Data Mining on Sequences with recursive Self-Organizing Maps

Sebastian Blohm
Universität Osnabrück
sebastian@blomega.de

Bachelor's Thesis
International Bachelor Program in Cognitive Science,
Universität Osnabrück

Abstract

Analyzing sequences of continuous data is an important step in perception. It has been shown that extensions of the self-organizing map (SOM) learn temporal dynamics. Here, the aim is to find an abstract symbolic description using the SOM for Structured Data (SOMSD). Sequences of real vectors with added noise are generated by stochastic processes described by Markov models and are trained to a SOMSD. Two algorithms are presented that can extract finite order Markov models (FOMMs) from the trained SOMSD using clusterings of the map according to each neuron's weight and context information. Clustering is done using U-Matrices. The algorithms succeed in producing Markov models similar to those used for sequence generation. Comparison of the extracted FOMMs and the input models allows inferences on how the temporal dynamics are represented and on whether the SOMSD in combination with U-Matrix clustering can be used for data mining on sequences.

Table of Contents

1. Introduction

Neural Networks are structures that are built to process computations distributed on many connected minimal units. These units are called neurons as they imitate information processing in the human brain, which also takes place in a distributed manner. Human learning, that is the change of behaviour due to experience (here seen as pattern input), takes place through changes to neurons and their connectivity. The Psychologist Donald Hebb[6] has proposed a learning principle, in which temporal correlation of stimuli leads to a modification of connections between neurons (the synapses) and amplifies future reactions of the neurons in a way that allows association of the previously correlated stimuli. The respective biological mechanisms have been demonstrated on single neuron basis[18]. Training of artificial neural networks can be based on similar principles (the modification of the reaction strength of single neurons) which might allow to draw interesting analogies.

Neural networks use the notion of activation of neurons as the main values that operations are applied to. Activation can take place directly through sensor input or through weighted connections to activated neurons. Input, connection weights and activation are usually represented as vectors. Many neural networks are constructed to process input data of a fixed format that is presented as a whole at one point of time. The mathematical basis of such processing is well understood. When a fixed data format is used, vector space algebra can be applied. The approximation of unknown relations through vectors can be done using standard methods. Analogies to this one-directional information processing can be found in early phases of perception. For example in the visual cortex, where the flow of information is basically one-directional, neurons can be identified that respond to particular shapes[11, p. 532].

However, real-world data might be too complex to be directly encoded into one vector. Complex temporal or spatial relations between values might contain important information. Therefore other architectures of neural networks have been developed that allow calculations on input of more complex structure such as time series. Here, data is presented in several steps, the order of which holds parts of the information. Among the frequently formulated tasks of such networks (mostly networks with recurrent or dynamic features) are time series prediction and classification. It is important to note that the scenario of successively incoming input with the linear order holding information can be assumed to be the standard case in human information processing. Continuously active [11, ch. 30] pathways have been observed. In general, it is no problem for humans to recognize sequenced stimuli as coherent. For example, a sequence of phonemes is recognized as a word or changes of visual input over time can easily be identified as one movement. One possibility to allow computation on sequences in neural networks is to introduce the notion of time steps. This leads to a discretisation of the data with respect to the temporal (or spatial, or otherwise relational) dimension. The output of the network now not only depends on the current input but also on activations in previous time steps. This can be done by adding connections among neurons that transfer the value of activation of time step i of one neuron to another neuron at time step i+ 1 (recurrent networks) [14].

Neural networks approximate functions by assigning to each possible input pattern an output calculated through activation of neurons. Two fundamentally different types of learning tasks for neural networks can be distinguished: In supervised learning, the connection strength of neurons (weights) is adjusted to fit a given set of input-output pairs. In unsupervised learning, no such input-output pairs are given. The adjustment of weights only takes place on the basis of input data. The function that is produced in unsupervised learning reflects regularities in the input and therefore yields a – previously unknown – description of the data. In supervised learning, standards for processing of sequential data have already been established[14]. However, unsupervised models with recurrence are not yet well established and only few applications of recurrent unsupervised models exist so far. A taxonomy and standards of unsupervised recurrent models are much less straightforward than in the supervised case[1].

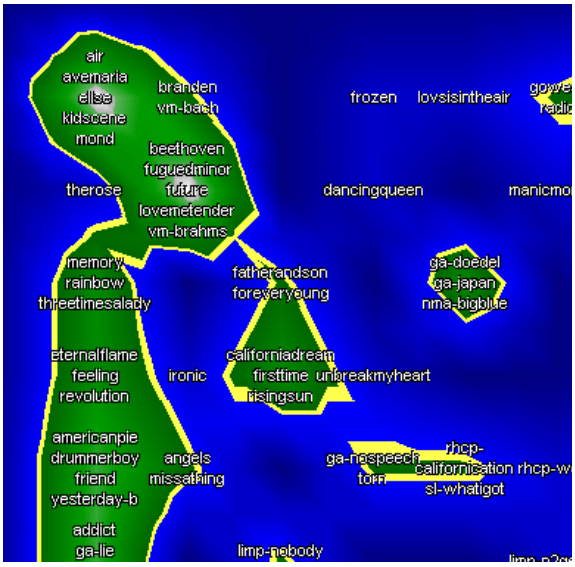


Figure 1 Example of a SOM trained to classify songs by acoustic properties [22]. Some titles are given. Unsupervised training is assumed to arrange similar songs next to each other on the map. Areas representing numerous songs are coloured green.

A very successful network paradigm for unsupervised learning is the self-organizing map (SOM) proposed by Teuvo Kohonen[13]. The SOM is a (usually) two-dimensional arrangement of neurons that assigns similar input to the same or neighbouring neurons, such that arrangement of data in reduced dimensions as well as classification are possible. The SOM is frequently used because it is able to produce intuitively understandable 2D visualisations (see Figure 1 for an example study). Its quantisation and clustering abilities make the SOM a powerful tool in data mining. For a given input pattern, a winner neuron is determined. The winner neuron is defined to be the neuron that is closest to the pattern with regard to a previously chosen distance measure. This neuron is considered to be the SOM's representation of the input pattern. Training is done by gradual adaptation of the neurons called Hebbian learning. The adaptation assures that similar input patterns are represented in the same area of the SOM which leads to the above mentioned advantages. Analogously, structures representing similarity through neighbourhood can be found in the human brain in the form of cortical maps in perception [11, p. 344] and on higher level as regions activated during the processing of linguistic concepts[5, ch. 8].

Several approaches have been taken to apply unsupervised SOM learning to sequenced data. One way to make neural networks process sequences is to divide the sequence into time windows of a fixed size and combine the elements of a time window to an input vector that is presented at a time[7]. This method makes the step from static to sequential input in both supervised and unsupervised learning trivial. However, it results in obvious disadvantages: The influence of the context (previous input) is restricted to a fixed number of steps back in time. And, the higher this number is, the higher is the dimensionality of the input. A high dimensionality makes

function approximation difficult as no dense samples of data can be obtained: the curse of dimensionality. Therefore several approaches have been developed to use the SOM directly for sequential data. They use modified distance measures, that take previous activation (context) into account. Classical approaches are the Recurrent SOM (RSOM)[12] and the Temporal Kohonen Map[2] both integrating each neuron's old activity when calculating new activations. A recent approach is the Recursive SOM (RecSOM)[20], where the previous activation of the SOM is considered part of the input to the next time step. Here, the SOM for Structured Data (SOMSD)[8] is used, which is a recursive technique that represents the previous activity as references to a point on the SOM's grid, so called context vectors. Like the RecSOM, the SOMSD processes information on previous activation that is contained in the context-vectors as additional input to the next time step. However, in the SOMSD the context vector is a low-dimensional representation of the previous activation. The SOMSD is therefore a compressed model with reduced complexity. In spite of this reduction, the SOMSD has performed comparably well to the RecSOM in case studies. The SOMSD has originally been developed for the domain of tree-structured data. Its application in this study will be restricted to sequences which is possible because sequences can be considered a special case of tree structures.

The work presented here consists of experiments that try to apply SOM data mining methods to the SOMSD. It is tried to recognize regularities underlying the generation of input data in the weight and context vectors. For this purpose, data has been generated using Markov models. The question is how temporal structures are represented in the SOMSD so that clustering allows to re-extract the Markov model. This is done to asses the applicability of recursive SOM representations as a basis for data mining. The material used here consists of sequences of real values with some noise to simulate real world data (i.e. sensoric input). As opposed to symbolic sequences further quantisation is necessary to find a Markov model. Quantisation is a strength of SOMs in particular as they allow clustering. Clustering is here done with the help of the U-Matrix[19].

This thesis is organized as follows: In section 2 the SOM algorithm is described in detail and in section 3 the U-Matrix clustering technique is presented. Section 4 presents the SOMSD as a network for sequence learning before in section 5 Markov models are introduced. In section 6 considerations are made on how performance of sequence processing SOMs can be measured. Section 7 outlines the questions for the experiments. In section 8 a data mining algorithm is presented. Experiments with this algorithm are presented in section 9 as well as further experiments in section 10. These are followed by considerations on the generalization of the technique to HMMs in section 11. If the reader is unfamiliar with training of SOMs referring to the training subsection of section 9 for practical issues might help to imagine the process. Section 12 holds concluding remarks while section 13 gives suggestions for further research. In the appendix implementation details and some output from the experiments can be found.

2. The SOM

The SOM is a powerful and intuitively understandable tool for unsupervised learning[13]. Input patterns are usually multi-dimensional vectors out of one input space G. To allow reasonable comparison, neuron weight vectors also are vectors

from G. As an unsupervised learning mechanism, the SOM is able to find descriptive mapping of the input space on a usually two-dimensional output space solely based on the input. The two-dimensional output space is considered to be the map with each area of the map representing an area of the input space. Local similarity of input patterns hereby is reflected by proximity on the map. Training, that is the iterative adjustment of weight vectors to obtain a desired mapping, is done by successive presentation of all input patterns where each presentation includes the adjustment of weights to the presented pattern (Hebbian learning). These adjustments lead to the shaping of the map.

Formally, the SOM is defined as a set of neurons with each neuron n_j having a weight vector w_j from the same vector space G as the input patterns x_i. Neurons are arranged on a (usually two-dimensional) grid that assigns each neuron n_j a grid coordinate g_j (usually $g_j \in N^2$). The activation of each neuron depends on its distance to the input pattern, therefore a distance function $d(x_i, n_j) : G \times G \to \mathbb{R}$ is applied. The distance function allows to determine the winner of each pattern. The winner c of x_i is the neuron with the smallest distance d to x_i:

$$c = \arg\min_j \{d(x_i, w_j)\}.$$

A canonical distance function is the squared Euclidian distance:

$$d(x_i, w_j) = \|x_i - w_j\|^2.$$

Figure 2 shows an example of winner selection. Each neuron becomes winner of some part of the input space. It can be considered representing this part. Thus, the whole input space is mapped onto the grid of neurons, which consequently forms a map of the input space. In Figure 1 for example, the input space was given as a set of feature vectors describing pieces of music. Each neuron on the map is selected winner for different songs, depending on their neighborhood in the input space. If the feature vector have been generated so that music of the same style is close by in the input space, a neuron or a set of neurons can be seen to represent a particular style of music.

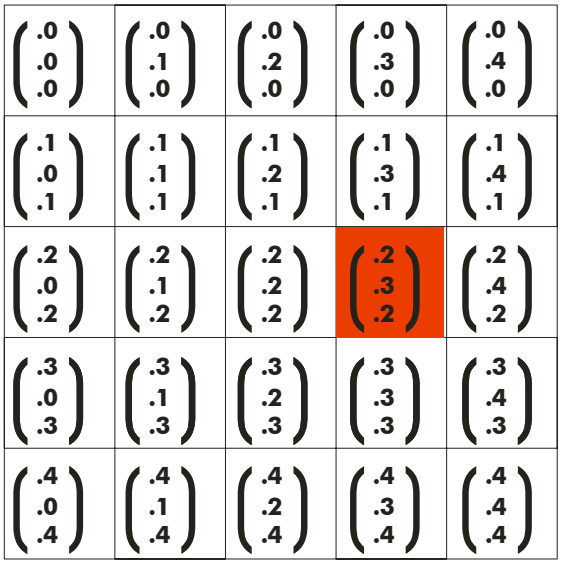


Figure 2 Weight vectors of a SOM arranged on a grid. The winner of the input pattern $\binom{.2}{.3}$ is marked in red.

During training, neurons are adjusted according to input data to produce a salient map. The SOM training algorithm specifies, what is done during each presentation of each input pattern x_i.

As already described, the winner neuron n_c is determined using $c = \arg\min_j \{d(x_i, w_j)\}$ and usually $d(x_i, w_j) = \|x_i - w_j\|^2$.

All neurons w_j are then adjusted by Δw_j depending on their degree of neighbourhood to the winner, the learning rate and the difference from the input pattern:

$$\Delta w_j = h(n_c, n_i)\eta(x - w_j)$$

The degree of neighbourhood to the winner neuron n_c is given by the function h. The learning rate η is a parameter of the training session and is usually larger in early phases of the training to allow faster broad arrangements prior to careful fine tuning.

An example neighbourhood function is

$$h(n_c, n_j) = e^{\frac{-\|g_j - g_c\|^2}{2\sigma^2}}.$$

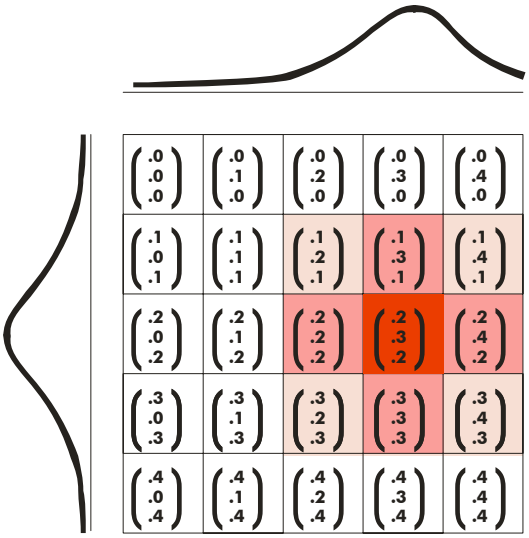


Figure 3 Neighbourhood function. The winner is coloured red, other neurons are coloured in shades of red according to their degree of neighbourhood. The curves on the axis give 1-d samples of a neighbourhood function.

