - N + 1.5 \right)$$

Hence the first sum equals

$$\begin{aligned} &\frac{1}{2} \sum_{j,o} \delta\left(\sum_s \mathrm{H}\left(\sum_t \eta(nh(n_s, n_t))\tilde{d}(T_i, n_t) - \sum_t \eta(nh(n_j, n_t))\tilde{d}(T_i, n_t) \right) - N + 1.5 \right) \cdot \\ &\delta\left(\sum_t \eta(nh(n_o, n_t))\tilde{d}(T_i, n_t) - \sum_t \eta(nh(n_j, n_t))\tilde{d}(T_i, n_t) \right) \cdot \\ &\qquad\qquad \sum_k \tilde{d}(T_i, n_k)\eta(nh(n_j, n_k)) \left(\eta(nh(n_o, n_l)) - \eta(nh(n_j, n_l)) \right) \end{aligned}$$

This vanishes because δ is symmetric and the above δ-terms are nonvanishing only if the weighted distances of n_o and n_j can be substituted.

Therefore, we obtain the slightly altered Hebbian update rules from a gradient descent, if we again disregard the summands $(1.2), (1.3), (2.2), (2.3), (3.2), (3.3)$:

$$\begin{aligned} \mathcal{L}_0(n_l) &:= \mathcal{L}_0(n_l) + \eta \cdot \alpha \cdot \textstyle\sum_k \chi_S(T_i, n_k)\eta(nh(n_k, n_l))(a - \mathcal{L}_0(n_l)) \\ \mathcal{L}_1(n_l) &:= \mathcal{L}_1(n_l) + \eta \cdot \beta \cdot \textstyle\sum_k \chi_S(T_i, n_k)\eta(nh(n_k, n_l))(R_1 - \mathcal{L}_1(n_l)) \\ \mathcal{L}_2(n_l) &:= \mathcal{L}_2(n_l) + \eta \cdot \beta \cdot \textstyle\sum_k \chi_S(T_i, n_k)\eta(nh(n_k, n_l))(R_2 - \mathcal{L}_2(n_l)) \end{aligned}$$

where as before a denotes the current label and R_i denotes the representation of the subtrees.

If we would like to perform an exact gradient descent, the specific adaptations of this method for the recursive SOM and SOMSD follow the same line as in the previous case. Note that we could again use the alternatives of forward or backward propagation as described above. Again the function rep_S of SOMSD is not differentiable. However, we can approximate the function *rep* up to every desired degree. Denote the indices of the neurons by $\vec{\iota}(1), \ldots, \vec{\iota}(N)$. The representation function which computes the index of the minimum input can be approximated by the soft-min function

$$rep_S(x_1, \ldots, x_n) \approx \sum_{i=1}^N \vec{\iota}(i) \cdot \frac{\exp(-x_i/\gamma)}{\sum_l \exp(-x_l/\gamma)}$$

where $\gamma > 0$ controls the quality of the approximation. This approximation is differentiable. The original function is recovered for $\gamma \to 0$ except for situations where the minimum is not unique. Note that tie breaks are here solved by an interpolation between the indices of the winners instead of the choice of the first winner. The derivative reads as $\partial_l rep(x_1, \ldots, x_N) =$

$$\frac{\exp(-x_l/\gamma)}{\gamma\left(\sum_i \exp(-x_i/\gamma)\right)^2}\left(\sum_{i=1}^{N} \vec{\iota}(i)\exp(-x_i/\gamma) - \vec{\iota}(l)\sum_{i=1}^{N}\exp(-x_i/\gamma)\right)$$

Note that for SOMSD the lattice of neurons has to be chosen a priori. It can be expected that a two-dimensional regular grid is appropriate only for very restricted data sets due to the more complex data structure. The experiments of [18] show that different structures are stored in different clusters in a regular grid map. The lattice can combine different structures only partially due to a combinatorial explosion of possible neighbors: already for sequences over a finite alphabet, the number of ways in which a sequence can be expanded by i terms constitutes an exponential number in i. An even more rapid increase can be found for tree structures, of course. One could expect that lattices which mirror this behavior of an exponentially increasing neighborhood like the hyperbolic SOM [37] could be a good choice for huge data sets which cover a variety of different structures. Note that topologocial defects of the lattice have an effect on the recursive processing. Hence one could again alternatively treat the indices of neurons as orthogonal directions, i.e. choose the representation rep_V for SOMSD even if a lattice is available.

Energy function of NG

The neural gas algorithm constitutes another alternative where the lattice structure is not given a priori but emerges according to the data structure. Given a data point, those neurons are considered as neighbors which have similar places in the ranking according to the distance from the data point. The cost function of NG adapted to structured data has the following form:

$$\mathcal{E}_N(T) = \frac{1}{2}\sum_{i=1}^{d}\sum_{j=1}^{N}\eta(rk(i,j))\tilde{d}(T_i, n_j)\,.$$

$rk(i,j)$ equals the number of neurons k such that $\tilde{d}(T_i, n_k) < \tilde{d}(T_i, n_j)$. Again $\eta(x) = \eta_0 \exp(-x/\sigma^2)$ is a scaling factor. If we put this cost function in the above general form, we find for the function f

$$f(\tilde{d}(T_i, \mathcal{N})) = \frac{1}{2}\sum_j \eta(rk(i,j))\tilde{d}(T_i, n_j)$$

The derivative with respect to $\tilde{d}(T_i, n_l)$ yields

$$\frac{1}{2}\sum_j \frac{\partial\eta(rk(i,j))}{\partial\tilde{d}(T_i, n_l)}\tilde{d}(T_i, n_j) + \frac{1}{2}\sum_j \eta(rk(i,j))\frac{\partial\tilde{d}(T_i, n_j)}{\partial\tilde{d}(T_i, n_l)}$$

where the second term yields the usual update rule for NG, i.e. the contribution $\frac{1}{2}\eta(rk(i,l))$ to the above formulas. The first term vanishes, which can again explicitely be computed using the identity

$$rk(i,j) = \sum_k H(\tilde{d}(T_i, n_j) - \tilde{d}(T_i, n_k))$$

Then the first term becomes

$$\frac{1}{2}\sum_{j,k}\frac{\partial\eta(rk(i,j))}{\partial rk(i,j)}\delta(\tilde{d}(T_i, n_j) - \tilde{d}(T_i, n_k))\tilde{d}(T_i, n_j)\left(\frac{\partial\tilde{d}(T_i, n_j)}{\partial\tilde{d}(T_i, n_l)} - \frac{\partial\tilde{d}(T_i, n_k)}{\partial\tilde{d}(T_i, n_l)}\right)$$

This vanishes due to the properties of δ.

The Hebbian update rules hence result from a gradient descent, if we disregard the summands (1.2), (1.3), (2.2), (2.3), (3.2), (3.3):

$$\begin{aligned}\mathcal{L}_0(n_l) &:= \mathcal{L}_0(n_l) + \eta \cdot \alpha \cdot \eta(\mathit{rk}(i,l))(a - \mathcal{L}_0(n_l))\\ \mathcal{L}_1(n_l) &:= \mathcal{L}_1(n_l) + \eta \cdot \beta \cdot \eta(\mathit{rk}(i,l))(R_1 - \mathcal{L}_1(n_l))\\ \mathcal{L}_2(n_l) &:= \mathcal{L}_2(n_l) + \eta \cdot \beta \cdot \eta(\mathit{rk}(i,l))(R_2 - \mathcal{L}_2(n_l))\end{aligned}$$

where as before a denotes the current label and R_i denotes the representation of the subtrees.