The application of the neighbourhood function is illustrated in Figure 3. It has the essential properties of being maximal (equal to 1) for $n_j = n_c$. It decreases with larger grid distance of the two neurons. $\sigma > 0$ is also a parameter that is reduced during training to ensure broader neighbourhood in the beginning of the training. This ensures the formation of patches in the beginning and stable arrangement later. After that the SOM training algorithm proceeds to the next input pattern x_{i+1}.

Training a SOM with the appropriate parameters leads to a map representation with the key property that regions of similar input patterns are represented by one region on the SOM. The way neighbours are adjusted leads to the effect that if there are regions in the input space from which many patterns originate, their representing regions on the map are larger. In these cases, clustering is possible, that is these regions can be determined and considered distinct classes that are represented on the SOM. Input patterns now can be assigned to one of these classes according to the location of their winner on the SOM.

3. The U-Matrix

The U-Matrix[19] constitutes a popular tool for extraction of clusters and will be used in this project. The aim is to divide the set of all neurons N into m subclasses $\{W_1 \ldots W_m\}$ according to unknown coherence in the input space. The U-Matrix exploits the fact that when parts of the input space are mapped onto the SOM the area of the map representation does not correlate with the size of the part[1]. Space taken by the representation of a particular input space region on the SOM rather depends on the frequency of presentation of input patterns from that region. The U-Matrix allows to detect such differences in density and to use them for clustering. As clusters are regions of frequent patterns, they take more area on the SOM than adjacent parts of the input space. Consequently less neurons are available to cover the input space

[1] Note that the input space might be of much higher dimensionality than the SOM grid.

between clusters. The U-Matrix is based on the observation that for this reason, there are larger differences between the weight vectors of neighbouring neurons on cluster borders and between clusters. Each neuron n_j is assigned a U-value,

$$u(n_j) = \sum_{r(n_j,n_k)=1} \|w_j - w_k\|$$

the sum of the distance to all its immediate neighbours. Therefore r is chosen as $r(n_j, n_k) = 1$ if n_j and n_k are direct neighbours[2] on the grid and 0 otherwise.

Each neuron's U-value will be understood as its height, so that a 2D SOM and its U-Matrix form a 3D landscape (Figure 4). The following properties of this landscape are used for clustering:

- If several neighbouring neurons represent similar parts of the input space, they lie in a common valley.
- The valley is larger (contains more neurons) and deeper (lower u-values) if more patterns come from this part of the input space.

Reversely, mountain ranges exist between the clusters. One valley on the U-Matrix can therefore be seen as a cluster. A clustering algorithm thus has to recognize valleys in the landscape.

A heuristic has been developed for this project to allow the determination of a reasonable amount of clusters. This clustering method uses a 'water level' approach: Whatever lies within a common lake, when the landscape is flooded up to a given level is regarded to belong to the same cluster. A neuron is considered flooded, if it's U-value is smaller than a fixed water-level value p: $u(n_j) < p$. Two flooded neurons belong to the same cluster, if and only if there is a path on the grid from one neuron to the other one only passing flooded neurons.

If the SOM is trained properly and the U-Matrix is clearly expressed, no important cluster borders are flooded and no clusters that might belong together from external knowledge of the input space are split. This assumes a reasonable choice of p. The choice here is done in the following way, which has empirically shown to be a good method. Let i be equal to the number of neurons winning at least one pattern and choose p to cover i/2 neurons. After these clusters are determined, the above-level neurons ($u(n_j) > p$) are assigned to the neighbouring cluster which they have the smallest distance to. (See the appendix for an exact description of the clustering algorithm.) Finally all neurons are assigned to some cluster.

[2]Every non-border neuron is assumed to have four direct neighbours in this work, i.e. a two-dimensional regular grid is assumed.

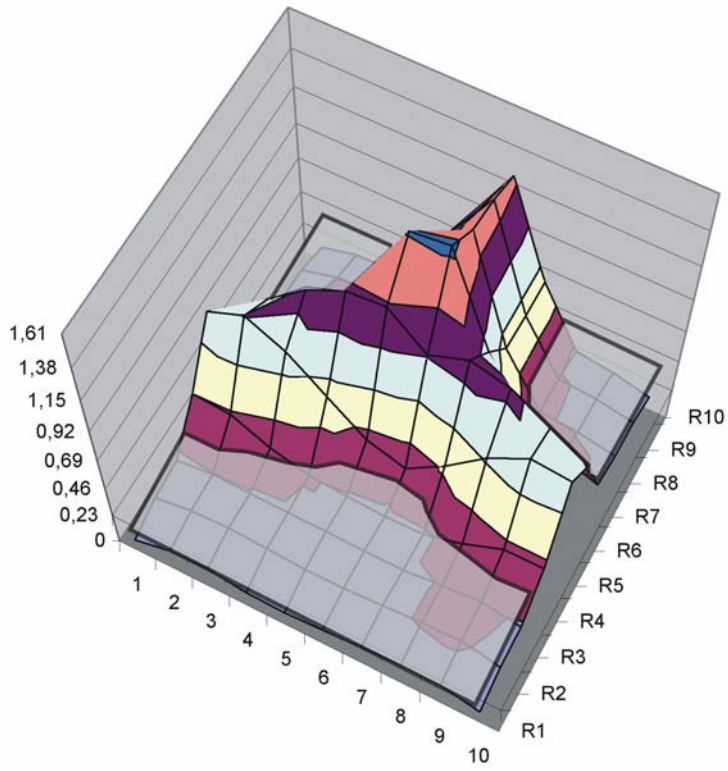


Figure 4 A U-Matrix of a SOM of size 100 in 3D view. Neuron 0 is in the top left corner. The water level of 0.3 is shown. All neurons found below that level are in the same cluster as their neighbours. The three clusters in this case can clearly be seen.

4. SOM for Sequential Data

Learning of sequences of patterns extends the problem. Information is not only contained in the patterns but also in the order they are presented in. Now, the SOMs are trained to a sequence $s = (s_1, ..., s_t)$ rather than a set of patterns, with $s_i \in G$ denotes a single entry of the sequence and t the current generally arbitrary length of the sequence. Presentation takes place one s_i after the other. To ensure the influence of the order of the presented patterns, each iteration of the SOM training algorithm must be influenced by previously presented pattern. These previously presented patterns are called the context $(s_2, ..., s_t)$ of the currently presented pattern s_1.[3] Various approaches have been taken in the literature. The integration of the context is usually done during winner selection so that if n_j is selected winner of s_1 it can be considered winner of the pattern within the context $(s_2, ..., s_t)$. The approaches differ in the choice of the distance function. As the context now plays a role in distance calculation, d is no longer a function of the neuron n_j and a single pattern x_i but rather of n_j and a whole sequence s: $d(s, n_j)$. This new distance function will be called recursive distance to distinguish it from the usual SOM distance function $d(x_i, w_j)$. However, the usual distance function is part of the recursive distance. The approaches presented here only differ in the choice of the recursive distance function[10].

[3] Sequences are here put in reverse notation, i.e. s_1 denotes the most recent entry.

Classical approaches apply the history of each neuron on its own in the sense that the previous activation of a neuron only plays a role in the calculation of the recursive distance of that neuron. Two early and simple forms are the Temporal Kohonen Map (TKM) [2] and the Recurrent SOM (RSOM) [12]. In TKM, the recursive distance is a weighted sum of the distance values of the neurons weight and all context patterns.

$$d_{TKM}(s,n_j)=\sum_{i=0}^{t-1}\eta(1-\eta)^i\left\|s_{i+1}-w_j\right\|^2$$

While the RSOM integrates differences:

$$d_{RSOM}(s,n_j)=\left\|\sum_{i=0}^{t-1}\eta(1-\eta)^i s_{i+1}-w_j\right\|^2$$

As influence of the activation decays over time (that is for patterns further back in the sequence) each neuron can be understood to be a leaky integrator of its own history. A limitation of these leaky integrator approaches is, that each neuron only has its own history available as context information when calculating the winner.

Newer approaches overcome this problem by a more explicit context representation. They will be called recursive approaches. Here, the map is self-referent as it learns to classify its own activity. The context information available to each distance calculation is not limited to one neuron as the whole activation history of the map is taken into account. Each neuron n_j is in addition to the weight vector w_j provided with a context vector c_j that is adjusted to express preference of the neuron to a particular szenario of previous activation. Two recursive approaches are the RecSOM[20] and the SOM for Structured Data (SOMSD)[8]. These approaches differ in the data type of this context vector.

For distance calculation, the context vectors are compared to a context representation C(s) of the sequence. In the RecSOM, the context representation stores (exponentially decayed) all distance values of the previous time step, leading to a |N|-dimensional vector with |N| being the number of neurons on the map.

$$C_{\mathrm{Rec}SOM}(s)=\left(e^{-d_{recursive}(s,n_i)}\right)_{i=1}^{|N|}$$

$d_{recursive}$ will be defined below and is recursive in the sense that it, in turn, includes C_{RecSOM}. The context representation at every time step thus requires the consideration of the recursively calculated recursive distance of every neuron on the map at preceding time steps, which leads to a very complex winner selection mechanisms.

Through a compressed representation the SOMSD ensures faster processing and a very straightforward interpretation of the context information. As can be seen later, the SOMSD context vectors can be directly used to recognize state transitions. The SOMSD essentially compresses this information by taking the grid coordinates g_c of

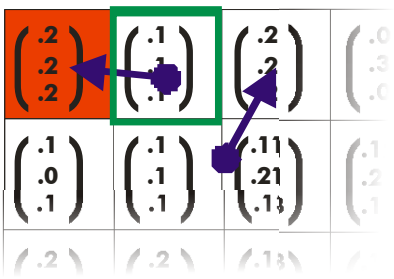


Figure 5 The context vectors take influence on winner selection. Two vectors are equally close to the pattern presented. Both their context vectors are displayed as arrows from the grid coordinates denoted by the context vector into the neuron that is owner of the context vector. A green box shows the winner of the previously presented pattern. Now, the neuron with a context vector closer to the grid coordinates of the previous winner is chosen as a winner.

the previous winner as the context representation:

$$C_{SOMSD}(s) = g_c$$

with

$$c = \arg\min_j \{d_{recursive}((s_2 \ldots s_t), n_j)\}$$

While c is the index of the winning neuron after presentation of s.

The recursive distance function applied in both cases is:

$$d_{recursive}((s_1, \ldots, s_t), n_j) = \alpha \|s_1 - w_j\|^2 + \beta \|C(s_2, \ldots, s_t) - c_j\|^2$$

α and β are scalars to determine context influence on distance calculation. C stands for the respective context representation CRecSOM or CSOMSD. Previous activation now influences the selection of winners (see Figure 5 for an illustration). Consequently context takes effect in both, weight adjustment and determination of representing neurons. Neurons now no longer represent a particular part of the input space, but they represent differently large portions of the input space given different contexts.

Adjustment of the context vectors in the recursive approaches is done through Hebbian learning analogously to the weight adjustment. The weight vectors of the winning neuron n_j and its neighbourhood are adjusted towards the currently presented pattern and the context vectors towards the context representation $C(s_2 \ldots s_t)$ (see Figure 6) In the particular case of the SOMSD, context vectors are updated analogously to the update of weights:

$$\Delta c_j = h(c,i)\eta(g_{c-old} - c_j)$$

Where $g_{c-old} = C_{SOMSD}(s_2 \ldots s_t)$ are the grid coordinates of the winner of the previous time step. Through this way of training recursive SOMs allow similar winner selection by similar sequences.

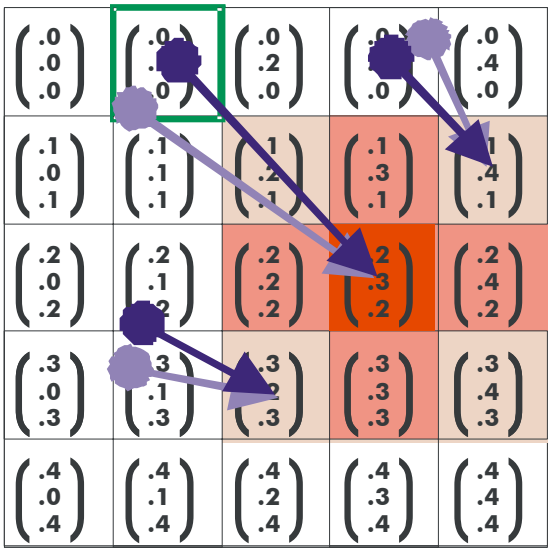


Figure 6 When a neuron is selected as winner. Context vectors in the neighbourhood are adapted towards the previous winner (green box). Old context vectors are visualized in light colour.

In a trained SOMSD, the context vector can be seen as representing likely transition between winners: As the context vector is part of the distance function, a neuron is more likely to be the winner of pattern s_1 if the winner of the previous pattern s_2 had coordinates similar to the context vector of that neuron. Because contexts of winners and their neighbours are adjusted in the direction of the previous winner one can assume that context vectors after training of a SOMSD represent transitions between patterns in the input sequence: if the context vector of n_j is close to the coordinates of n_i, one can assume that patterns activating n_j as a winner occur frequently after patterns activating n_i. The neuron n_j can be seen as represent-

ing s_1 with the (preferred) predecessor n_2. As n_2 in turn is representative of a symbol (ideally s_2) and a predecessor the whole sequence $s_1...s_t$ is recursively represented. For the extraction of Markov models that describe transitions in the sequence, the explicit representation of transitions in the context vectors is a major advantage of the SOMSD over the RecSOM.

5. Markov Models

In the experiments, sequences generated by stochastic processes will be used. Markov models[21] are a powerful and well understood formalism for stochastic temporal processes. They describe the sequence as a sequence of symbols from a finite alphabet A that are emitted whenever a state out of the set of states S is entered. Probabilities of state transitions and symbol emission are fixed and given by the model. It is an important property of Markov models that the probability distribution of following states depends on the current state of the process.

Formally, a Markov model (MM) is defined as:

- a set of states $Z = \{z_1,...,z_{|Z|}\}$,
- an output alphabet $K = \{k_1,...,k_{|K|}\}$,
- a function[4] p_t: $Z \times Z \rightarrow [0,1]$, that determines the probability of a transition between two states, i.e. $\sum_{z_i \in Z} p_t(z_i \mid z_h) = 1$ has to hold for all z_h,
- a function p_e: $Z \times K \rightarrow [0,1]$, that assigns each symbol in K a probability with that it is emitted when the model is in state $z \in Z$, thereby $\sum_{k_i \in K} p_e(k_i \mid z_h) = 1$,
- a function p_i:$Z \rightarrow [0,1]$ with $\sum_{z_i \in Z} p_i(z_i) = 1$ that assigns each state its probability of being the initial state of the process.

Figure 7 shows an example Markov model as it is used later. This gives a generative model of the sequences. A sequence can be generated by randomly choosing an initial state and then iteratively noting the emitted symbol and doing a state transition both following the probability distributions given for the respective state.

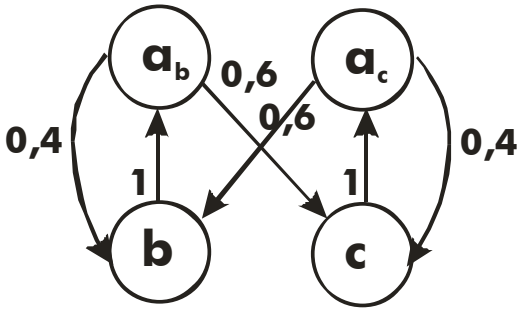


Figure 7 Example of a Markov model. The state names (without the indices) give the symbols that are emitted.

Consequently the model defines the probability $p(s)$ of a sequence $s = (s_1, ..., s_t)$ over K. The probability given a state sequence $y = (y_1...y_t)$ with all $y_i \in Z$ is given by:

$$p(s \mid y) = p_e(s_1 \mid y_1)...p_e(s_t \mid y_t)$$

and

$$p(y) = p_t(y_1 \mid y_2)...p_t(y_{t-2} \mid y_{t-1})p_t(y_{t-1} \mid y_t)p_i(y_t)$$

and therefore as a sum over all possible y of length t:

$$p(s) = \sum_{y \in Z^t} p(s \mid y)p(y) = \sum_{y \in Z^t} p_e(s_1 \mid y_1)p_t(y_1 \mid y_2)...p_e(s_{t-1} \mid y_{t-1})p_t(y_{t-1} \mid y_t)p_e(s_t \mid y_t)p_i(y_t)$$

The notion of equivalence of Markov models can be introduced using p(s). It is a weak notion of equivalence because it allows the models to be different with regard

[4] note that probability functions given by the model hold indices so that they can be recognized as such in calculations.

to their set of states and individual probabilities. However, it allows to compare MMs well, when understood as generative models: Two MMs are called equivalent if they assign the same probability p(s) to every possible sequence s out of their joint alphabets.
The general case of Markov models is called hidden Markov model (HMM) because hidden states are possible. Hidden states are states from which different symbols can be emitted. Several concepts that divide HMMs into subclasses are used in this work.

A Markov model is said to be deterministic if all probabilities are either 0 or 1. In that case, every sequence is uniquely determined by its start state y_t.

A model is called a finite order Markov model (FOMM)[9] if the next state transition probability can be uniquely determined by observing the last o $\in\mathbb{N}$ symbols emitted. Denote by $p(y_1 \mid s_2 \ldots s_t)$ the probability that state y_1 is entered after the presentation of $s_2 \ldots s_t$ and by $p(s_1 \mid s_2 \ldots s_t)$, the probability that s_1 is emitted after $s_2 \ldots s_t$. Then, by the definition of FOMMs $p(y_1 \mid s_2 \ldots s_t) = p(y_1 \mid s_2 \ldots s_{o+1})$ and $p(s_1 \mid s_2 \ldots s_t) = p(s_1 \mid s_2 \ldots s_{o+1})$ hold.
Note that o<t is assumed for all applications of FOMMs used here. The first o symbols of a sequence are assumed to be fixed.
Any FOMM can be transferred to an equivalent Markov model with states $z_{s_1 \ldots s_o}$. Each state stands for one particular preceding sequence ($s_1 \ldots s_o$) with $s_i \in K$. All non-zero transitions are given as $p_t(z_{s_1 \ldots s_o} \mid z_{s_2 \ldots s_{o+1}})$ corresponding to the next symbol s_1. Each state $z_{s_1 \ldots s_o}$ forces the emission of the first symbol of the sequence it denotes, s_1. All emission probabilities are thus $p_e(s_1 \mid z_{s_1 \ldots s_o}) = 1$. Hence, FOMM states can be uniquely determined by the last o symbols. Consequently FOMMs have at most $|K|^o$ states with at most $|K|$ possible following states differing with respect to the symbol s_0 emitted. Given the set of states, such an FOMM can be described by a k×o transition matrix M containing the probabilities $p_t(z_{s_1 \ldots s_o} \mid z_{s_2 \ldots s_{o+1}})$. Figure 8 shows the Markov model from Figure 7 with states renamed to fit the new naming conventions.

The sequences generated by MMs as presented here have a finite alphabet. For the experiments, each element of the alphabet is a two-dimensional vector. Gaussian noise is added to all components to extend the models to real valued vectors instead of a limited set of symbols. This allows to observe the handling of infinitely many possible input patterns and to mimic natural data (Table 1 gives an overview over the steps during sequence generation.). Data like this might stem from a system that can be in a fixed number of discrete states (e.g. operation modes of a machine) that are expressed in values (e.g. the machine's performance) which are slightly modified by other variables (e.g. environment).

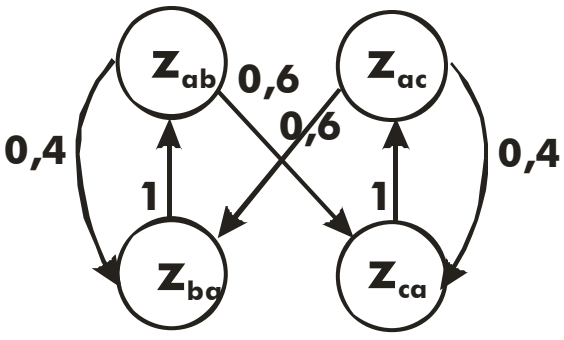


Figure 8 The Markov model from Figure 7 is an order 2 FOMM in standard form. States were renamed.

variable	symbol name	symbol vector	noise	pattern
s_1	a	(0;0)	(0.02;-0.01)	(0.02;-0.01)
s_2	b	(1;0)	(-0.03;0.13)	(0.97;0.13)
s_3	a	(0;0)	(0.04;0.02)	(0.04;0.02)
s_4	c	(0;1)	(-0.01;0.01)	(-0.01;1.01)

Table 1 Example of the steps in sequence generation.

6. Measuring the performance of SOMs processing sequences

There is no default solution for evaluation of the performance of unsupervised learning. As by definition in unsupervised learning no function is given that is to be trained, several – also unpredicted – results might be a good solution to the training task. SOMs are frequently used for visual inspection. Therefore the desired results are not formulated numerically and a formal notion of good performance is far out of reach. However, several formal criteria have been established for evaluation of SOMs: The conservation of important parts of the topology of the input space, the quantization error, which is a measure for the similarity of patterns assigned to the same neuron, as well as the specialization of neurons to relevant classes of the input space.

The notion of the quantization error can be directly transferred to sequence learning: Good quantization has taken place if, whenever n_j becomes winner after presentation of a sequence s, it is likely that the last k patterns of s are similar to a sequence r that is characteristic for n_j. Thereby the prototypic sequence r is called the receptive field of n_j and it is defined as the average sequence of a fixed length k that leads to the activation of n_j as a winner. This receptive field for sequence processing SOMs was introduced by Voegtlin[20] and can be used to measure the performance of sequence learning. The winner selection after presentation of sequences can be considered a quantization process: The space of all possible sequences is divided into classes by the neurons they activate as a winner. The quantization error of these classes can be taken as a quality measure.

Also, for each entry of the receptive field, the standard deviation over all sequences presented can be measured. These sequence quantization errors allow a judgement on how salient the activation of a particular neuron is with regard to past sequence entries. The sequence quantization error can be taken from all the mentioned sequence learning SOMs and allows a comparison of the amount of temporal information learned. It can also be applied to the SOM itself. This would be a baseline condition showing how predictable the context is from the current pattern alone. Figure 9 presents a comparison of the SOM to the SOMSD and other extensions for sequence processing. Considerable improvement through application of recursive structures have been shown for various architectures in the case of symbolic data when considering 30 past time steps[16]. This shows that recursive SOMs are able to divide a presented sequence into classes of similar sub-sequences.

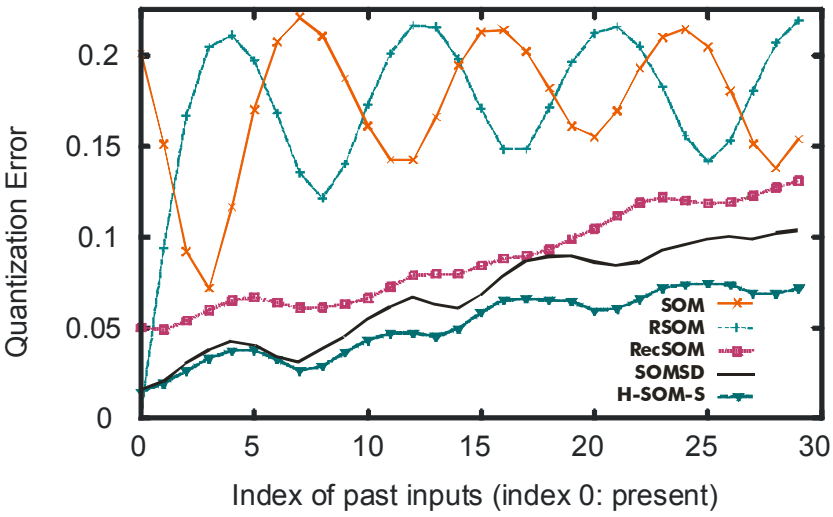


Figure 9 Sequence quantization errors of the SOM, the SOMSD and other SOM extensions (100 neurons each) when trained on the Mackey-Glass continuous time series. The H-SOM-S is an extension of the SOMSD to a hyperbolic grid. Adopted from [17].

Similarly and more directly, the context error can be measured by the quantization of contexts. This is an additional performance measure specific to the SOMSD. It is calculated as the average difference of each winning neuron's context and the previous winner neuron's grid coordinates. As the context ideally selects the previous winner, a low context error indicates that context has taken a considerable influence on the training. The context error has been developed for this project to determine if the SOMSD has reached a stable state with respect to the contexts. This is the case when the context error does not considerably change between training cycles.

7. Aim of the project

The aim of data mining on sequences is to identify processes that might underlie the generation of the data. The SOMSD produces a representation of the input sequence. It can be used as a data mining tool, if that representation allows interpretation which leads to an abstract description of the sequence. The following experiments are done to show that this is in principal possible: Sequences generated using Markov models are trained to SOMSDs. Then, extraction algorithms are applied that extract a Markov model from the resulting SOMSDs. By comparison of the extracted model with the one used for generation, the SOMSD's ability to learn the relevant temporal structures can be assessed.

Results in [16] show that the SOMSD topology reflects the dynamic of sequences when discrete sequences are trained. Here, continuous data is presented in order to extend the range of possible applications. Continuous data requires clustering in order to find an abstract representation. For this reason, the U-Matrix is used in model extraction.

The focus in this project is on finite order Markov models (FOMMs), in which the current state can be determined by a finite number of previously emitted symbols. Generalization to all HMMs will be considered but not fully assessed.

Extracting a Markov model involves the identification of symbols and states and the identification of possible transitions and their probabilities. The information needed for these tasks can be obtained from a recursive SOM using:

a) The arrangement of the symbol representation (weight vectors). Due to the above mentioned salience of proximity on the SOM, clustering should allow the determination of an alphabet $K = \{k_1,...,k_k\}$.
b) The context representation (context vectors). Recall from section 4, that a context vector determines a preferred predecessor neuron. If neurons can be associated with states, this relation can be understood as state transition.
c) The arrangement of context representation. Context vectors are adjusted analogously to weight vectors. It will be seen in the experiments that clusters of context vectors arise. They might subdivide weight clusters. This subdivision ideally distinguishes different states that emit the same symbol, a task which cannot be done from weight vectors alone.

If FOMMs are to be extracted, exploiting a) and b) is sufficient to allow for a simple extraction algorithm that counts transitions among weight clusters by considering every single neuron. Such an extraction algorithm will be introduced in the following section, before results from experiments using this algorithm are presented.
The arrangement of context vectors c) might be used to allow interpretation on groups of neurons with similar predecessors instead of considering the context vector of every single neuron. An extraction algorithm based on weight and context clusters is outlined in section 10.
The experiments are done to see, whether in a particular setup, FOMM extraction is possible. That is, if (1) the general connectivity of the underlying Markov-Model can be found and (2) transition probabilities are comparable. Also, (3) the stability of the extraction mechanism towards stronger noise is evaluated as FOMM extraction from noisy data is a considerably harder task.

As weights and contexts are treated similarly in SOMSD training, it can be assumed by analogy that when a U-Matrix clustering is applied to context vectors, clusters of similar predecessors can be determined. In section 10, considerations will be made on whether these clusters can be used to describe Markov-Models. In section 11 the possibility of extracting non-FOMM HMMs will be assessed.

8. FOMM Extraction

Finite order Markov models can by their definition be extracted from the data. This task is trivial, when symbolic data is given. However, for continuous input a classification facility is needed to obtain a limited number of states. The SOM in combination with the U-Matrix has been successfully applied to classification tasks. Here, an extraction algorithm based on the SOMSD and the U-Matrix is presented and applied. It uses the U-Matrix clustering to identify symbols and the context representation specific to the SOMSD for state transitions.
The U-Matrix clustering algorithm from section 3 applied to the neuron weights returns a set of classes (clusters) $\{W_1...W_m\}$. These classes can be used to identify symbols. If clustering succeeds, the average weight vector of each W_i might be a good approximation to symbols. These symbols will be denoted $[W_i]$ and for neurons n_j, $[n_j]=[W_i] \Leftrightarrow n_j \in W_i$ is defined. Experimental results will show that there is hardly any ambiguity in the assignment of symbols to clusters if noise is sufficiently low.