As before, adaptation for SOMSD is necessary if the precise gradients should be computed instead of this approximation. Since no lattice is given a priori, the space of indices which constitutes $\mathcal{R}$ in the original approach consists of points which do not possess any prior metrical information. The distance between indices should be dynamically computed during the training algorithm according to the ranking with respect to the respective data point. Hence we again choose the representation function

$$\mathit{rep}_N(x_1,\ldots,x_n) = (\mathit{rk}(x_1),\ldots,\mathit{rk}(x_N)) = \left(\sum_{j=1}^{N} \mathrm{H}(x_1 - x_j),\ldots,\sum_{j=1}^{N} \mathrm{H}(x_N - x_j)\right)$$

where $\mathit{rk}(x_i)$ denotes the rank of x_i if the values $x_1, \ldots, x_N$ are ordered according to their size. Since this function is not differentiable, we have to approximate by a differentiable function. We can substitute H by a sigmoidal approximation $\mathrm{sgd}_\gamma(x) := \mathrm{sgd}(x/\gamma) \to \mathrm{H}(x)$ for $\gamma > 0$, $\gamma \to 0$ and $x \neq 0$ where $\mathrm{sgd}(x) = (1+\exp(-x))^{-1}$. Then we can compute the component j of the derivative of this approximation as

$$(\partial_i \mathit{rep}_s(x_1,\ldots,x_n))_j = \begin{cases} \sum_{l\neq i} \mathrm{sgd}_\gamma(x_i - x_l)(1 - \mathrm{sgd}_\gamma(x_i - x_l))/\gamma & \text{if } i = j\\ -\mathrm{sgd}_\gamma(x_j - x_i)(1 - \mathrm{sgd}_\gamma(x_j - x_i))/\gamma & \text{otherwise}\end{cases}$$

Note that the same effort is laid on each position of the ranking in this formulation. It can be expected that the position of winning units is more important than the rank of neurons which are far away. Hence one could modify the formal representation of trees such that that winning neurons are ranked higher. One possibility is to include an exponential weighting $\mathit{rep}(x_1,\ldots,x_n) = (\exp(-\mathit{rk}_k(x_1)),\ldots,\exp(-\mathit{rk}_k(x_N)))$. This has the effect that deviations for neurons which are close to the actual tree are ranked higher than neurons which are further away. Alternatively, we could use a simple truncation $\mathit{rep}(x_1,\ldots,x_n) = (\mathit{rk}_k(x_1),\ldots,\mathit{rk}_k(x_N))$ where

$$\mathit{rk}_k(x_i) = \begin{cases} \mathit{rk}(x_i) & \text{if } \mathit{rk}(x_i) < k\\ k & \text{otherwise}\end{cases}$$

for some $k \in \{1,\ldots,N\}$. This has the effect that the precise position of only the closest k neurons is of interest, the others are identified in the formal representation.

6 Properties of *rep*

The function *rep* which is used in order to obtain a formal representation of the trees is obviously a crucial part of the dynamics. It has to be made sure that the important information of the distributed representation of a processed tree is preserved through this function *rep*. At the same time, *rep* should be efficiently computable and noise tolerant. We want to discuss several properties which seem of importance concerning the formal representations, i.e. *rep*, $\mathcal{R}$, and $d_\mathcal{R}$. We start with a simple observation:

- The function *rep* should map to a bounded domain, i.e. it should be possible to uniformly bound the value $d_\mathcal{R}(r_1, r_2)$ for all representations r_1 and r_2 which might occur during a recursive computation or as a component of a weight.

Note that unless in the standard SOM, this property is not automatically fulfilled if all labels of trees come from a bounded domain, since distances may blow up through the recursive processing. Assumed, for example, we dropped the exponential function when computing the formal representation *rep* for the recursive SOM, we would obtain a blow up of the values during the recursive computation: the distances

could accumulate during the recursive computation such that no universal bound could be found unless the height of trees to be processed was limited. Both, for SOMSD and the recursive SOM this behavior is prohibited due to the compactness of the respective domain $\mathcal{R}$.
Another obvious demand on *rep* is the following:

- *rep* should be noise tolerant, i.e. if $x \in \mathbb{R}^N$ is disrupted by noise to $x + \eta$, the distance between $d_\mathcal{R}(\textit{rep}(x), r)$ and $d_\mathcal{R}(\textit{rep}(x + \eta), r)$ should be small for all possible $r \in \mathcal{R}$.

This point is of particular importance, since small errors of the computation e.g. due to noisy inputs may blow up easily at recursive processing. Hence some focus on important parts of the distributed representation of trees is to be introduced when computing *rep*. Assumed that important information lies in the winner, then clearly SOMSD precisely extracts this point. The recursive SOM amplifies large values and smoothes vanishing terms via the exponential function. Another fact makes insensitivity with respect to noise a crucial part of the design: The dimensionality of the input for *rep* is commonly high dimensional since it coincides with the number of neurons of the map. The formal representations may be high dimensional, too, as in the case of the recursive SOM. Hence errors may accumulate simply due to the curse of dimensionality.

One can explicitely compute the sensitivity with respect to noise for concrete implementations of *rep*: Assume $x = (x_1, \ldots, x_n) \in \mathbb{R}^N$ is an activation, r a formal representation, and $\eta = (\eta_1, \ldots, \eta_N) \subset \mathbb{R}^N$ a vector of i.i.d. noise with zero mean and finite variance. We are interested in the term

$$E(|d_\mathcal{R}(\textit{rep}(x), r) - d_\mathcal{R}(\textit{rep}(x + \eta), r)|) \quad (*)$$

where E denotes expectation with respect to η. If *rep* is simply the identity and $d_\mathcal{R}$ the Euclidian distance, one obtains

$$\begin{aligned}
(*) &= E(|\sum(x_i - r_i)^2 - \sum(x_i + \eta_i - r_i)^2|) \\
&= E(|\sum 2\eta_i(x_i - r_i) + \eta_i^2|) \\
&\geq \sum 2(x_i - r_i)E(\eta_i) + \sum E(\eta_i^2) \\
&= N \cdot E(\eta_i^2)
\end{aligned}$$

hence this representation clearly suffers from the curse of dimensionality. We obtain for the recursive SOM

$$\begin{aligned}
(*) &= E(|\sum(\exp(-x_i) - r_i)^2 - \sum(\exp(-x_i - \eta_i) - r_i)^2|) \\
&= E(|\sum \exp(-2x_i)(1 - \exp(-2\eta_i)) - 2\exp(-x_i)r_i(1 - \exp(-\eta_i))|) \\
&\leq \sum \exp(-2x_i)|1 - E(\exp(-2\eta_i))| + 2\exp(-x_i)|r_i| \cdot |1 - E(\exp(-\eta_i))| \,.
\end{aligned}$$

The term $E(\exp(-\eta_i))$ approaches 1 if the support of the noise is concentrated on 0. Usually, only few terms x_i are close to 0 whereas the majority will be large corresponding to a unique winner for each tree of the network. Hence the curse of dimensionality is softened through the factors $\exp(-x_i)$, but can possibly still be observed for high dimensional formal representations.

For SOMSD with *rep* as the index of the winner, $(*)$ obviously only yields a nonzero term, if the noise η changes the winner. For data where the winner is distinct, i.e. its activation considerably smaller than the remaining entries, this probability is small for noise with limited support or Gaussian noise. However, even if the winner changes, it can be expected that the new winner lies in a similar region, since the neurons which are neighbored to the winner tend to have small activations as well. Hence this representation is robust to noise. Moreover, the accumulation of errors through noise in recursive computation steps is obviously avoided in this approach due to the discrete values in $\mathcal{R}$.

A further point of discussion concerns the representational capability of $\mathcal{R}$. Note that the number of neurons is determined corresponding to the number of different data points we would like to recognize, or alternatively the granularity of the representation which we would like to obtain. However, the computation of each neuron goes through the bottleneck of $\mathcal{R}$ or the image of *rep*, respectively, i.e. the activation of different trees is internally represented by values in $\mathcal{R}$ computed with *rep*. If two activation profiles cannot be distinguished by such values then the corresponding trees are identified by the network. Hence we demand the following:

- If two trees should not be identified through a computation, then their respective representation via *rep* in $\mathcal{R}$ has to be different.

The cardinality of $\mathcal{R}$ and the image of *rep* is infinite for the recursive SOM. For SOMSD *rep* only maps to the number of indices, hence SOMSD can differentiate between all trees which have different winners in the recursive computation. One can explicitely lower bound the cardinality of the image of *rep*, assumed a network should recognize all trees with binary labels of height at most T. We consider here an elementary definition of the notation of recognition:

Definition 6.1 *A neuron n_i in a network* recognizes *a tree t iff $\tilde{d}(t, n_i) > \tilde{d}(t, n_j)$ for all $n_j \neq n_i$. A network* recognizes *a set of trees $\mathcal{T}$, if for each tree $t \in \mathcal{T}$ a* different *neuron $n(t)$ can be found which recognizes t.*

Note that the choice of the symbol '$>$" is arbitrary and should depend on the respective setting. I.e. it should take into account whether $d_{\mathcal{W}}$ and $d_{\mathcal{R}}$ are metrics or dot products. We could substitute the symbol '$>$' in the above definition by '$<$'. Then obviously SOMSD could recognize every finite set of trees. Moreover, a number of neurons which equals the number of all different subtrees of trees to be recognized would be sufficient. Obviously, trees with real-valued labels can only be distinguished if an infinite number of internal representations is available. However, for finite labels (e.g. a sufficiently fine-grained representation of real labels) SOMSD uses almost the minimum cardinality for the image of *rep* which will do for this purpose. We here assume that labels come from a binary set $\{0, 1\}$.