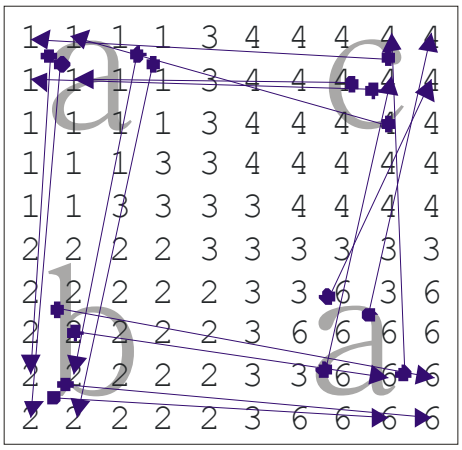


Figure 10 A SOMSD trained to a long sequence of 'abac...' with the cluster number for each neuron and context vectors for 16 corner neurons.

Considering the way context vectors are defined in a SOMSD, it is justified to assume that state transitions can be re-obtained from the information they contain. During winner selection, a pattern s_1 is preferred if the winner of the previous pattern s_2 had coordinates similar to the context vector of that neuron. During training, context vectors are adjusted to ensure that if the context vector of n_j is close to the coordinates of n_i, one can assume that patterns activating n_j occur frequently after patterns activating n_i. This can easily be replicated in experiments with simple deterministic sequences (see Figure 10).

It has been stated above (and clearly seen in plotted SOMs) that frequent presentation of similar patterns leads to more neurons adopted to these patterns. This can be used to extract the transition probabilities as relative frequencies: If there are many neurons in W_4 with context vectors pointing to neurons in W_3 (recall that all neurons of a cluster are adjacent on the grid) then it can be assumed that the transition $[W_3] \rightarrow [W_4]$ has been frequent in the training sequence. If the count of these neurons is high relative to the total number of neurons in W_3 then this transition has to be represented by a high probability in the model. Note that these are cluster transition probabilities and not yet state transition probabilities which require further assumptions and inference.

The algorithm extracts an FOMM in standard form for a fixed and previously chosen order o. That is, it determines for each state $z_{s_2 \dots s_{o+1}}$ the probability of transitions to all possible states $z_{s_1 \dots s_o}$. Thereby, sequences that precede the activation of each neuron n_j are found by recursively determining possible preceding symbols by their context vectors. In order to be able to calculate probabilities through relative frequencies, frequency measures for every sequence s of length o+1 are accumulated over all neurons in a(s).
If a unique sequence s could be determined that precedes n_j, it would make sense to assign frequency measure 1 to it. However, as contexts do not necessarily match the grid coordinates of one neuron, several preceding sequences are possible. The shareOfFrequency function reflects the ambiguity in the selection of the predecessor neuron through the context vector. It distributes the volume of 1 among (here, the four closest) possible predecessor neurons according to the difference between their grid coordinate and the context vector. The multiplication of shareOfFrequency-values during recursion is done so that for each neuron n_j the a(s) values for all possible preceding sequences s is increased. The total amount of the incrementations adds up to 1.

In detail, the algorithm runs as follows (Recall that $[n_j]$ the symbol denoted by the weight cluster n_j lies in.):

extract()
    for all sequences s ∈ K^{o+1} set a(s) = 0
    for every neuron n_j
        trace(n_j, [n_j], 1)
    calculateProbabilities()

//sub-functions:
trace(neuron n_j, sequence s, frequency f)
    if sequence s has length o+ 1
        //save result and end recursion
        set a(s)= a(s)+ f
    else
        P = {x | x is a possible predecessor of n_j according to the context of n_j}
        for all n_p∈P
            trace(n_p,s •[5] [n_j],f*shareOfFrequency(n_j, n_p))

shareOfFrequency(neuron n_j, neuron n_p)
    return a fair share of 1 for each possible predecessor n_p of n_j and 0 otherwise

calculateProbabilities()[6]
    for all sequences s = (s_1, ..., s_{o+1}) with a(s) ≠ 0

$$p(z_{s_1 \dots s_o} \mid z_{s_2 \dots s_{o+1}}) = \frac{a(s_1 \dots s_{o+1})}{\sum_{k \in K} a(k, s_2 \dots s_{o+1})}$$

It can be shown that for each standard form FOMM exists a SOMSD for which the above presented algorithm extracts approximately this FOMM. Here, all probabilities are assumed to be in ℚ.
This SOMSD can be constructed as follows:

- Let $q_{s_1 \dots s_{o+1}}$ denote the smallest common denominator of all given $p(z_{s_1 \dots s_o} \mid z_{s_2 \dots s_{o+1}})$.
- Let {W_i | i∈K} be weight clusters of the SOM for each symbol in K .
- Let each W_{x_1} have subclusters $W_{s_1,[s_1 \dots s_{o+1}]}$ for each possible state transition $z_{s_1 \dots s_o} \mid z_{s_2 \dots s_{o+1}}$.
- Let each $W_{s_1,[s_1 \dots s_{o+1}]}$ contain $q_{s_1 \dots s_{o+1}} p(z_{s_1 \dots s_o} \mid z_{s_2 \dots s_{o+1}})$ neurons with the weight vector equal to s_1 and the context vector equal to the grid coordinates of some $W_{s_2,[s_2 \dots s_{o+1},s_{o+2}]}$ with arbitrary s_{o+2}.

For the extraction algorithm, the shareOfFrequency function has to be chosen in a way that assigns 1 to the predecessor neuron closest to the context vector. (Note that in the constructed case all predecessor choices are unambiguous because each context vextor is equal to an existing grid vector). A so constructed SOMSD will lead to the extraction of exactly the encoded FOMM in standard form.

[5] • refers to sequence concatenation.

Note that encoding and extraction here makes use of the fact that only one symbol is emitted at each state. In particular non FOMM-HMMs cannot be decoded with this mechanism.

As the SOMSD is an unsupervised learning mechanism, the order of the FOMM that is to be extracted is not given. The order might possibly be determined on inspection of the results of several extraction processes from the same SOMSD trying different orders. By definition, FOMMs of a fixed order o can be described as an FOMM of any order higher than o. Consequently, extracted models with a fixed order larger than o should be almost equivalent to models of lower order.

The finding that a SOMSD that allows extraction of the underlying FOMM can always be constructed does not mean that it can always be obtained from the data through SOMSD training. The limitations of SOMSD training are left to be evaluated – preferably in empirical analysis as the dynamics of non-trivial SOMs cannot be fully accessed analytically. The approach relies on the number of neurons in one cluster as a measure of frequency of presentation of patterns from that cluster. However, this correlation has shown to be not linear as it is assumed here [15]. For the SOM a magnification factor can be determined that allows proper interpretation of cluster sizes, but the mechanism to determine such a magnification factor does not directly transfer to the SOMSD as the training dynamics are different due to the implementation of context vectors. Therefore the numerical values of the FOMMs extracted in the following cannot be expected to perfectly match the FOMM used to generate the sequence. However, they should reflect the general connectivity and the general form of probability distributions.

In section 10, an example is given of how FOMM extraction can be done using the context topology of the SOMSD rather than evaluating context vectors on a single neuron basis as it is done in the algorithm presented in this section.

9. Experiments on FOMM extraction

In the following experiments, a SOMSD is trained to sequences generated by simple FOMMs and it is tested whether they can be extracted using the method introduced above.

The Markov models used to generate the sequences in experiment 1 had three states and three symbols were used. The symbols were three different vectors in $\mathbb{R}^2$. In this text, they will be abbreviated by a= (0.0,0.0), b= (1.0,0.0) and c= (0.0,1.0). There was always the same symbol emitted at each state. That means, no hidden states were present. The possible transitions were the same[7] in all sequences, while transition probabilities were varied. The model was designed to be of finite order 2. It can

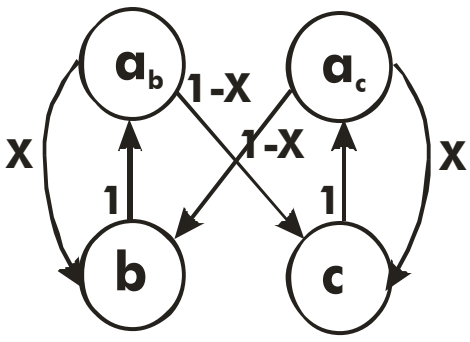


Figure 11 Markov model that was used for training.

[7] The only exception are two deterministic cases where additional transition probabilities become 0.

therefore be given as transition probabilities in the standard form of the FOMM (as depicted in Figure 11). The model was built such that every second symbol is an 'a'. The models differ in the parameter X, that determines the probability with which 'a' is followed by the same letter that preceded the 'a'.

As a network, a square SOMSD with Euclidian grid has been used. The size of the map was 100 or 196. Weight vectors were chosen from $\mathbb{R}^2$ with each component randomly initialized in [0,1]. Context vectors, also from $\mathbb{R}^2$ were initialized with the grid coordinates of the neurons they were assigned to.

Training has been done by successively presenting the patterns s_t to s_1 (reverse to keep consistency with the above notation). Presentation of one sequence element s_i includes finding its winner, applying the neighbourhood-function and calculating the adjustment of all neurons n_j as their Δw_j and Δc_j. A predetermined number of presentations of the whole sequence (usually 15000) were done. This number was large enough so that no relevant changes in the important indicators could be observed towards the end of the training. The three parameters for SOMSD training are (see section 2): α/β ratio for context influence, the σ coefficient of the neighbourhood-function that determines the size of the neighbourhood and the learning rate η that influences the overall magnitude of the adjustments made. All these parameters were reduced during training. α was chosen higher in the beginning of the training to allow a primary arrangement following the weight vectors, σ was initialized with 5 (as a function of |N|, the number of neurons: $0{,}5\sqrt{|N|}$) and later reduced down to 0.9 leading to a stronger adjustment of a wider area around the winner in early phases of the training, when a broad global arrangement has to be formed. The learning rate was hardly decayed (from 0.02 to 0.01). It had to be low in late phases of the training to prevent big changes from erasing previously learned information encoded in weight and context vectors. The α/β ratio was chosen depending on the size of the SOMSD to compensate for higher absolute values of context vectors on larger maps. Here, it was decayed from 1.0 (no context influence) to 0.97 (Figure 13)
To evaluate the progress of the training, several values were regularly calculated and printed (Figure 12):

- The number for neurons winning at least one pattern of the sequence. (# w for short)
- The context error: average variance of a winners context and the coordinates of the previous winner (CE for short).
- The context radius: average distance between a neuron's coordinates and its context. (CR for short)
- The number of weight clusters on a U-Matrix. (#c for short)

Also, the U-Matrices were regularly visualized so that the development of the SOM representation as a whole could be judged.

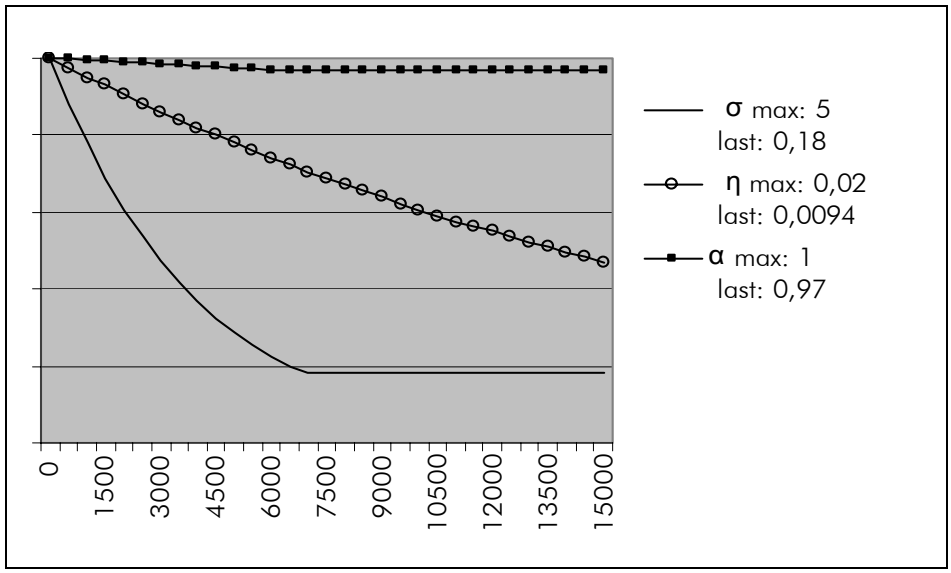


Figure 12 Properties of the SOMSD measured during training. Number of active (winning) neurons (#w), context error (CE), context radius (CR) and number of weight clusters (#c).

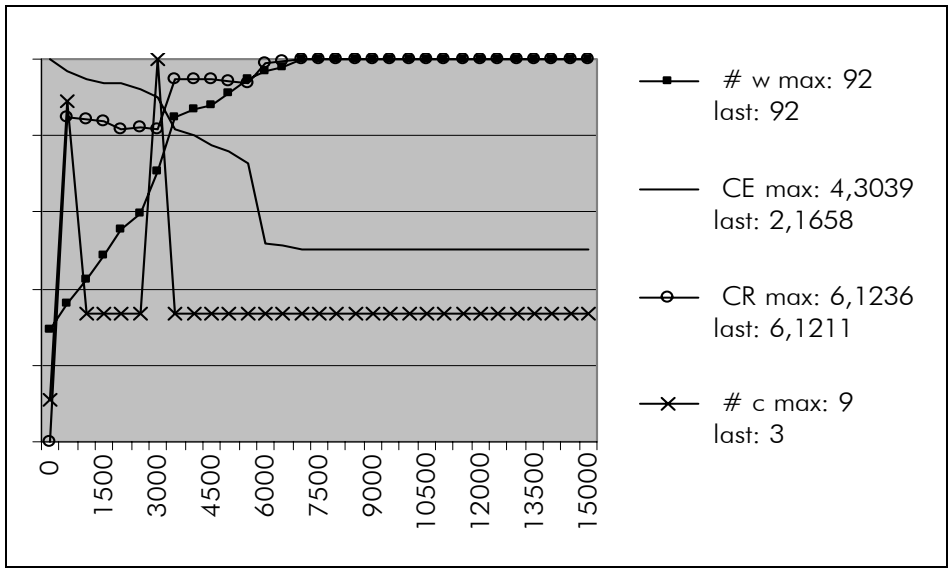


Figure 13 Decay of parameters to the SOMSD training algorithm over the 15000 presentation of the sequence of length 1000.

Figure 12 shows that in this particular run of the training, the final state has been nearly reached after 8500 presentations of the sequence and that after that the SOMSD was
relatively stable. Major changes of the number of active neurons (#w), the context error (CE) and the context radius (CR) frequently appeared at the same time. Also, they reached their final level simultaneously. This suggests that these values reflect the state of the training and that training had the wanted effects: Nearly the whole SOMSD was active and context vectors approximately reflected possible transitions.