Theorem 6.2 *Assume e network recognizes all trees with labels in $\{0, 1\}$ up to height T. Then the cardinality of the image of rep is lower bounded by the number of trees with labels in $\{0, 1\}$ of height at most $T - 1$.*

Proof. Denote the number of trees with elements in $\{0, 1\}$ of height at most $T - 1$ by h. Assume a given SOMSD can recognize all such trees. Assume the cardinality of the image of *rep* is at most $h-1$. Then two trees with labels in $\{0, 1\}$ of height at most $T-1$ exist, say t and t' such that $r = rep(\tilde{d}(t, n_1), \ldots, \tilde{d}(t, n_N))$ and $r' = rep(\tilde{d}(t', n_1), \ldots, \tilde{d}(t', n_N))$ are identical. Hence the activity profile of t and t' cannot be distinguished in further recursive computations. In particular, a tree which contains t as a proper subtree cannot be distinguished from the same tree where t is substituted by t'. □

Note that SOMSD does not need the indices for winners for trees with maximum height, since their activity profile is not used in further recursive computations.

We can expect from the standard use of self organizing networks that important information of an activity profile of a tree is the location of the winner. SOMSD uses precisely this information, whereas the recursive SOM has access to all activations including the winner. The TKM obviously does not use this information and is therefore restricted concerning its capacity as we will see later. Since the respective winners are usually spreaded among the network e.g. according to the different tree structures, the network has to have access to the activation of all neurons in recursive steps. Stated informally, we demand the following:

- *rep* should be a global function which has access to (and effectively uses) information of the activation of all neurons.

One could alternatively try to design networks such that the recursive similarity of one neuron does only depend on the activity profile of neighbored neurons. In such a local setting, the information about the global winner may be lost. Hence it can be expected, that such a network has only limited capacities. We will formalize the notion of locality precisely in Section 7 and investigate the necessary degree of globality of a network for the recognition of a set of trees in Section 7.

Commonly, information in self organizing networks is detected through the metric structure of the data. In our case, $d_{\mathcal{R}}$ and $d_{\mathcal{W}}$ will commonly be metrics or scalar products. They are used for local comparisons of labels or formal representations, respectively. The computation for tree structures emerges through the recursive processing using $d_{\mathcal{W}}$, $d_{\mathcal{R}}$, and *rep*. *rep* should take this fact into account. I.e. it should map similar trees to similar formal representations such that further use of $d_{\mathcal{R}}$ is justified. One problem consists of course in the fact that we have no notion for the similarity of trees. We did not fix a metric on trees itself;

rather it emerges from the recursive processing and recursive comparisons of labels and representations using $d_{\mathcal{W}}$ and $d_{\mathcal{R}}$. We will discuss this topic more closely in the context of topology preservation in Section 8. We can expect that important information lies in the winners for a tree. Trees are considered as similar in all standard ways of using the trained SOM if they have the same or similar winners. Hence at least the metrical information concerning the winner should be preserved through the function *rep* for appropriate processing.

- *rep* should map two vectors x and $x' \in \mathbb{R}^N$ which winners lie at similar positions to formal representations which are similar with respect to $d_{\mathcal{R}}$. If x and x' have winners at very different positions, their formal representations should be less similar.

Actually, the lattice which is induced a posteriori in the neural gas algorithm will be appropriate iff the position of the first two winners is respected through *rep* as we will discuss in Section 8.

7 Globality of processing

Note that we did not adapt the learning algorithms based on an cost functions for TKM since TKM is appropriate for processing only very limited sets of trees due to the following reason: Denote by $st_i(t)$ the sequence of labels which can be observed at the ith level in the tree t. Formally, $st_i(\xi)$ is the empty sequence for all i, $st_0(a(t_1,t_2))$ is the sequence $[a]$, and $st_i(a,t_1,t_2) = st_{i-1}(t_1) \bullet st_{i-1}(t_2)$ for $i > 0$ where '$\bullet$' denotes the concatenation of sequences. For TKM, we find that $\tilde{d}(t_1,n)$ and $\tilde{d}(t_2,n)$ are identical for trees t_1 and t_2 such that the labels at each level of t_1 and t_2 are permutations of each other, i.e. $st_i(t)$ and $st_i(t')$ are permutations for all $i \geq 0$. Hence TKM can only differentiate between trees of mutually different height and trees which have mutually different labels at each level. It cannot differ between trees which have a different structure but the same labels at each level such as $a(b,c)$ and $a(c,b)$ or $a(b,c(d,e))$ and $a(b(d,e),c)$.

Obviously, the dynamics of TKM is in some sense extremely local. Each neuron restricts its computation to the activation of the neuron itself during recursive processing. It would be interesting to see which degree of locality or globality is sufficient for adequate processing of general trees. In contrast to TKM, SOMSD and recursive SOM are global since their processing depends on the activation of the global winner or all neurons, respectively. Assume a fixed network is given. We define the transition function of neuron n_i which is used for the recursive computation of $\tilde{d}(\,\cdot\,,n_i)$ as

$$\begin{aligned} &f_{n_i} : \mathcal{W} \times \mathbb{R}^N \times \mathbb{R}^N \to \mathbb{R}, \\ &(a,r_1,r_2) \mapsto \alpha\, d_{\mathcal{W}}(a,\mathcal{L}_0(n_i)) + \beta\, d_{\mathcal{R}}(rep(r_1),\mathcal{L}_1(n_i)) + \beta\, d_{\mathcal{R}}(rep(r_2),\mathcal{L}_2(n_i))\,. \end{aligned}$$

Definition 7.1 *A network is* local with degree $k > 0$ *if for each neuron n_i there exist k indices $i_1, \ldots, i_k$ such that f_{n_i} depends on the coefficients $i_1, \ldots i_k$ of r_1 and r_2 only. That means, for all $a \in \mathcal{W}$, r_1, r_2, r_1', $r_2' \in \mathbb{R}^N$ such that the coefficients $(r_1)_{i_1} = (r_1')_{i_1}, \ldots, (r_1)_{i_k} = (r_1')_{i_k}$ and $(r_2)_{i_1} = (r_2')_{i_1}, \ldots, (r_2)_{i_k} = (r_2')_{i_k}$ coincide, we find $f_{n_i}(a,r_1,r_2) = f_{n_i}(a,r_1',r_2')$.*

Now we are interested to see whether a network which is local up some degree k can recognize a set of structures adequately. Of course, the notion of adequate processing depends on the respective application. We consider here an elementary property which should be met by every adequate dynamics and simply refer to the definition of recognition in Definition 6.1: Each tree to be recognized should have a different winner. As already mentioned, SOMSD can recognize every finite set of trees. The function *rep* in SOMSD computes the winner which is a global function taking information into account which only exists globally.

Degrees of locality can be investigated for the recursive SOM and variations. For simplicity, we restrict in the following to binary trees with labels in $\{-1,1\}$ up to height T. Define $\mathcal{T}_T$ as the set of all nonempty binary trees with binary labels up to height T. Then we find the following result:

Theorem 7.2 *Assume that $\mathcal{R} = \mathbb{R}^N \cup \{r_\xi\}$. Assume that r_ξ is a specified formal representation for the empty tree such that $d_{\mathcal{R}}(r,r_\xi) = 0$ for all $r \neq r_\xi$ and $d_{\mathcal{R}}(r_\xi,r_\xi) > 0$. Assume $d_{\mathcal{R}}$ and $d_{\mathcal{W}}$ are given by the standard dot product. Assume rep decomposes into $rep(x_1,\ldots,x_N) \to (rep_1(x_1),\ldots,rep_1(x_N))$ such that $rep_1 : \mathbb{R} \to \mathbb{R}$ is strictly monotonically increasing. Then a network of size $|\mathcal{T}_T|$ exists which is local with degree 1 which recognizes $\mathcal{T}_T$.*

Proof. Consider a network with $|\mathcal{T}_T|$ neurons. Fix a bijective mapping $\mathcal{T}_T \to \mathcal{N}$, $t \mapsto n(t)$. Assign the following weight to node $n(t)$: If $t = a(t_1, t_2)$, then $\mathcal{L}_0(t) = a$,

$$\mathcal{L}_1(t) = \begin{cases} 1_{n(t_1)} & \text{if } t_1 \neq \xi \\ r_\xi & \text{if } t_1 = \xi \end{cases}$$

where $1_{n(t_1)}$ is the unit vector which is 1 at the position of $n(t_1)$ and 0 at all other positions, and

$$L_2(t) = \begin{cases} 1_{n(t_2)} & \text{if } t_2 \neq \xi \\ r_\xi & \text{if } t_2 = \xi \end{cases}$$

With this definition the network is obviously local with degree 2. The maximum recursive similarity for neuron $n(a(t_1, t_2))$ can be observed if the tree $a(t_1, t_2)$ is processed which can be seen by induction: The tree consisting of only one node labeled -1 or 1, respectively, obviously has maximum recursive similarity for the node weighted with $(-1, r_\xi, r_\xi)$ or $(1, r_\xi, r_\xi)$, respectively. If $a(t_1, t_2)$ is processed where t_1 and t_2 are nonempty trees, the nodes $n(t_1)$ and $n(t_2)$ will give maximum recursive similarity for t_1 and t_2, respectively, by induction. This property is preserved if *rep* is applied due to our assumptions on *rep*. Hence the node $n(a(t_1, t_2))$ yields maximum recursive similarity for $a(t_1, t_2)$ because the three dot products involved in the computation are maximum for a, and the representation of t_1 and t_2, respectively. If t_1 or t_2 is the empty tree, we substitute the corresponding dot product by the comparison with r_ξ. □

Hence local processing in approaches like the recursive SOM can recognize all trees with an appropriate number of neurons and an appropriate connection structure for the local information flow. However, the connection structure in this local processing has to be precisely fitted to the structure of the trees. In particular, we have to know beforehand which nodes are responsible for which trees in order to connect those nodes to the nodes responsible for their corresponding subtrees. Moreover, if the maximum height T is not fixed, the structure cannot be embedded in any rectangular lattice of finite dimensionality. It would be more natural to consider locality with respect to a given fixed lattice structure. E.g. given a two-dimensional lattice as in the standard SOM, it would be natural to allow that a neuron only uses the recursive similarities of all neighbors in the lattice up to a fixed degree k.

Definition 7.3 *Assume a network together with a lattice structure is given. Assume the lattice structure defines a neighborhood function nh* : $\mathcal{N} \times \mathcal{N} \to \mathbb{N}$. *A network is* local with degree $k > 0$ with respect to a given neighborhood *nh if the following holds: for each neuron n_i the function f_{n_i} depends on the coefficients of r_1 and r_2 only which correspond to neurons which are neighbors of degree $\leq k$. That means, for all $a \in \mathcal{W}$, r_1, r_2, r'_1, $r'_2 \in \mathbb{R}^N$ such that the coefficients $(r_1)_j = (r'_1)_j$ and $(r_2)_j = (r'_2)_j$ coincide for all j such that $nh(n_i, n_j) \leq k$, we find $f_{n_i}(a, r_1, r_2) = f_{n_i}(a, r'_1, r'_2)$.*

Speaking in these terms, the TKM is local with degree 0 with respect to every neighborhood given by some lattice. We would expect that this locality severely restricts the information flow in the network. I.e. important tasks cannot be solved with local networks. However, there is still a technical difficulty if we would like to formally prove this conjecture: the recursive similarities are contained in the real numbers with possibly infinite precision. Hence an arbitrary amount of information could theoretically be stored in each single value of the computation and possibly make information flow over the network superfluous. Indeed, if we allowed a slightly more complex computation of recursive similarities, networks which are local with degree 0 with respect to any neighborhood could be constructed which recognize the above set $\mathcal{T}_T$ of binary trees with binary labels up to height T. The idea behind is that encoding tricks can be applied such that the already processed tree is precisely stored in the value of the recursive similarity computed by each neuron.

More precisely, we assume, that $\tilde{d}$ may yield three-dimensional outputs and the transition may use correlations of the inputs as already discussed beforehand in the context of RSOM and EKFM. We assign to each tree t in $\mathcal{T}_T$ the prefix representation p_t encoded as a real number: $p_\xi = 0$, $p_{a(t_1,t_2)} = (a+2) \bullet p_{t_1} \bullet p_{t_2}$, '$\bullet$' denoting concatenation. Assume that l_t denotes the length of this representation. Assume *rep* yields the identity, i.e. formal representations are contained in $(\mathbb{R}^3)^N$. Assume the formal representation for the empty tree is $(0, 0.p_\xi, (0.1)^{l_\xi}) = (0, 0, 0.1)$ for each of the N three-dimensional components. Assume the following weight is assigned to the neuron $n(a, t_1, t_2)$ which should recognize the tree $a(t_1, t_2)$:

$(a,(0,\ldots,(0,0.p_{t_1},0),\ldots,0),(0,\ldots,(0,0.p_{t_2},0),\ldots,0))$, the nonempty entries for the formal representation of the two subtrees being at the position of the neuron. Then the recursive computation for this neuron n_i

$$\begin{aligned} f_{n_i}(a,(\ldots,(r_{11},r_{12},r_{13}),\ldots),(\ldots,(r_{21},r_{22},r_{23}),\ldots)) = \\ (3-(|a-a'|+|0.p_{t_1}-r_{12}|+|0.p_{t_2}-r_{22}|), \\ 0.1\cdot(a+2)+0.1\cdot r_{12}+0.1\cdot r_{13}\cdot r_{22}, \\ 0.1\cdot r_{13}\cdot r_{23}) \end{aligned}$$

where due to the locality we only consider the position i of the formal representations, computes the following: In the first position it yields a value ≤ 3 with the value 3 iff the node is responsible for the actual tree t which is processed; the second component yields the prefix representation $0.p_t$ of the actual tree; the third component yields $(0.1)^{l_t}$. Hence this network recognizes all trees in $\mathcal{T}_T$ if the winner in each step is defined by the first components of the recursive similarity.

The above computation with respect to *rep*, $d_{\mathcal{W}}$, and $d_{\mathcal{R}}$ is simple. Moreover, we could avoid to use three-dimensional outputs of $\tilde{d}$ if we allowed a more complicated computation of the transition function of neurons. However, it is not not realistic in the above argumentation to assume that precise values can be stored in the recursive similarities, i.e. $\tilde{d}$ is computed with perfect reliability and infinite precision. In order to take the finite precision of each realistic computation into account, we can assume that the outputs of $\tilde{d}$ are contained in a finite set $\{1,\ldots,D\}$ instead of the whole set $\mathbb{R}$. Then locality with respect to a rectangular lattice prohibits the possibility of recognizing $\mathcal{T}_T$ if T is large:

Theorem 7.4 *A d-dimensional rectangular grid structure for a network refers to the fact that there exist $N_1, \ldots, N_d$ such that the indices of the neurons coincide with $\{1,\ldots,N_1\}\times\ldots\times\{1,\ldots,N_d\}$. The neighborhood is given by $nh(n_{(i_1,\ldots,i_d)},n_{(j_1,\ldots,j_d)}) = |i_1-j_1|+\ldots+|i_d-j_d|$. Assume the dimensionality of the grid d is fixed. Assume the degree k of locality with respect to a neighborhood induced by a d-dimensional rectangular grid is fixed. Assume the computation is performed with finite precision, i.e. $\tilde{d}$ yields values in a finite set $\{1,\ldots,D\}$ where D is fixed. Then we can find a number T such that there cannot exist a network with d-dimensional rectangular grid structure which is local with degree k with respect to this neighborhood which recognizes $\mathcal{T}_T$.*

Proof. We can assume w.l.o.g. that $r_\xi = rep(1,\ldots,1)$. Consider trees of height T which have the maximum number of nodes. These trees have 2^{T-1} leaves, hence there exist at least $2^{2^{T-1}}$ mutually different trees of height T with the same structure and binary labels. Denote this set of trees by $\mathcal{T}_T^r$. If a network recognizes $\mathcal{T}_T^r$, there exists a different neuron $n_i(t)$ for every $t\in\mathcal{T}_T^r$ which has maximum recursive similarity for t. Hence at least $2^{2^{T-1}}$ neurons n_i exist which yield mutually different functions $\tilde{d}(\cdot,n_i)$ if restricted to inputs in $\mathcal{T}_T^r$.