Weight clusters obtained from the U-Matrix are an important basis for the extraction algorithm. In training, weight vectors were adjusted to the currently presented pattern. Due to a high α value, the arrangement on the SOM occurred primarily with respect to the weight vectors. Figure 14 shows a U-Matrix as it is typical for the experiments. Note that the overall layout and the size ratio of the clusters were the same regardless which parameter X was used in the FOMM because the overall number of occurrences of each symbol was the same in all types of sequences that were generated. For four neurons s,t,u and v the weight vector and the number of patterns won are displayed in the figure. They show the characteristics typical of the U-Matrix. The neurons n_u and n_v, which lie in a valley, have weight vectors close to the symbol (input symbol was a= (0.0,0.0)+ noise) and win a considerable amount of patterns (1000 are presented). The neurons n_s and n_t, which have higher U-values, have weight-vectors different from all symbols. A larger difference lies between them – although they are neighbours – and they win only few patterns.

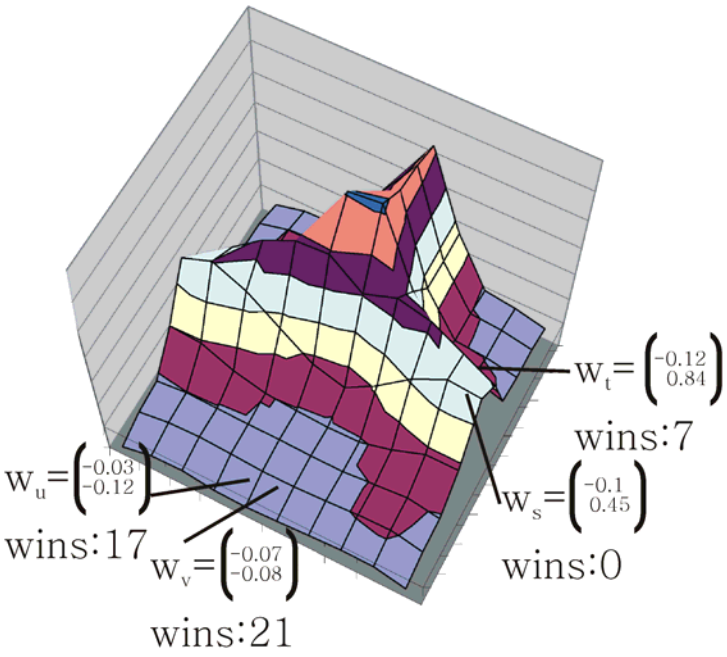


Figure 14 U-Matrix with the weight vectors and the number of patterns won given for example neurons.

For the extraction, two design choices had to be done. First, to allow a good comparison of the extracted FOMMs to the FOMMs used for sequence generation, identification of symbols out of clusters was necessary. This was done prior to the counting of transitions by labelling them directly with the names of the symbols used for sequence generation, if they were sufficiently close to one of them. This of course was only possible because the symbols were known in advance, which should not be assumed in the general case. If the symbols are unknown, some reasonable choice for their identification such as the mean values within clusters could be made. In the experiments described here, the input symbols were used. However, assignment was unambiguous in all cases so that an extraction without these a-priori-labels would have led to the same results. In assigning these labels, sometimes two neighbouring clustered were unified to represent one symbol. However, in all cases cluster centers of unified clusters were very similar so that this would have been possible without a-priori-labels as well. Their Euclidian distance was smaller than 0.1 while the distance of symbols during generation was at least 1.0. Secondly, the function shareOfFrequency had to be implemented.

model	transition	input	extracted
X=0.0	p(a|ba)	0.0	
	p(b|ba)	0.0	
	p(c|ba)	1.0	1.00 (22.72)
	p(a|ca)	0.0	
	p(b|ca)	1.0	1.00 (21.44)
	p(c|ca)	0.0	
X=0.1	p(a|ba)	0.0	0.01 (24.07)
	p(b|ba)	0.1	0.08 (24.07)
	p(c|ba)	0.9	0.91 (24.07)
	p(a|ca)	0.0	
	p(b|ca)	0.9	0.81 (25.0)
	p(c|ca)	0.1	0.19 (25.0)
X=0.2	p(a|ba)	0.0	
	p(b|ba)	0.2	0.30 (24.0)
	p(c|ba)	0.8	0.70 (24.0)
	p(a|ca)	0.0	
	p(b|ca)	0.8	0.80 (23.03)
	p(c|ca)	0.2	0.20 (23.03)
X=0.3	p(a|ba)	0.0	0.01 (25.0)
	p(b|ba)	0.3	0.31 (25.0)
	p(c|ba)	0.7	0.68 (25.0)
	p(a|ca)	0.0	0.00 (24.91)
	p(b|ca)	0.7	0.66 (24.91)
	p(c|ca)	0.3	0.34 (24.91)
X=0.4	p(a|ba)	0.0	
	p(b|ba)	0.4	0.38 (23.38)
	p(c|ba)	0.6	0.62 (23.38)
	p(a|ca)	0.0	
	p(b|ca)	0.6	0.52 (24.73)
	p(c|ca)	0.4	0.48 (24.73)
X=0.5	p(a|ba)	0.0	0.04 (22.99)
	p(b|ba)	0.5	0.55 (22.99)
	p(c|ba)	0.5	0.41 (22.99)
	p(a|ca)	0.0	0.01 (23.83)
	p(b|ca)	0.5	0.55 (23.83)
	p(c|ca)	0.5	0.44 (23.83)
X=0.6	p(a|ba)	0.0	0.00 (20.86)
	p(b|ba)	0.6	0.68 (20.86)
	p(c|ba)	0.4	0.32 (20.86)
	p(a|ca)	0.0	0.01 (25.42)
	p(b|ca)	0.4	0.32 (25.42)
	p(c|ca)	0.6	0.67 (25.42)
X=0.7	p(a|ba)	0.0	0.04 (23.0)
	p(b|ba)	0.7	0.66 (23.0)
	p(c|ba)	0.3	0.30 (23.0)
	p(a|ca)	0.0	
	p(b|ca)	0.3	0.31 (25.0)
	p(c|ca)	0.7	0.69 (25.0)
X=0.8	p(a|ba)	0.0	0.01 (25.78)
	p(b|ba)	0.8	0.78 (25.78)
	p(c|ba)	0.2	0.21 (25.78)
	p(a|ca)	0.0	0.01 (21.35)
	p(b|ca)	0.2	0.24 (21.35)
	p(c|ca)	0.8	0.75 (21.35)

Table 2 FOMM extraction using weight clusters and context vectors. Using differend model parameters.

The following implementation was used:

shareOfFrequency(neuron n_j, neuron n_p)
return $\left(1-\left|c_{1j}-g_{1p}\right|\right)..\left(1-\left|c_{dj}-g_{dp}\right|\right)$

With d being the dimensionality of the grid (here: 2) and c_{1j} for example denoting the first component of c_j. Among all nearest neighbours (here given as n_p) of n_j's context vector the frequency volume of 1 is shared. The above function does this according to their component-wise grid distance from the neighbours. It reflects that the ambiguity of predecessor selection is higher, if the context vector lies in the middle between grid coordinates than if it is relatively close to one of them.

Table 2 shows selected transition probabilities extracted from various SOMSDs. Extraction has been done using the algorithm from the previous section with the order selected to be o=2. In the table, the probabilities are named in an abbreviated manner: for example, p(c|ba) stands for the probability of 'c' occurring after 'b', 'a' that is $p(z_{ca} \mid z_{ab})$ in the FOMM standard form. Only those transitions depending on the parameter X are shown. All other transitions were deterministic, as expected, and clearly close to 0 or 1. The frequency value of each state of the standard form FOMM is given in brackets. The frequency value of a state is defined to be the sum of the frequency values over all sequences which are tracked to calculate the probabilities leaving that state. This is the denominator in the term used to calculate the transition probabilities.

$$p(z_{s_1 \ldots s_o} \mid z_{s_2 \ldots s_{o+1}}) = \frac{a(s_1 \ldots s_{o+1})}{\sum_{k \in K} a(k, s_2 \ldots s_{o+1})}$$

It reflects how often the state $z_{s_2 \ldots s_{o+1}}$ is reached relative to the total number of neurons |N|. In the table, these values are always close to 25, which shows that transitions out of the given state make up around 25% of the context vectors (as

noise	transition	input	extracted
0.1	p(a|ba)	0.0	0.01 (22.81)
	p(b|ba)	0.4	0.42 (22.81)
	p(c|ba)	0.6	0.57 (22.81)
	p(a|ca)	0.0	0.01 (24.35)
	p(b|ca)	0.6	0.59 (24.35)
	p(c|ca)	0.4	0.4 (24.35)
0.2	p(a|ba)	0.0	
	p(b|ba)	0.4	0.49 (25.0)
	p(c|ba)	0.6	0.51 (25.0)
	p(a|ca)	0.0	
	p(b|ca)	0.6	0.6 (24.0)
	p(c|ca)	0.4	0.4 (24.0)
0.3	p(a|ba)	0.0	
	p(b|ba)	0.4	0.4 (27.28)
	p(c|ba)	0.6	0.6 (27.28)
	p(a|ca)	0.0	
	p(b|ca)	0.6	0.44 (20.17)
	p(c|ca)	0.4	0.56 (20.17)
0.4	p(a|ba)	0.0	0.1 (19.28)
	p(b|ba)	0.4	0.24 (19.28)
	p(c|ba)	0.6	0.66 (19.28)
	p(a|ca)	0.0	0.09 (20.61)
	p(b|ca)	0.6	0.39 (20.61)
	p(c|ca)	0.4	0.52 (20.61)
0.5	p(a|ba)	0.0	0.98 (18.0)
	p(b|ba)	0.4	0.02 (18.0)
	p(c|ba)	0.6	
	p(a|ca)	0.0	
	p(b|ca)	0.6	
	p(c|ca)	0.4	

Table 3 Extraction of FOMMs with varied noise level (given as the standard deviation of Gaussian noise).

there are 100 neurons). This is the expected value considering the input FOMM. The frequency of the state is given here because it can be used as a quality measure of the extraction. If it strongly deviates from the value that is suggested by the overall frequency of occurrence of the state in the input model, one can assume that either probabilities are somewhat biased or frequency mass is wasted on states that do not occur in the input model. The frequency can also be used to rule out states that might not be present in the original model.

Note that all extracted FOMMs can clearly be distinguished. An extraction of finite order Markov models from SOMSD context vectors using a U-Matrix clustering has thus been shown to be possible for small models.

Noise plays an important role in these experiments. The identification of states of an FOMM causes no problems, if no noise is present, that is, if input data is symbolic. As noise increases, the assignment of input patterns to states becomes less clear. The ability to cope with noise is an important criterion for a data mining mechanism, because noisy data contains less obvious information. One way to cope with noise is taking into account context information. This is possible because particularly in an FOMM, the probability of every symbol to be emitted next is fixed for a given context. In a SOMSD both, the current symbol and the previously observed ones are taken into account in winner selection as the recursive distance function incorporates weight and context vectors. Therefore a noisy current symbol can be compensated by a well matching context. For example, in the models used here 'a','c' is always followed by an 'a'. A well trained SOMSD recognized a relatively noisy 'a' as such

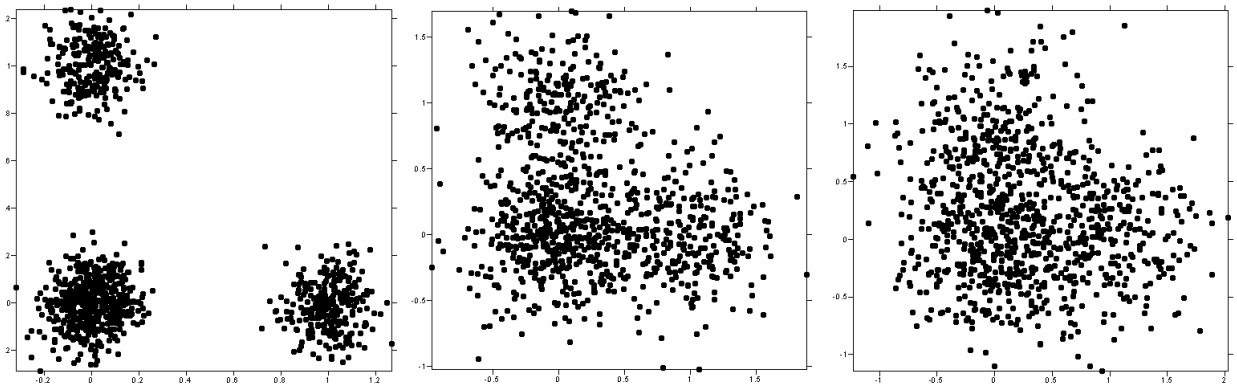


Figure 15 Plot of patterns from the input space in sequences generated with noise of standard deviations of 0.1, 0.3 and 0.4

because context vector configuration leads to the respective winner selection. This strategy is known from human high level information processing as top-down processing[3].

Figure 15 shows a 2D plot of the patterns from sequences with Gaussian noise of 0.1, 0.3 and 0.4 standard deviation. Clustering solely based on the patterns might not allow to extract the correct FOMM, but the above algorithm does. The result of an FOMM extraction are shown in Table 3. The three symbols were identified and the count of the transition probabilities are reflected better at lower noise levels. Even with a noise standard deviation of 0.3 the model could clearly be recognized. At a noise level of 0.5 however, the model was no longer present at all. The frequency value of some states was lower in experiments with higher noise which suggests a lower correctness of the model, because necessarily a larger amount of frequency value was assigned to other states. However, the non-zero transition probabilities could be recognized clearly.

Good extraction at higher noise levels is due to the context arrangement. An ambiguous pattern can be recognized by its context and therefore assigned to the right cluster. Note that this setup was highly sensitive to the alpha parameter used. The FOMMs from Table 3 were extracted from a SOMSD trained with a constant alpha of 0.97.

The experiments in this section show that FOMM extraction from SOMSDs using the U-Matrix and the algorithm from section 8 is possible even if noisy input is given. The method shown in the following section demonstrates a method that uses context clusters for the identification of transition probabilities.