Consider a transition function assigned to a neuron $f_{n_i}:\{-1,1\}\times\mathbb{R}^N\times\mathbb{R}^N\to\mathbb{R}$. This function has the form

$$(a,r_1,r_2)\mapsto \alpha\, d_{\mathcal{W}}(a,\mathcal{L}_0(n_i))+\beta\, d_{\mathcal{R}}(rep(r_1),\mathcal{L}_1(n_i))+\beta\, d_{\mathcal{R}}(rep(r_2),\mathcal{L}_2(n_i))$$

Since the network is local with degree k with respect to the neighborhood and the computation is done with finite precision, we can assume that the domain and codomain of f_{n_i} are in fact restricted:

$$f_{n_i}:\{-1,1\}\times\{1,\ldots,D\}^{(2k+1)^d}\times\{1,\ldots,D\}^{(2k+1)^d}\to\{1,\ldots,D\}.$$

For combinatorial reasons, there exist at most $D^{2\cdot D^{(2k+1)^d}\cdot D^{(2k+1)^d}} =: D_f$ different functions of this form. D_f is a constant since we fixed the parameters D, k, and d. The function $\tilde{d}(\cdot,n_i)$ which maps a tree from the set $\mathcal{T}_T^r$ to the recursive similarity with respect to the neuron n_i is based on the transition functions f_{n_j} of several neurons n_j. In the last step of the recursive computation, only f_{n_i} itself is taken into account; in the last but one step, all k-neighbors of n_i may contribute; in the last but two step, all k-neighbors of these latter neurons may contribute, too, which corresponds to neighbors of degree $2k$ of n_i due to the lattice structure. Hence for the whole computation, all neighbors up to degree Tk of n_i may contribute to the function. That implies, that the choice of the transition functions for neurons in a neighborhood of degree Tk uniquely determines the function $\tilde{d}(\cdot,n_i)$. Note that the way in which the transition functions are combined is identical for all inputs in $\mathcal{T}_T^r$ because all trees in this set have the same structure. The number of different possible choices for the transition function of all neurons in a neighborhood of size Tk is $D_f^{(2Tk+1)^d}$. This is smaller than $2^{2^{T-1}}$ for large T, hence the network cannot differentiate between all trees in $\mathcal{T}_T^r$ for large T. $\square$

Note that the argumentation does not rely on the specific form of f_{n_i} or the specific form of a rectangular lattice. Rather the following two properties are of importance: f_{n_i} is a function which only depends on the label of the actual tree and the recursive similarities computed in the previous step. The lattice is such that the number of t-neighbors of each fixed neuron increases only polynomially with t. In practice, only a finite precision is available such that the above assumption on $\tilde{d}$ seems reasonable. This argumentation leads to an abstract design criterion for structure processing self organizing nets: Processing of information through the formal representations of the trees and the function *rep* has to be global. It should be guaranteed that all neurons have access to those parts of the network which recognize important substructures.

8 Topology preservation

One important aspect why a lattice structure is added to the neurons in SOM or NG compared to simple vector quantization is the fact that a topology preserving mapping should be found by the network and simple clustering of the data points by neurons or prototypes is not sufficient. A topology allows more complex actions like browsing in the network. If the final lattice structure lies in two or three dimensions, an immediate visualization of the underlying possibly high dimensional data space is provided through the lattice structure. These properties have proven beneficial in a number of application areas like Web-mining, visualization of satellite data or medical images, or robotics [28, 29]. Since the success of these approaches depends crucially on faithful topology representation, the question of what topology preservation precisely means has been studied in depth in the literature [4, 44].

Commonly, topology preservation means that close data points are mapped to neurons which are close in the lattice structure, and conversely, close neurons in the lattice structure come from close data points. Assumed a metric is given on the data points, then the first point is easy to formalize: If a data point x_1 is closer to another point x_2 than x_3 with respect to this metric, then $n(x_1)$, where $n(x_i)$ denotes the winning neuron for data point x_i, should be not further away from $n(x_2)$ than from $n(x_3)$. The converse direction is less obvious since many different data points may lie in the receptive field of a prototype. Commonly, one simply considers the weight attached to the neuron as the representative data point for the neuron and requires that if neuron n_1 is closer to n_2 in the lattice structure than to n_3, then the weight of n_2 should not be further away from the weight of n_1 than n_3. Going from the data points to the weights of the neurons which represent all data points, one can explicitely derive formulas which measure the degree of topology preservation. In [44], for example, the distances of the neurons in the lattice are compared to the distances of their weights. In order to take the fact into account, that the manifold of data points may be curved, the latter distances are measured through the Delaunay graph. This graph can be obtained if precisely those neurons which have intersecting receptive fields in the data manifold are connected with a vertex. Using this definition, one can explicitely compare the neurons which are direct neighbors in the lattice to the degree of neighborhood of their weights in the Delaunay graph and vice versa such that a degree of neighborhood preservation can be computed, too. [4] propose a different measure, the topographic product, which can be computed more efficiently and yields similar results for non curved data manifolds. Note that the focus lies on the ranking of the neurons or their weights, respectively, rather than their precise distances in both approaches.

The standard SOM or NG take care through their learning rule that neighbored neurons in the lattice have similar weights. This is achieved due to the fact that the neighborhood is updated together with the winner. Precise topology preservation can only be obtained for SOM if the dimensionality and the structure of the lattice precisely coincide with the dimensionality and the structure of the training data. Hence the above measures can be used in order to check whether this property holds. If not, a different lattice structure or the use of adaptive methods like the growing SOM [5] are necessary. The neural gas algorithm or topology representing networks, respectively [33, 34], do not use a prior lattice structure but they explicitely define the lattice structure a posteriori such that similar point are mapped to similar neurons: Those neurons become neighbored which are the first two winners for at least one data points. Of course, this will not yield a low dimensional or regular lattice in general. Hence the possibility of easy visualization is lost. However, the topology precisely matches the topology of the given data. One can use the lattice proposed by NG as a reference for an optimum topology.

These are details which indicate that the topic of topology preservation comprises several difficulties even for simple vector spaces. However, the global picture for the standard case is as follows: The input data and the respective metric is represented by the weights of neurons which are contained in the same vector space equipped with the same metric. Topology preservation means that the topology of the weights corresponds to the topology of the lattice, i.e. the weights of two neurons are similar iff the neurons are close in the lattice structure. The fact that close indices yield close weights is accounted for by standard learning algorithms which adapt in each step the whole neighborhood of the winner. The fact that close weights can only be found for close indices of neurons is achieved automatically by the posterior lattice of NG, whereas the dimensionality has to match for SOM. In all cases topology preservation can be measured via comparing the topology of the weights of neurons and the topology of the indices of neurons.

In the case of structured data, the situation is more complicated. A first problem consists in the fact that we do not know in general an appropriate metric on the given input data: labeled trees instead of simple vectors. We can alternatively look at the metrics which are accounted for by the training algorithms. The algorithms induce various distance measures for tree structures. A given trained structure processing network maps trees recursively to an activity profile and, via *rep*, a formal representation which is used to substitute the tree in further processing. Obviously, *rep* induces a pseudometric on trees as follows: $\tilde{d}_{\mathcal{R}} : \mathcal{W}^{\#} \times \mathcal{W}^{\#} \to \mathbb{R}$,

$$\tilde{d}_{\mathcal{R}}(t,t') = d_{\mathcal{R}}(rep(\tilde{d}(t,\mathcal{N}))rep(\tilde{d}(t',\mathcal{N})))$$

where n_i are the neurons of the map and $\tilde{d}$ is the recursive similarity as above. This is a pseudometric since $d_{\mathcal{R}}$ itself is a metric. However, $\tilde{d}$ might not be injective. There exist two alternative pseudometrics:

$$\begin{aligned}&\tilde{d}_{\mathcal{R}_2}(a(t_1,t_2),a'(t_1',t_2')) = \alpha d_{\mathcal{W}}(a,a') + \\ &\beta d_{\mathcal{R}}(rep(\tilde{d}(t_1,\mathcal{N})),rep(\tilde{d}(t_1',\mathcal{N}))) + \beta d_{\mathcal{R}}(rep(\tilde{d}(t_2,\mathcal{N})),rep(\tilde{d}(t_2',\mathcal{N})))\end{aligned}$$

and

$$\tilde{d}_{\mathcal{I}}(t,t') = \text{ distance of the indices of the winners for } t \text{ and } t'$$

What is the role of these pseudometrics in learning? We will argue in the following that the above pseudometrics are induced by the structure processing network for trees and topology preservation can be tested comparing these metrics. Note that we will refer to metrics in the argumentation, dropping the prefix 'pseudo' for simplicity. One has to consider two directions to establish topology preservation: The standard update rule makes sure that similar neurons have similar weights. This direction proposes a metric on the trees induced by the similarity of the weights of the respective neurons, the metric accounted for by the learning rule. Conversely, similar weights can be found at similar neurons. The lattice proposed by NG takes care for this fact. This data optimal lattice structure induces another metric on the trees given by the indices of neurons. Topology preservation means that the two metrics in both directions, which can both be identified with metrics on trees as we will see in the following, coincide.