10. Experiments on the arrangement of context vectors

Through the development of recursive SOMs an important feature of the SOM - the topological representation - has been transferred to contexts. The topological arrangement of context vectors has been listed among the features of the SOMSD that hold information that can be useful for Markov model extraction in section 7. However, it was not used in the algorithm presented in section 8 and the above mentioned experiments. The FOMM extraction algorithm is only based on a) the arrangement of weight vectors and b) the transitions established among these weight clusters by context vectors. During SOMSD training, contexts are handled analogously to weights in winner selection and adaptation. It is therefore justified to try to find context clusters using the U-Matrix in the same way as it has been done before with weight vectors. Clearly, well expressed context clusters are groups of common predecessors. These groups allow a more abstract description of the dynamics of the sequence because transitions are established abstracting from the context vectors of single neurons. Such a more abstract representation might allow a clearer, computationally less demanding description that might even be directly used as visualization. Also, further information might be extracted such as the order of the FOMM.

The parameters α and β decide on the degree to which context vector distances are taken into account during winner selection. It became evident during the experiments that whether or not salient context clusters occur, depends on the choice of α and β. The SOMSDs analyzed in this section have been trained on the same model as those

in the previous section. With the difference that through a lower α[8] (decayed from 0.97 to 0.93) the context vectors had a larger influence on winner selection during training.

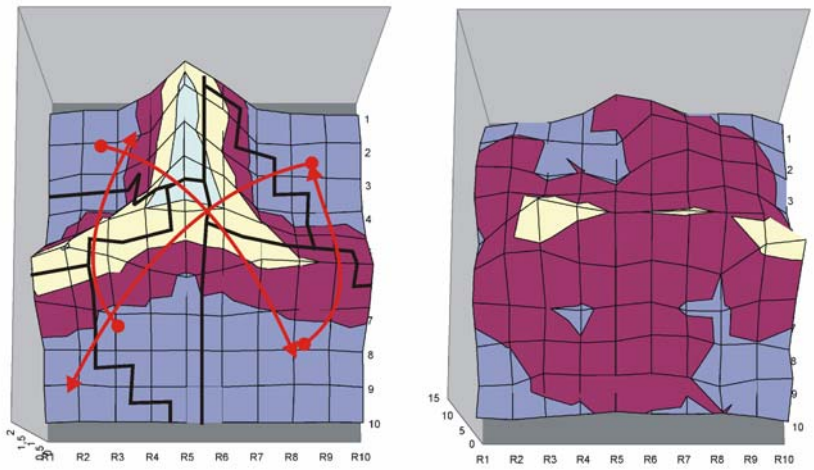


Figure 16 Weight (left) and context (right) U-Matrix of a SOMSD. In the weight U-Matrix sub-cluster boarders and some transitions (red arrows) are shown.

Figure 16 shows the U-Matrix calculated from the weight vectors (left) and from the context vectors (right). The calculation of a U-Matrix from context vectors can be done in the same way as from weight vectors. With the only difference that now, the U-value is the sum of the distances to context vectors of neighbouring neurons. The weight U-Matrix shows in addition to the U-Matrix landscape the cluster borders and some of the average context vectors. Figure 11 in section 9 shows the standard form of the FOMM used to generate the sequence that was trained here (X = 0.2). Note the obvious similarity between the transitions depicted on the U-Matrix and the FOMM diagram. Apparently, context clusters can be used to visualize state transitions in Markov models.

As FOMMs are considered here, it can be assumed that always the same symbol is emitted at each state. Therefore the notion of sub-clusters of weight clusters is introduced here. A sub-cluster is the intersection of a weight and a context cluster. All neurons that lie within the same weight cluster and the same context cluster belong to one sub-cluster. How FOMM states can be identified from sub-clusters will be demonstrated on examples based on the experiments.

Figure 17 displays weight clusters and sub-clusters of the same SOMSD. The upper part shows how sub-clusters are established. The weight and the context cluster are given for each neuron: The symbols 'a', 'b' and 'c' denote the weight clusters for the respective symbols and the numbers 1-6 denote context clusters. The weight clusters are shown in different colours, while sub-clusters of the same weight-cluster are distinguished by varying lightness values. Arrows depict average transitions. Starting points of the arrows are given by the average context vector of a sub-cluster. The tip of each arrow points into the center of that sub-cluster. The occurrence of sub-clusters in Figure 17 clearly demonstrates that topological arrangement of context vectors is possible. It also shows that information on the dynamics of the sequence is reflected in the topological arrangement. For example, the arrows from the c1 sub-cluster into a5 and from b6 into a3 reflect that the symbol 'a' can be preceded by either a 'b' or a 'c'. a5 contains 25 neurons which is approximately half of the neurons in the 'a' weight cluster. This correctly reflects that 'a' is preceded by a 'c' in 50% of the cases. In the following it will be shown, how this information can be employed to extract an FOMM.

[8] β was set to 1-α in all training sessions.

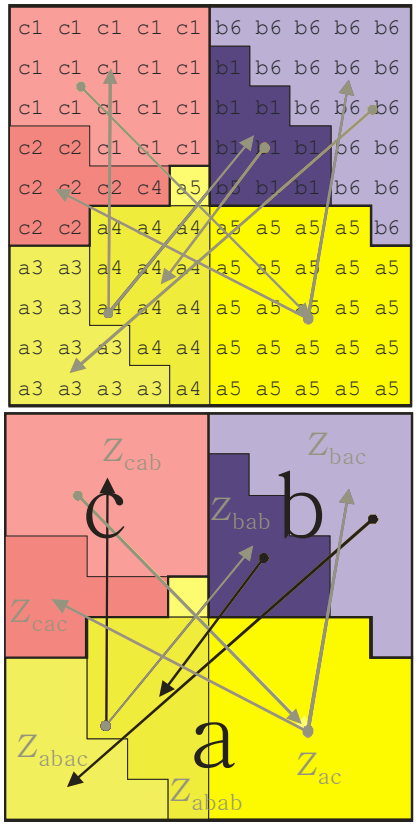


Figure 17 Clusters and sub-clusters on a SOMSD trained to the Markov model from figure X. (X=0.2) Arrows denote average context vectors with the arrow tail being at the average grid point coordinates of the context vectors.

By the definition of an FOMM of order o, the FOMM is in the same state z whenever a particular preceding sequence of length o, $s_1...s_o$ has been emitted. A sequence $s_1...s_o$ can be found that describes predecessors[9] of each context cluster. The symbols within the sequence are determined by the names of weight clusters. The sequence is given by the arrows among sub-clusters as depicted in the figures. This sequence can be found for any order o, as each sub-cluster has an incoming arrow. It will in most cases be unique for each sub-cluster given a large enough o.[10] The sequence can be used for naming sub-clusters. Then, the length of the sequence each sub-cluster can be chosen to be minimal while staying different for each sub-cluster and unambiguous in the sense that they could not be assigned to another sub-cluster. This sequence will be called defining sequence of a sub-cluster. In the lower part of Figure 17 the defining sequence was used to identify each sub-cluster with a state in an FOMM that could be denoted by this defining sequence. z_{cab} for example is in the weight cluster 'c', its average context vector lies in an 'a' weight-cluster (arrow runs from 'a' to 'c'). More precisely, it lies in a sub-cluster that has an average context vector in a 'b' weight cluster. Now that FOMM states have been identified, transition probabilities like $p_t(z_{ba} \mid z_{ab})$ are to be determined. Assuming – like previously done for weight clusters – that the size of context clusters reflects the frequency of occurrence of the denoted transition probabilities can be ex-

model	$\lvert z_{bab} \rvert$	$\lvert z_{ab} \rvert$	input	extracted
X = 0.0	0	25	0	0
X = 0.1	7	20	0.1	0.35
X = 0.2	10	23	0.2	0.43
X = 0.3	6	21	0.3	0.29
X = 0.4	13	25	0.4	0.52
X = 0.5	12	26	0.5	0.46
X = 0.6	15	27	0.6	0.56
X = 0.7	15	25	0.7	0.60
X = 0.8	19	21	0.8	0.90

Table 4 Extraction of selected transition probabilities for SOMSDs from different models using sub-cluster sizes.

[9] The symbol emitted in the weight cluster the sub-cluster belongs to is considered s_1. It is thus counted as predecessor for easier formulation.

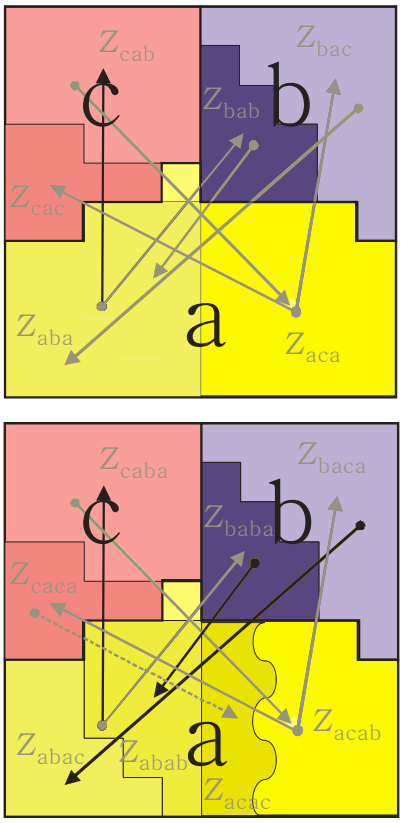


Figure 18 (top)The SOMSD after clustering and renaming to obtain defining sequnces of length 3. (bottom) The SOMSD after clustering and renaming to obtain defining sequnces of length 4.

tracted like in the following example:

$$p_t(z_{ba} \mid z_{ab}) = \frac{|z_{bab}|}{|z_{ab}|}$$

Where $|z_{bab}|$ denotes the size of the sub-cluster named z_{bab}. This is an example from order 2 FOMM extraction. Clearly a set of sub-clusters with a defining sequence of length 3 and one of length 2 is needed to calculate all probabilities. The steps of merging and renaming of sub-clusters needed to obtain these will be discussed after the presentation of some extraction results.

The figures in Table 4 have been calculated from sub-clusters on SOMSDs where $p_t(z_{ba} \mid z_{ab})$ varied with the parameter X of the input sequence. The extracted transition probabilities do not match the input probabilities as precisely as those extracted with the previously introduced algorithm. However, the broad tendency is properly reflected. In most cases, extracted probabilities are closer to 0.5 than the respective input probabilities. Sub-clusters representing less frequent transitions are over proportionally large. This suggests that some kind of magnification factor of the SOMSD causes the deviation.

In general, first the order o has to be set. Then the extraction of an order o FOMM is done by merging and renaming sub-clusters in a way that the SOMSD is covered by sub-clusters with a defining sequence of length o+1. And afterwards again with a defining sequence of length o. Transition probabilities are then calculated for all sequences $s_1 \ldots s_{o+1}$ allowed from the alphabet, whenever $|z_{s_2 \ldots s_{o+1}}| \neq 0$:

$$p_t(z_{s_1 \ldots s_o} \mid z_{s_2 \ldots s_{o+1}}) = \frac{|z_{s_1 \ldots s_{o+1}}|}{|z_{s_2 \ldots s_{o+1}}|}$$

This approximation is based on the assumption that sub-cluster sizes reflect the frequency of the occurrence of their defining sequence. $|z_{s_1 \ldots s_{o+1}}|$ therefore can be put in relation to $|z_{s_2 \ldots s_{o+1}}|$ to find the fraction of cases in which $s_2 \ldots s_{o+1}$ is followed by s_1.

Figure 18 shows the SOMSD from Figure 17 with sub-clusters that were merged and renamed so that they all have a defining sequence of length o = 3.

Merging of sub-clusters takes place whenever two or more sub-clusters have defining sequences that do not differ in the first o (or o+1 respectively) symbols. This is by definition the case for all defining sequences longer than o (or o+1). These

[10] Two sub-cluster might be identified by the same sequence if the SOMSD does not express the dynamics sufficiently clear. They may in this case be assumed to jointly describe an FOMM state.

clusters are merged to jointly denote a state. This state inherits all the possible predecessors of the merged sub-clusters. In Figure 18(top), z_{abac} and z_{abab} were merged to denote z_{aba} with two predecessors: z_{bab} and z_{bac}. Note that merging is the only way to obtain FOMM states with several predecessors. As every finite non-deterministic FOMM has states with in-degree > 1, merging is a necessary step in extraction.

Renaming sub-clusters becomes necessary whenever defining sequences are too short to take part in the above formula. In that case, predecessors as defined by average context vectors have to be taken into account. The defining sequence can be completed using the defining sequence of the predecessor. In this way, z_{ac} has been renamed to s_{aca} due to the predecessor z_{cab}. Assuming, defining sequences of length 4 are needed (Figure 18, bottom), a problem becomes obvious when renaming $z_{ac.}$ Renaming it z_{acab} would lead to:

$$p_t(z_{aca} \mid z_{cab}) = \frac{|z_{acab}|}{|z_{cab}|} = \frac{25}{18} > 1$$

At the same time, $p_t(z_{aca} \mid z_{cac})$, which is clearly a possible transition in the input model will be zero. For this reason, neighbours of direct predecessors should be taken into account. Recalling the above assumption that size reflects frequency, there must be another predecessor of what is now called z_{acab} to account for the size of 25. In this case, obviously z_{cac} should be taken into account as the average context vector of the sub-cluster named z_{ac} lies close to it. z_{ac} should therefore be renamed to both, z_{acac} and z_{acab}. In order to assign appropriate probabilities their sizes have to be adjusted to the size ratio of the predecessors z_{cac} and z_{cab}. The appropriateness of this measure can be observed by symmetry to z_{abac} and z_{abab} in Figure 18(bottom).
As a general rule, renaming should take into account neighbours to the predecessors as soon as there are signs that the predecessor selection is ambiguous. These signs are: Probabilities calculated to be larger than 1 and average context vectors close to sub-cluster borders.
With the help of merging and renaming of sub-clusters, FOMMs of any order can be calculated from weight and context clusters of a SOMSD. Extraction of order 2 (Table 4) and order 3 (Figure 18) FOMMs has been considered here. The question remains, whether it is possible to determine the order of the underlying FOMM. Possibly the number of merging and renaming operations necessary to extract an FOMM of a certain order can be an indicator for the right order.
This experiment shows that the topological ordering of context vectors can be used for extraction. This leads to a more efficient extraction mechanism because the possible predecessors of every single neuron do not have to be evaluated. Also, the order of the order of the FOMM underlying input generation might be found. Finally, through the additional notion of context clusters it might be possible to generalize the approach to more complex dynamics such as non-FOMM HMMs, in which weight clusters cannot be used for state determination as several symbols might be emitted at each state.