We consider the metric induced by the weight updates: Given a data point the winner and its neighbors are updated such that their weights become closer to the triple of the label and the formal representations of the two subtrees of the actual tree. Even if we not only applied Hebbian learning, but a gradient descent on an appropriate cost function including all terms of the gradient, we would update each neuron together with its neighborhood into the direction of the respective tree at the respective level. Hence the training algorithms make sure that neurons which are similar in a prior or posterior lattice have similar attached weights, i.e. similar triples. I.e. the update rule accounts for metric $\tilde{d}_{\mathcal{R}_2}$; neighbored neurons get similar weights with respect to this metric.

Consider the converse direction: What would be the optimum lattice proposed by NG? This lattice ensures that close data points are mapped to close neurons. NG would define those prototypes as neighbors which are the first two winners for at least one tree used for learning. Hence trees are mapped to neurons which are close to each other if the first two winners of the neurons are similar. I.e. this argument would propose metric $\tilde{d}_{\mathcal{I}}$ as appropriate metric for the self organizing map. Hence topology preservation means that the topology proposed by $\tilde{d}_{\mathcal{I}}$ and $\tilde{d}_{\mathcal{R}_2}$ match.

What is the role of $\tilde{d}_{\mathcal{R}}$? The metric $\tilde{d}_{\mathcal{R}}$ coincides with $\tilde{d}_{\mathcal{I}}$ for SOMSD. In general, they are related under the following reasonable assumption on *rep*: We demand that *rep* respects winners. I.e., assume $x \in \mathbb{R}^N$. For simplicity, we assume that all components of x are different. Denote by *first*(x) the index i_0 such that $x_{i_0} \leq x_i \, \forall i$. Assume x, x', y, $y' \in \mathbb{R}^N$ are vectors such that *first*$(x) =$ *first*(x') and *first*$(y) \neq$ *first*(y'). Then $d_{\mathcal{R}}(\mathit{rep}(x), \mathit{rep}(x')) \leq d_{\mathcal{R}}(\mathit{rep}(y), \mathit{rep}(y'))$. This property means that activations due to trees which have the same winner yield formal representations which are more similar than representations of activations with different winners. Under this assumption $\tilde{d}_{\mathcal{R}}$ and $\tilde{d}_{\mathcal{I}}$ are similar. This property holds for recSOM if the winner has a sufficiently smaller activation than the remaining neurons and it is obviously fulfilled for SOMSD. Moreover, the assumption on *rep* is reasonable if we assume that the location of the winner is important information for the formal representation of trees. It states that *rep* respects this information.

Obviously, the linear combination $\tilde{d}_{\mathcal{R}_2}$ is closely related to $\tilde{d}_{\mathcal{R}}$ and can be recovered from it. $\tilde{d}_{\mathcal{R}}$ corresponds to a further application of a recursive processing step *rep*. Hence we can expect that these metrics are similar due to the learning process. Under the above assumption that $\tilde{d}_{\mathcal{R}}$ and $\tilde{d}_{\mathcal{I}}$ can be merely identified, topology preservation boils down to matching metrics $\tilde{d}_{\mathcal{R}_2}$ and $\tilde{d}_{\mathcal{R}}$. However, note that unlike the standard SOM there might be a principled difficulty due to the involved structure which manifests very nicely in the comparison of these metrics: $\tilde{d}_{\mathcal{R}_2}$ and $\tilde{d}_{\mathcal{R}}$ are defined via a domain of different dimensionality. We have to consider formal representations on the one hand and triples consisting of labels and formal representations on the other hand. The dimensionality cannot match in principle if dense data sets are contained in both parts. Hence it is likely that topologic defects can be observed in dense regions. As already pointed out beforehand for SOMSD, this yields discontinuities and clusters of similar structure in the lattice because we try to embed a higher dimensional space into a lower dimensional space: the triples consisting of labels and indices into the indices, for example. Nevertheless GSOMSD might show perfect topology preservation for sparse data which is often given in real life applications involving complex structures. Moreover, it can of course visualize the local topological structure although some global discontinuities might be unavoidable.

Since $\tilde{d}_{\mathcal{R}}$ and the other metrics are given implicitly through the network, the question arises whether we can approximate this measure by a metric which is given explicitely for tree structures. That means, we would like to obtain a metric for trees which approximates $\tilde{d}_{\mathcal{R}}$ and which does not depend on the recursive processing of trees through the network.

Explicit metric for SOMSD

We here investigate the case of SOMSD: The neurons lie on a regular lattice and *rep* yields the index of the winning neuron, i.e. the neuron with smallest recursive similarity, and we assume that $d_{\mathcal{W}}$ and $d_{\mathcal{R}}$ are metrics for which, in particular, the triangle inequality holds. For simplicity, denote by $n(t)$ the winning neuron for tree t and by $\vec{\iota}(t)$ the index of the winning neuron, denote by r_ξ the index of the empty tree ξ, as above. We assume that this index is not reachable by any other tree. Given a neuron n, denote the index of this neuron by the symbol $\vec{\iota}(n)$. Define the following metric on trees: $\tilde{d}_T : \mathcal{W}^\# \times \mathcal{W}^\# \to \mathbb{R}$, $\tilde{d}_T(\xi, \xi) := 0$, $\tilde{d}_T(t, \xi) = \tilde{d}_T(\xi, t) := d_{\mathcal{R}}(r_\xi, \vec{\iota}(t))$ for $t \neq \xi$, and

$$\tilde{d}_T(a(t_1, t_2), a'(t_1', t_2')) := \alpha' \, d_{\mathcal{W}}(a, a') + \beta' \, \tilde{d}_T(t_1, t_1') + \beta' \, \tilde{d}_T(t_2, t_2')$$

where α' and β' are positive constants. This is a metric since $\tilde{d}_T$ is obviously symmetric, positive for two different inputs, and zero for two identical inputs. The fact that the triangle inequality holds follows easily by induction over the height of the trees. $\tilde{d}_T$ is a recursive metric which no longer depends on the processing of trees. The network does only determine the distance of the representation for the empty tree r_ξ to other trees. We would like to correlate the metric $\tilde{d}_{\mathcal{R}}$ induced by SOMSD to such a metric $\tilde{d}_T$ with appropriate constants α' and β' of the above form. We assume for this purpose, that the lattice is a perfect topology preserving mapping. More precisely, we assume the following two facts:

- We assume a perfect topology preservation and preservation of the scaling of all weights, i.e. there exist positive values α' and β' and a value $Q_0 > 0$ such that for all neurons n_1 and n_2 and their respective indices and labels, the following inequality is valid:

$$\begin{array}{l}|d_{\mathcal{R}}(\vec{\iota}(n_1), \vec{\iota}(n_2)) - \alpha'\, d_{\mathcal{W}}(\mathcal{L}_0(n_1), \mathcal{L}_0(n_2)) \\ \quad - \beta'\, d_{\mathcal{R}}(\mathcal{L}_1(n_1), \mathcal{L}_1(n_2)) - \beta'\, d_{\mathcal{R}}(\mathcal{L}_2(n_1), \mathcal{L}_2(n_2))| \leq Q_0\end{array}$$

I.e. the distances of the indices can be linearly related to the distances of their respective labels. This property not only includes that the order of the labels of the neurons and their respective indices match which is the standard requirement for topology preservation, but also the respective distances can be compared. In general, this property will hold only locally as already discussed. We will point out the implications if the above property holds only locally, afterwards.

- We assume a certain granularity of the lattice structure. I.e. we find positive values ϵ_0 and ϵ_1 such that we find a node n with label $(\mathcal{L}_0(n), \mathcal{L}_1(n), \mathcal{L}_2(n))$ for each triple $(a, r_1, r_2) \in \mathcal{W} \times \mathcal{R} \times \mathcal{R}$ which occurs during the computation with $d_{\mathcal{W}}(a, \mathcal{L}_0(n)) \leq \epsilon_0$, $d_{\mathcal{R}}(r_1, \mathcal{L}_1(n)) \leq \epsilon_1$, and $d_{\mathcal{R}}(r_2, \mathcal{L}_2(n)) \leq \epsilon_1$. This property is fulfilled with small values ϵ_0 and ϵ_1 if training has been done with a sufficient number of neurons such that the neurons form a sufficiently dense covering of the data space.

Theorem 8.1 *Under the above assumptions the following inequality holds for all trees t and t' which are in the scope of the network :*

$$\left|\tilde{d}_{\mathcal{R}}(t,t') - \tilde{d}_T(t,t')\right| \leq (Q_0 + 2\alpha' Q_1 + 4\beta' Q_2)\frac{1-(2\beta')^{H+1}}{1-(2\beta')}$$

where H is the minimum of the height of t and t', respectively, $\tilde{d}_T$ is computed with the above constants α' and β', $Q_1 = \epsilon_0 + 2\beta\epsilon_1/\alpha$, and $Q_2 = \alpha\epsilon_0/\beta + 2\epsilon_1$.

Proof. The proof is via induction over the height of the trees t and t', respectively. If t and t' are both empty, then the above distance is obviously 0. If only one of both trees, say t, is empty, we find $\tilde{d}_{\mathcal{R}}(t,t') = d_{\mathcal{R}}(r_\xi, \vec{\iota}(t')) = \tilde{d}_T(t,t')$. Otherwise, we can write $t = a(t_1, t_2)$ and $t' = a'(t_1', t_2')$. We find

$$\begin{array}{rl} & |\tilde{d}_{\mathcal{R}}(t,t') - \tilde{d}_T(t,t')| \\ = & |\tilde{d}_{\mathcal{R}}(\vec{\iota}(t), \vec{\iota}(t')) - (\alpha'\, d_{\mathcal{W}}(a,a') + \beta'\, \tilde{d}_T(t_1,t_1') + \beta'\, \tilde{d}_T(t_2,t_2'))| \\ \leq & Q_0 + |\, \alpha'\, d_{\mathcal{W}}(\mathcal{L}_0(n(t)), \mathcal{L}_0(n(t'))) - \alpha'\, d_{\mathcal{W}}(a,a') \\ & + \beta'\, d_{\mathcal{R}}(\mathcal{L}_1(n(t)), \mathcal{L}_1(n(t'))) - \beta'\, \tilde{d}_T(t_1,t_1') \\ & + \beta'\, d_{\mathcal{R}}(\mathcal{L}_2(n(t)), \mathcal{L}_2(n(t'))) - \beta'\, \tilde{d}_T(t_2,t_2')| \\ =: & (*) \end{array}$$

$(a, \vec{\iota}(t_1), \vec{\iota}(t_2))$ is closest to the weight of $n(t)$ compared to all other weights. This implies that the value $\alpha\, d_{\mathcal{W}}(a, \mathcal{L}_0(n(t))) + \beta\, d_{\mathcal{R}}(\vec{\iota}(t_1), \mathcal{L}_1(n(t))) + \beta\, d_{\mathcal{R}}(\vec{\iota}(t_2), \mathcal{L}_2(n(t)))$ is minimum. We have assumed a certain granularity, i.e. there exists a weight (a, r_1, r_2) such that the distance to $(a, \vec{\iota}(t_1), \vec{\iota}(t_2))$ is at most $\alpha\epsilon_0 + 2\beta\epsilon_1$. Hence we obtain the estimation $d_{\mathcal{W}}(a, \mathcal{L}_0(n(t))) \leq \epsilon_0 + 2\beta\epsilon_1/\alpha = Q_1$ and $d_{\mathcal{R}}(\vec{\iota}(t_i), \mathcal{L}_i(n(t))) \leq \alpha\epsilon_0/\beta + 2\epsilon_1 = Q_2$ for $i \in \{1,2\}$. An analogous estimation can be obtained if t is substituted by t'. Therefore we find using the triangle inequality

$$\begin{array}{rl} & (*) \\ \leq & Q_0 + 2\alpha' Q_1 + 4\beta' Q_2 + \beta' |d_{\mathcal{R}}(\vec{\iota}(t_1), \vec{\iota}(t_1')) - \tilde{d}_T(t_1,t_1')| \\ & + \beta' |d_{\mathcal{R}}(\vec{\iota}(t_2), \vec{\iota}(t_2')) - \tilde{d}_T(t_2,t_2')| \\ \leq & Q_0 + 2\alpha' Q_1 + 4\beta' Q_2 + 2\beta'(Q_0 + 2\alpha' Q_1 + 4\beta' Q_2)\frac{1-(2\beta')^H}{1-(2\beta')} \\ = & (Q_0 + 2\alpha' Q_1 + 4\beta' Q_2)\frac{1-(2\beta')^{H+1}}{1-(2\beta')} \end{array}$$

by induction. □

Note that the above term can be uniformly bounded by $(Q_0 + 2\alpha' Q_1 + 4\beta' Q_2)/(1-(2\beta'))$ if $\beta' < 0.5$. In this case, it can be made arbitrarily small if the quantization granularity is increased, i.e. the quantities

Q_0, Q_1, and Q_2 approach 0. Hence we obtain an estimation of the deviation of the metric $\tilde{d}_{\mathcal{R}}$ which is taken into account by the self organizing learning algorithm, and the metric $\tilde{d}_T$ which constitutes a recursive metric directly defined on the trees. The assumptions under which this estimation holds are twofold: The mapping possesses a certain granularity. This assumption is reasonable since the mapping performs a quantization through the codebooks. If the information lost at the quantization steps is large, we cannot expect a thorough representation of trees and their respective distances. The distances concerning the representation in SOMSD are naturally affected by identification of certain trees through this quantization. A second assumption consists in the fact that the distances of the indices and their respective weights are linearly correlated. This implies, of course, standard topology preservation. However, it is a stronger assumption since not only the rankings but the precise values have to match.

It will in general only locally be fulfilled due to two reasons: For non-uniform data distributions it can be expected that the neurons accumulate at regions of high probability. This means that the correlation between the neurons and their respective weights will be nonlinear. This effect can explicitely be computed for specific cases for the standard SOM: The density of neurons with respect to the underlying data distribution has a specific ratio [38]. Hence the distances of the indices are larger for neurons which lie in regions with a high density of the data compared to regions where only few data, hence few neurons can be observed. What is the effect if recursive data are processed? Naturally, the deviation of subtrees which are represented by neurons in a region with high resolution is ranked higher than deviations in regions were the resolution of the map is low because in further processing steps the indices of neurons in the respective region are used as representation of the trees. Hence instead of using a simple linear map in $\tilde{d}_T$ which combines the distances of the labels and the respective subtrees, a more involved mapping which depends on the respective deformation of the distances through the resolution of the map would be necessary. This recursive computation would implicitly use large weighting factors β' if the indices are contained in a region with high resolution and small factors if the neurons are sparse in the respective region. This mapping can, of course, be locally approximated by simple linear maps such that the above estimation holds for trees which are similar, hence in the same local linear region of the map. Globally, the function $\tilde{d}_T$ need no longer be a metric and involves a scaling term β' which explicitely depends on the respective local resolution of the network.

A second problem lies in the fact that the above assumption of topology preservation will in general not be fulfilled as already discussed. Since the dimensionality of the weights is always of a larger dimension than the lattice – the lattice indices are part of the weights in SOMSD – topology preservation is impossible for the global map in the above form for continuous data distributions with support of positive measure. Moreover, it can be expected even for realistic discrete data sets that two-dimensional or low-dimensional lattice structures for SOMSD give rise to topological defects with respect to the formal representation of trees: It is very likely that neurons which weights include non similar formal representations of the two subtrees are close to each other in the network. ¿From the experiments of [18] it can be observed that the winning neurons for the same structure form clusters. Naturally, there exist different clusters which are neighbored in the lattice structure. Hence there is a discontinuity if the corresponding indices are taken into account: The indices might be similar, whereas the respective trees which are represented are different concerning the structure. It is a remarkable property of SOMSD that it tries to avoid this deformation as much as possible vie introducing idle units at the borders between two clusters. Depending on the overall size of the network compared to the number of data points, [18] observes that a certain percentage of the neurons is not used for representation of data but only for separation of two different classes, i.e. putting as many neurons between two different classes as possible such that their respective indices are as different as possible. Hence the estimation of Theorem 8.1 will only locally hold, i.e. within regions where no topology deformation can be observed and trees of similar structure are clustered together. At the borders of these respective clusters, the indices of neurons are similar whereas the corresponding weights are not.