11. Generalization to HMMs

Up to now, the extraction of FOMMs has been considered. In order to generalize the technique to non-FOMM HMMs. Apart from FOMMs hidden Markov models with infinite order exist. That is the current state cannot be determined from any finite number of observed output symbols, which is due to hidden states, in which different symbols can be emitted. (Figure 19 gives a simple example) Therefore general HMMs can only be represented in a SOMSD if hidden states can be treated. It was tried to train several HMMs with hidden states to square SOMSDs of size 100 and 196 in the same fashion as above. However, context clustering usually produced too many states to allow for a straightforward interpretation. At the same time, context errors remained considerably higher than in previously presented experiments. It is therefore reasonable to ask, if there are in principal problems with the representation of hidden states in a SOMSD.

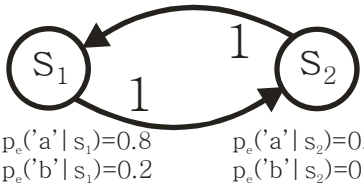


Figure 19 Example of a simple non-FOMM HMM with all states beeing hidden states and emission probabilities given at each state.

FOMMs are defined as Markov models in which the state can be unambiguously determined from the observed symbols. This partnership of state and symbols has been used for extraction. The context vectors were interpreted as a recursive reference to a set of weight clusters which were in turn interpreted as symbols. Hidden states are by contrast defined as states that abstract from the emitted symbols. The ability of the SOMSD to represent hidden states depends on the ability to represent states abstracted from their symbols. Clearly, this is difficult on a single neuron basis as every neuron has only one weight vector and one context vector. The weight vector can hardly represent more than one symbol and the context vector can only refer to a very limited set of possible predecessors. Longer contexts are represented recursively through further references. This imposes a hierarchy on the influence of patterns within a context on winner selection. With high α values, the alternation of a symbol in the recent context is considerably more prominent than an alternation further back in the context. Low α values will not be considered here because with low α no reasonable clustering could be obtained due to the lack of convergence in the weight vector topology.

If hidden states can not be represented on a single neuron basis, groups of neurons have to be taken into account. For identifying a group of neurons as such, clustering seems to be a reasonable choice. As several symbols are involved at each state, identifying these groups by weight clusters is not advisable. It therefore seems reasonable to consider context clustering to identify states. A context cluster contains a set of neurons with similar predecessors, which is a prerequisite of representing a state. However, its property of being a hidden state imposes requirements on the topology: Clearly, it has to be possible to identify a weight cluster within a context cluster representing a hidden state for each symbol that can be emitted. Moreover, the arrangement of weight clusters within the context cluster has to reflect the emission probability of each symbol (i.e. for example equal probability). Also, the context cluster has to be sufficiently compact to allow context vectors of neurons representing possible successors to refer to the cluster as one state. It might well be, that these requirements cannot be met even for relatively small HMMs. Therefore no perspective of the above extraction mechanisms to non-FOMM HMMs can be seen. It remains an open question, if the SOMSD is suitable for representing hidden states as

due to the recursive representation of predecessors and the fixed relationship of weights and contexts training might not produce the desired topology.

12. Summary

It is the aim of this work to explore the application of data mining methods to the SOMSD. The question is, which temporal structures in a sequence can be learned by a SOMSD and whether it is possible to determine structure underlying the generation of that sequence. In the experiments, sequences of real values generated by Markov models were used as input to the SOMSD to simulate measured values that stemmed from processes which can be in several distinct states. The Markov models of the presented sequences were thereby known. Therefore comparison of the extracted model and the actual model was possible. A high correlation could be taken as an indicator of the SOMSD's ability to store relevant information. Moreover, the power of the data mining tool, the U-Matrix and its applicability to the SOMSD has been be demonstrated.

In particular because continuous data was presented, quantisation was a central step in model extraction as it was done here. Two levels of quantisation take place: Winner selection was a quantisation process which is done in with regard to both, the current pattern and the context in a weighted fashion. After training, neuron clustering was done separately for weight and context vectors. This way a limited number of states and symbols from a possibly unlimited number of data sets was determined, which was necessary to find Markov models. The following has been verified in order to show that extraction is possible:

(1) The distribution of weight vectors reflects the distribution of the patterns. In particular, many weight vectors represent areas of the input space where many input patterns lie in.
(2) The topological arrangement of weight vectors on the grid preserves similarity of input patterns so that neighbourhood of two neurons representing some part of the input space actually has a meaning with regard to these parts of the input space.
(3) Drawing cluster borders is possible. (1) and (2) are prerequisites for that. The U-Matrix basically exploits particularities of the distribution and the arrangement of weight or context vector to allow a clustering.
(4) Context vectors reflect the dynamics of the underlying Markov model: The weight vectors of neurons that are preferably activated after each other according to context vectors form sequences that are likely to occur in the input sequence.
(5) The number of contexts representing a certain transition among neurons of a particular weight cluster are in some monotonous relation to the respective transition probability in the Markov model.
(6) The context vectors also allow clustering in the sense that neighbourhood of context vectors has a meaning and cluster borders can be drawn.

(1)-(3) were verified by showing that the U-Matrix expresses well on a trained SOMSD (see Figure 4, section 9), (4)-(6) demanded from the context vectors behaviour being analogous to the behaviour of weight vectors. The FOMM extraction algorithm from section 8 uses (1) to (5). Entry (6), the ability of the SOMSD to cluster contexts in a way that allows Markov model extraction, was explored in section 10.

The focus of this work has been on finite order Markov models (FOMMs). This is a sub-class of Markov models in which states can be identified by looking at the symbols that have been emitted. In particular, a standard form FOMM of order o can be found in which each state is unique with respect to a context of o previously emitted symbols. It is this standard form that was extracted during the experiments. The standard form of an FOMM stores information on a sequence similar to the way it is encoded in recursive SOMs. A neuron of a recursive SOM selects a current pattern by its weight vector and recursively several predecessors by the context vector. One or several similar neurons therefore can constitute a state of an FOMM standard form as these states are also denoted by a current symbol and its predecessors.

Two approaches to FOMM extraction have been taken. In section 9, the FOMM extraction algorithm was applied which counts the transitions denoted by context vectors. The extracted Markov models reflected the dynamics of the input sequence relatively precisely up to a rather high noise level. In section 10, extraction using context clusters was done. Intersections of weight and context clusters were used to identify FOMM states and their sizes gave approximations for transition probabilities. Extraction done this way turned out to be less exact in the examples given. However, the approach is appealing due to a higher degree of abstraction. The results show that a trained SOMSD reflects important information contained in sequenced data. Therefore an abstract description of the sequence is possible.

Extraction has shown to be possible under noisy conditions. This might be a strength of the SOMSD context representation. Winner selection always depends on both, weight and context vectors – that is the current pattern and the context. In this respect, SOMSD FOMM extraction is fundamentally different to canonical approaches that are based on the data rather than on some intermediate representation. Such an algorithm involves several distinct steps:

(1) Clustering of the input patterns.
(2) Choice of an order o
(3) Counting of subsequences of length o+ 1 and o
(4) calculation of transition probabilities

In particular, the clustering of input patterns has to be done before sequence dynamics are evaluated. This is because the evaluation of the sequence dynamics –namely counting – is only possible when a finite number of classes is given. Such a clustering algorithm has only the information contained in patterns themselves available for clustering, which can lead to difficulties with high noise. U-Matrix clustering uses the arrangement of the SOM which reflects the patterns as well as the dynamics of the sequence. This shows an analogy to human information processing, where top-down perception can be observed: People tend to understand words of spoken language under in an noisy environment easier in a context[3].

Abstract description of input can also be found in the human brain. Human language understanding is basically abstract analysis of a – certainly noisy – sequence of acoustic input. Of course, this is not a close analogy, as human language cannot be described by Markov models ([4]) and language understanding is a complex multi-layer process. However, one could imagine that structures that can be modelled with a SOMSD are involved in a part of the process. (e.g. the identification of syllables)

13. Further Research

The experiments so far have only been a mere demonstration of the feasibility of model extraction from the SOMSD. In order to claim that the technique can be applied in practice, it has to be shown that the useful features of the SOMSD are present in a broader range of circumstances. Some limitations of the technique or of details of the implementation have occurred and demand further investigation. They arise partly from theoretical reflection and partly from observations during the experiments:

One important point that has to be kept in mind is that the representation of context information is done recursively in the SOMSD. The way context is represented imposes a hierarchical order on the influence of preceding patterns. Alternating a more recent pattern usually has stronger influence on winner selection than alternating later ones. It is therefore unclear, up to which order, FOMMs can be extracted. For the same reason, hidden states can be poorly represented as the sensitivity to alternation of recent symbols hardly allows to abstract from the symbol that is emitted at one state.
Also it is clear, that equivalent Markov model cannot be distinguished by the SOMSD as they generate the same sequence with the same probability. FOMMs so far have been extracted in standard form, which is a form that is unique for each FOMM and can be predicted with the FOMM at hand.
During the experiments with FOMM extraction using context clusters, the number of sub-clusters was in all demonstrated cases sufficient to allow extraction of order 2 FOMMs. Occasionally, there were more sub-clusters than necessary. Theoretically there is no limit to the number of sub-clusters that can be distinguished, provided enough neurons are available. It is therefore left to the way the U-Matrix is evaluated to determine the right granularity of sub-clustering.[11]

Several points have to be assessed empirically to show that the SOMSD can be applied in real world data mining in the way it has been demonstrated here. The most prominent requirement is scalability. It is important to find out whether the technique can be applied to more complex problems. The complexity of the problems can be captured as properties of the Markov models that are to be extracted. Markov models of higher order will be more difficult to extract due to the recursive nature of the representation. The size of the alphabet of the MM will have an influence on the quality of the SOMSD representation. It is well possible that a larger SOM can fully compensate for the higher number of weight clusters present when more symbols are used. Also, the connectivity of the model is an important parameter, which is reflected in the number of non-zero transition probabilities or the number of states in an FOMM standard form. One special case of Markov model connectivity is self-reference, that is models in which states can be reached from themselves. It remains to be established, if model extraction is equally possible with such models. Apart from properties of the Markov models, the complexity of data mining tasks depends on properties of the patterns: A very important feature is the dimensionality of the input data. In the experiments done for this work, 2-dimensional input vectors have been

[11] For the U-Matrix to determine cluster borders, the clusters have to have a certain minimal size. Clearly, the way it is applied here, the U-Matrix will not declare each neuron a different cluster.

used. It is a strength of the SOM to cope with much larger input vectors and to find a salient topological arrangement in 2-dimensional space. However, there are limits to the possibility of topological arrangement in particular because the number of neighbours is limited. Neighbourhood on the map is an important concept in SOM representation, but 2-dimensional space allows a smaller number of neighbours than higher dimensional space. Also, the noise that is added can differ in its standard deviation and might also be biased. In connection with noise, one important limitation of the above experiments has to be mentioned: In FOMM extraction in both, section 9 and section 10, the clusters have to be identified and unified according to the symbol they denote. The proximity to one of the vectors used for sequence generation was used as a criterion. For further generalisation of the approach to previously unknown symbols, it has to be shown that the algorithms can cope without this a-priori information. (See section 9 for a short analysis on, why in the above cases the influence of the a-priori information was not that high.)

It might be advisable to modify the Markov model used in sections 9 and 10 in the following ways to adress these problems:

To test if higher order FOMMs can be stored in and extracted from a SOMSD, more complex input Markov models have to be used. Clearly, a larger number of weight or context clusters will be necessary to represent the additional standard form states.

Trying the same sequence on a SOMSD of a different size is a simple modification that should be done to determine, whether the quality of the model representation could be improved.

A larger alphabet cannot be applied without changing other properties of the Markov model, in particular, the connectivity always gets more complex, or the order is changed. Therefore no extra experiments on the size of the alphabet are suggested here. However, the size of the alphabet has to be kept in mind particularly when choosing the size of the SOMSD. In the models used above, the in-degree and out-degree of each state never exceeded 2. Representation might become difficult as soon as states tend to have more predecessors and successors than they have neighbours, or if relative transition probabilities vary very strongly within one state. Experiments with self-reference can be done with a similar setup as the one used above, with the only modification that symbols can be repeated with a certain probability.

Some experiments on noise have been done in section 9. Note that higher noise might require changes in the α/β ratio. Because higher differences between weight vectors and patterns might play a role in winner selection. Also – as seen above – clustering might get considerably harder. When training data with input of higher dimensionality, it is particularly interesting to try an input space topology that cannot be mapped on 2D space. This would cause the topology of the SOMSD to be dependant either on random processes or context dynamics.

Three points are left to be mentioned that have not been addressed and might reveal further power of the approach.

- Cluster determination on the U-Matrix is presently done by a uniform water level over the whole map. This is a heuristic approach that assures in no way that every relevant minimum is taken into account when determining clusters. A more detailed analysis of the U-Matrix landscape might be necessary to obtain all clusters needed to extract an FOMM of a certain order.
- In FOMM extraction, the number of neurons within a cluster or the number of neurons close to context vectors of a neuron is used to approximate transition probabilities. Every neuron here takes part in the count to the same degree. Especially if there is a certain amount of noise in the data, a considerable amount of neurons is located on mountain ranges of the U-Matrix. High U-values indicate that the respective neurons represent ambiguous input patterns, sometimes only a small number of them. In these cases it might make sense to give these neurons less importance in the count of clusters sizes and transitions to get more faithful transition probabilities.
- As shown in Figure 16, a SOMSD with clusters and sub-clusters might be a good visualisation of the Markov model underlying the trained sequence. Maybe this clear structure might also help to determine the order of an FOMM. The shape and size of sub-clusters might allow such interpretations.

The final and important test will be the application to real world data. For example, it might be possible to interpret results from eye tracking experiments using the SOMSD. In layout efficiency analysis, it is interesting to identify objects that attract attention and evaluate the timing and the order in which they are looked at [23]. Eye trackers do measurements from which time and coordinates of fixations can be taken. The fixation behaviour might be described with Markov models. FOMM extraction as described above may allow to identify objects and derive the pattern that underlies the order in which they are fixated. A sequence of coordinates of fixation points (maybe augmented by information on the duration) can be trained to a SOMSD. U-Matrix clustering on the weight vectors might then reveal a set of areas that are primarily fixated. Using these as FOMM states might and applying the extraction algorithm might lead to interpretation like: People who look at the text and then at the navigation bar, are not likely to look at the logo next. Hence the FOMM extraction using the SOMSD would have allowed the derivation of explicit abstract rules from a sequence of data.