Distributed representation of the trees

An alternative way to avoid the similarity of formal representation of trees belonging to different clusters consists in the use of a different representation of the indices. We could simply enumerate all neurons by 1 to N and store the indices in a unary way, the i th unit vector in $\mathbb{R}^N$ representing the index of neuron i as already proposed for VQ. Then the Euclidian distance restricted to these indices yields the simple differentiation:

$$d_{\mathcal{R}}(r_1, r_2) = \begin{cases} \sqrt{2} & \text{if } r_1 \neq r_2 \\ 0 & \text{otherwise} \end{cases}$$

where r_1 and r_2 are two unit vectors. Correspondingly, the norm $\tilde{d}_R$ induced on trees has the following simple form

$$\tilde{d}_{\mathcal{R}}(t, t') = \begin{cases} \sqrt{2} & \text{if } t \text{ and } t' \text{ have the same winner} \\ 0 & \text{otherwise} \end{cases}$$

Hence SOMSD with unary representation of the indices simply forms equivalence classes of trees which are mapped to the same neuron. Only the root label of a current processed tree and the equivalence class of the two respective subtrees are important for the comparison. The metric $\tilde{d}_R$ itself consists in a comparison of the equivalence classes, the related metric of the form $\alpha\, d_{\mathcal{W}} + \beta\, \tilde{d}_{\mathcal{R}} + \beta\, \tilde{d}_{\mathcal{R}}$ compares the root labels of two trees and the equivalence classes of their subtrees. In particular, different classes of structures with different winners are clearly separated in this approach hence making the use of idle units for the separation of classes in SOMSD superfluous. However, the similarity of the formal representation of trees of the same structure which lie within a cluster but have different winners is lost.

The formal representation for a tree in the recursive SOM consists of the vector

$$(\exp(-\tilde{d}(t, n_1)), \ldots, \exp(-\tilde{d}(t, n_N)))\,,$$

n_i being the neurons and $\tilde{d}$ the recursive similarities. This approach constitutes a compromise between the two demands: to separate the formal representations of trees with different structures in different clusters and to measure the degree of similarity of trees of the same or similar structure within a cluster. It can be expected that the weights of neurons within a cluster are similar, whereas a discontinuity of the weights can be observed at the borders. There the formal representation of the two subtrees, for example, will be different because the trees have a different structure. This will yield an activation profile, i.e. a formal representation of the actual tree, which shows a high activation for the winner and some kind of Gaussian activation profile for the surrounding neurons in the same cluster. Neurons in different clusters show only very low activation. However, due to high dimensional formal representations accumulation of small errors to comparably large terms might occur in this approach.

9 Discussion

We have proposed a general framework for unsupervised processing of structured data based on the key idea that processing obeys the recursive dynamics of the given data. For this purpose, the dynamics of supervised recurrent and recursive networks could be transferred directly to the unsupervised framework such that many common concrete approaches like TKM, RecSOM, SOMSD, and even recurrent and recursive networks are covered by the framework. A key issue of the dynamics consists in the notion of internal representations of context which enables networks to store activation profiles for recursively processed substructures in a distributed manner in a finite and fixed dimensional vector space. This allows the comparison of structural data of arbitrary size and the adaptation of internal representations towards a currently processed structure e.g. in Hebbian learning. The concrete choice of the internal representation in this framework is a crucial point for the success of the approaches and the several proposals which can be found in the literature differ with respect to this point. We have shown that the choice of *rep* has consequences on the noise tolerance of the network, the in-principle capability of representation, and the notion of topology preservation as well as the topology induced by the recursive dynamics on the original structured data set. This allows us to compare concrete approaches based on theoretical considerations even prior to training. Moreover, the general framework allows to formulate training mechanisms in a uniform manner. Hebbian learning with various topologies such as VQ and NG topology can immediately be formulated. It turns out that unlike the case of unsupervised vector processing Hebbian learning is only an approximation of a gradient dynamics of appropriate cost functions. We have formulated the cost functions for the general framework and we derived several ways to precisely compute the gradients. Hebbian learning disregards in all cases the contribution of substructures and hence is much less costly than the precise approaches. Nevertheless, the precise formulation allows us to recover supervised training mechanisms within the general framework, too. An appropriate of the cost functions yields standard training algorithms for supervised recurrent networks like BPTT and RTRL. Moreover, this formulation shows how to transfer different learning paradigms such as generalized vector quantization and variations to the recursive case [21, 39].

Starting from this general formulation, several questions are of interest and will be the subject of further investigation:

- Which explicit metrics are induced by the recursive framework in the several cases? Can an explicit metric induced in the supervised case be found, too?
- In which cases of given training data can topology preservation be guaranteed?
- Do there exist any cases where training with approximate Hebbian learning and the true gradients makes a difference?
- Can general measures how to evaluate the performance of recurrent SOMs be derived?
- Can convergence be guaranteed for the recursive SOM?
- What is the magnification factor of the distribution represented by the map in comparison to the data distribution? Does the factor depend on the depth of the processed structure and the parameters α and β?
- Can an optimal representation *rep* be learned during training instead of using a fixed priorly defined function?

References

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

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

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

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

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

[6] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE TNN*, 5(2):157–166, 1994.

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

[8] F. Costa, P. Frasconi, V. Lombardo, and G. Soda. Towards incremental parsing of natural language using recursive neural networks. To appear in: *Applied Intelligence*.

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

[10] I. Farkas and R. Miikkulainen. 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, 1999.

[11] P. Frasconi, M. Gori, A. Küchler, and A. Sperduti, 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, 2001.

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

[13] P. Frasconi, M. Gori, and A. Sperduti. Learning Efficiently with Neural Networks: A Theoretical Comparison between Structured and Flat Representations. In: *ECAI 2000. Proceedings of the 14th European Conference on Artificial Intelligence*, IOS Press, 2000.

[14] C. L. Giles, G. M. Kuhn, and R. J. Williams. Special issue on dynamic recurrent neural networks. *IEEE Transactions on Neural Networks*, 5(2), 1994.

[15] M. Gori, M. Mozer, A. C. Tsoi, and R. L. Watrous. Special issue on recurrent neural networks for sequence processing. *Neurocomputing*, 15(3-4), 1997.

[16] S. Günter and H. Buhnke. 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, 2001.

[17] M. Hagenbuchner, A.C. Tsoi, and A. Sperduti. 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, 2001.

[18] M. Hagenbuchner, A. Sperduti, and A.C. Tsoi. A Self-Organizing Map for Adaptive Processing of Structured Data. to appear in *IEEE Transactions on Neural Networks*.

[19] B. Hammer. *Learning with Recurrent Neural Networks*. LNCIS 254, Springer, 2000.

[20] B. Hammer, A. Micheli, and A. Sperduti. 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, 2002.

[21] B. Hammer and T. Villmann. Generalized relevance learning vector quantization. To appear in *Neural Networks*.

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

[23] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neur. Comp.*, 9(8):1735–1780, 1997.

[24] A. Hoekstra and M.F.J. Drossaers. 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, 1993.

[25] D.L. James and R. Miikkulainen. 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, 1995.

[26] J. Kangas. Time-delayed self-organizing maps. In: *Proc. IEEE/INNS Int. Joint Conj. on Neural Networks 1990*, 2, pages 331-336, 1990.

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

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

[29] T.Kohonen. *Self-organizing maps*. Springer, 1997.

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

[31] T. Koskela, M. Varsta, J. Heikkonen, and K. Kaski. 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, 1998.

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

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

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

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

[36] B. A. Pearlmutter. Gradient calculations for dynamic recurrent neural networks: A survey. *IEEE Transactions on Neural Networks*, 6(5):1212–1228, 1995.

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

[38] H. Ritter and K. Schulten. On the stationary state of Kohonen's self-organizing sensory mapping. *Biological Cybernetics*, 54(1):99-106, 1086.

[39] A. S. Sato and K. Yamada. Generalized learning vector quantization. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems*, volume 7, pages 423–429. MIT Press, 1995.

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

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

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

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

[44] T. Villmann, R. Der, M. Herrmann, and T. Martinetz. Topology Preservation in Self–Organizing Feature Maps: Exact Definition and Measurement. *IEEE Transactions on Neural Networks*, 8(2):256–266, 1997.

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

[46] Recursive self-organizing maps. To appear in: *Neural Networks*.

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