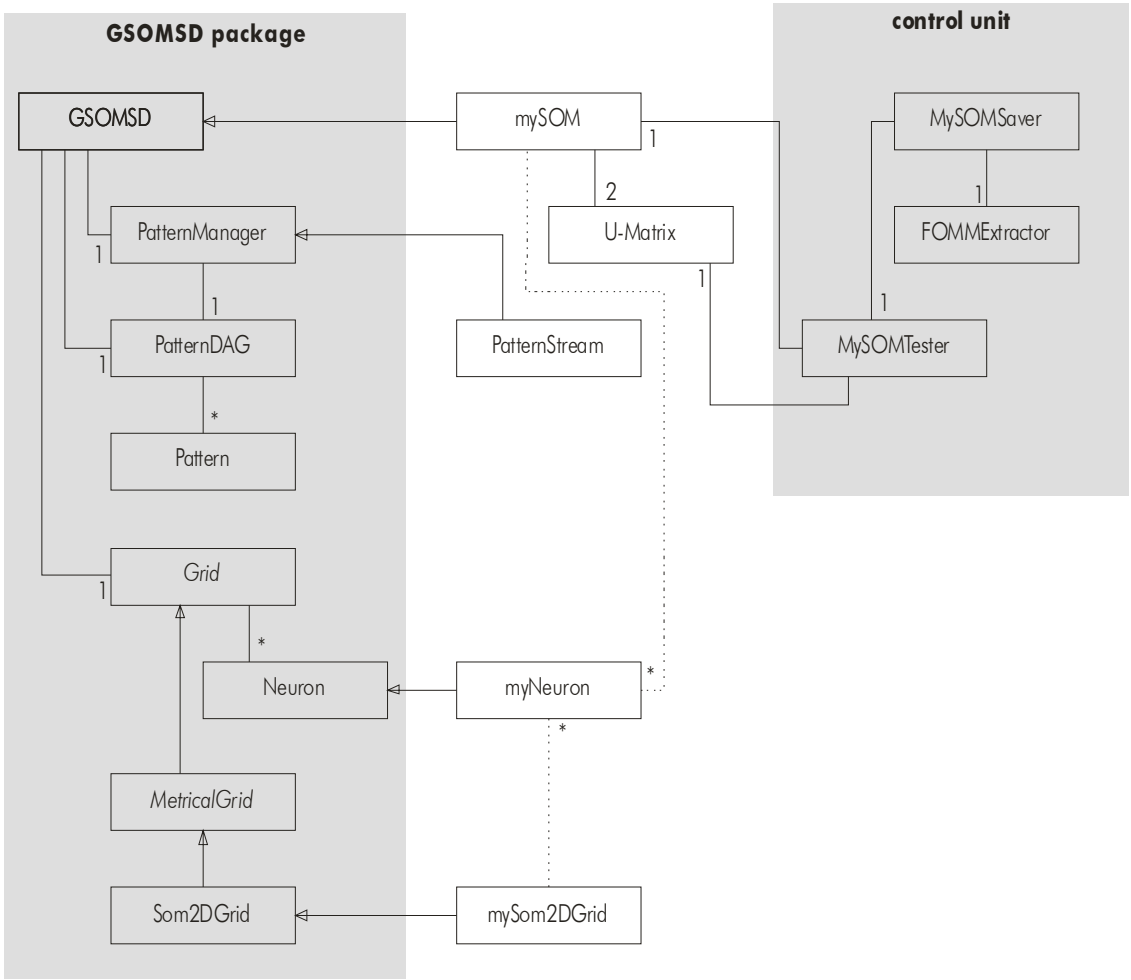
14. Appendix

The CD

The CD contains the Java source code and class files.

Data is also available from http://www.blomega.de/papers.htm .

Object Model:



The figure shows the object model (using UML conventions) of the Java™ implementation used for the experiments. The classes outside the GSOMSD package were developed for this thesis. Dotted lines reflect relations forced by casting. Some relations are not shown. For example, the class 'myNeurons' holds ArrayLists holding 'Pattern' objects that are won by the neurons to allow further analysis.

Training Protocols of the Presented Experiments in section 9 (for table 2 only)

Training parameters:

```
SOMWIDTH = 10
LOWERLIMIT = 0              UPPERLIMIT = 5000
REPETITIONFACTOR = 5              WINDOWSIZE = 15
TRAIN = 500          REPEAT = 30
minAlpha = 0.93             maxAlpha = 0.97
alphaEta = 1.0E-6
minSigma = 0.5              maxSigma = 5.0
sigmaDecay = 0.99995
minLearningR = 0.0050             maxLearningR = 0.02
learningREta = 1.0E-5
```

The protocol shows for each training session key measures before (c=0) and after (c=15000) training:

- "Depth" (not evaluated)
- "#Clusters", the number of weight clusters
- "Pats", the number of patterns presented per iteration
- "Wins", the number of active neurons
- "CE", the context error
- "CR", the context radius
- "alpha"
- "sigma"
- "LR", the learning rate

as well as the U-Matrices for weight and context vectors (' 'indicates low U-values, "." medium, "o" high), the weight U-Matrix is also displayed as a matrix of cluster numbers. Also, the number of clusters and the water levels are given.

```
file = 00mat-01noise.gsomsd
c      Depth  #Clusters    Pats   Wins    CE    CR     alpha  sigma  LR
0      5.4545       4      1000   22.0   2.2063 0.0    0.97   5.0    0.02
15000  1.3146       4      1000   89.0   0.8022 5.5493 0.93   0.5    0.0050
Weights      Contexts
   .O.          ..o. ..
   .Oo.        ...o. ...
   .oo..       ...oo....
.   oO....    ....ooo..
oo.. OOooo    .oooooooo..
.OOOo oooo    .ooOOooo..
 ..oO.         .oOOo....
   .Oo.        .........
   .oO.       .....oo...
    .o.        .  .oo...
 1 1 1 1 3 5 5 5 5 5
 1 1 1 3 3 5 5 5 5 5
 1 1 1 3 3 5 5 5 5 5
 4 1 4 4 3 5 5 5 5 5
 4 4 4 4 3 3 5 5 5 5
 2 2 4 4 4 4 4 4 4 6
 2 2 2 2 4 4 4 4 6 6
 2 2 2 2 2 4 6 6 6 6
 2 2 2 2 2 4 4 6 6 6
 2 2 2 2 2 2 4 6 6 6
6Weight Clusters: (Water-Level:0.3322179214837734)100
6Context Clusters: (Water-Level:4.826511816342036)
```

```
file = 01mat-01noise.gsomsd
c      Depth  #Clusters    Pats   Wins   CE     CR     alpha  sigma  LR
0      4.0          1      1000   28.0   3.0986 0.0    0.97   5.0    0.02
15000  1.0          4      1000   90.0   0.8803 6.5659 0.93   0.5    0.0050
Weights      Contexts
   oO.         . ....
   oO.        ..........
   .Oo        .o.  .....
   .OO.       .o. ooo.o.
. .OOooo..    .ooOo.oooo
ooOO. .ooo    .oOOooooo.
...     ..    .oo..o....
                ...... .
              . .......
                .......
 1 1 1 1 7 7 7 7 7 7
 1 1 1 1 7 7 7 7 7 7
 1 1 1 1 1 7 7 7 7 7
 1 1 1 1 1 7 7 7 7 7
 1 1 1 1 5 5 5 7 7 7
 1 1 1 5 5 5 5 5 5 5
 2 5 5 5 5 5 5 5 5 5
 2 2 5 2 2 2 5 5 2 2
 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2
4Weight Clusters: (Water-Level:0.3139357197007536)100
3Context Clusters: (Water-Level:5.271068293189625)

file = 02mat-01noise.gsomsd
c      Depth  #Clusters    Pats   Wins   CE     CR     alpha  sigma  LR
0      7.4167       1      1000   24.0   2.0634 0.0    0.97   5.0    0.02
15000  1.1505       4      1000   93.0   0.868  6.8371 0.93   0.5    0.0050
Weights      Contexts
                .....
       .      . .......
              .....o...
...    ..     .o...oo..
ooo.  oooo    oooO.ooOo.
..oOoOOooo    oooOoOOOoo
   oOO.       .o.OOooOOo
   .Oo        ..ooOoooo.
   .Oo        .o.oOo....
   .O.        .. .o.....
 1 1 1 1 1 1 1 1 1 1
 7 7 7 1 1 1 1 1 1 1
 7 7 7 7 7 7 7 7 7 7
 7 7 7 7 7 7 7 7 7 7
 3 3 7 7 7 7 7 7 7 7
 3 3 3 3 7 7 4 4 4 4
 3 3 3 3 7 4 4 4 4 4
 3 3 3 3 3 4 4 4 4 4
 3 3 3 3 3 4 4 4 4 4
 3 3 3 3 3 4 4 4 4 4
4Weight Clusters: (Water-Level:0.30774556659301167)100
5Context Clusters: (Water-Level:5.355940003265186)
```

```
file = 03mat-01noise.gsomsd
c      Depth  #Clusters    Pats   Wins   CE     CR     alpha  sigma  LR
0      6.0          1      1000   28.0   1.7895 0.0    0.97   5.0    0.02
15000  1.7368       4      1000   95.0   0.8811 6.4658 0.93   0.5    0.0050
Weights      Contexts
              ........
             ..o.......
             ..o.  ...
...          .oo. .....
.ooo  ....   .oOO..ooo.
 ..Oooo.o.   ..oOOooooo
   .OO. .    ....oo....
   .O.       oo.. .....
   .O.       ..oo...o..
    Oo         .......
 5 5 5 5 5 5 5 5 5 5
 5 5 5 5 5 5 5 5 5 5
 5 5 5 5 5 5 5 5 5 5
 5 5 5 5 5 5 5 5 5 5
 3 3 5 5 5 5 5 5 5 5
 2 3 3 3 5 5 6 6 6 6
 2 2 3 3 3 6 6 6 6 6
 2 2 3 3 3 6 6 6 6 6
 2 3 3 3 6 6 6 6 6 6
 3 3 3 3 3 6 6 6 6 6
4Weight Clusters: (Water-Level:0.28687614774993425)100
7Context Clusters: (Water-Level:6.126319928034869)

file = 04mat-01noise.gsomsd
c      Depth  #Clusters    Pats   Wins   CE     CR     alpha  sigma  LR
0      8.4118       1      1000   17.0   2.8605 0.0    0.97   5.0    0.02
15000  1.6087       3      1000   92.0   0.8958 5.7866 0.93   0.5    0.0050
Weights      Contexts
   ...         ...o..
   ...        .oooo....
   ...        .o.oo.....
  ..o.        ...ooo....
.ooOo.        o..o.o....
oooOO         oo.oo.....
.  .o.        .o.oOo....
    ..        .oooOO....
    .o.       ..ooOO..
     ..       ....oOo..
 1 1 1 1 1 6 6 6 6 6
 1 1 1 1 1 6 6 6 6 6
 1 1 1 1 1 6 6 6 6 6
 1 1 1 1 1 6 6 6 6 6
 1 1 1 1 1 6 6 6 6 6
 2 2 2 2 6 6 6 6 6 6
 2 2 2 2 2 6 6 6 6 6
 2 2 2 2 2 6 6 6 6 6
 2 2 2 2 2 6 6 6 6 6
 2 2 2 2 2 2 6 6 6 6
3Weight Clusters: (Water-Level:0.3074500642162425)100
6Context Clusters: (Water-Level:5.607533332849556)
```

```
file = 05mat-01noise.gsomsd
c      Depth  #Clusters    Pats   Wins   CE     CR     alpha  sigma  LR
0      4.6296       1      1000   27.0   3.6006 0.0    0.97   5.0    0.02
15000  1.3043       4      1000   92.0   0.9559 5.9557 0.93   0.5    0.0050
Weights      Contexts
              ..... ..
              ...... ..
              ...o.....
       ...    ...oo...oo
...    ooo.   ooooo.OOOO
.ooo.OO..     .OOoooOoOo
  .oOO.       oOoooo.ooo
    OO        oOoOOooooo
    .O.       .ooOoooo.o
    .o.       .ooo...ooo
 1 1 1 1 1 2 2 1 1 1
 1 1 1 1 1 2 1 1 1 1
 1 1 1 1 1 2 1 1 1 1
 2 2 2 2 2 2 1 1 1 1
 2 2 2 2 2 2 1 1 1 5
 3 3 3 2 2 1 5 5 5 5
 3 3 3 3 3 5 5 5 5 5
 3 3 3 3 3 5 5 5 5 5
 3 3 3 3 3 2 5 5 5 5
 3 3 3 3 3 3 5 5 5 5
4Weight Clusters: (Water-Level:0.3143893600441117)100
7Context Clusters: (Water-Level:5.731180319067941)

file = 06mat-01noise.gsomsd
c      Depth  #Clusters    Pats   Wins   CE     CR     alpha  sigma  LR
0      3.5455       1      1000   22.0   1.7352 0.0    0.97   5.0    0.02
15000  1.6022       4      1000   93.0   0.8795 7.1415 0.93   0.5    0.0050
Weights      Contexts
   .O.        .ooo..oo
   oO.        oooOoooOO.
   oO.        OOooooooooo
  .oOo..      .ooOoo..oo
..ooooooo..   .oOOoooOOo
ooo.  .ooo    OOoooo.oOo
..      ..    ooooooooo.
              ..ooo.....
              ..........
               .......
 1 1 1 1 1 6 6 6 6 6
 1 1 1 1 6 6 6 6 6 6
 1 1 1 1 6 6 6 6 6 6
 1 1 1 1 4 6 6 6 6 6
 1 1 1 4 4 4 4 6 6 6
 1 4 4 4 4 4 4 4 4 6
 4 4 4 4 4 4 4 4 4 4
 4 4 4 4 4 4 4 4 4 4
 2 2 4 4 4 4 4 4 4 4
 2 2 2 4 4 4 4 4 4 4
4Weight Clusters: (Water-Level:0.31782703139637936)100
5Context Clusters: (Water-Level:5.470390580283578)
```

```
file = 07mat-01noise.gsomsd
c      Depth  #Clusters    Pats   Wins   CE     CR     alpha  sigma  LR
0      4.6522       1      1000   23.0   3.0434 0.0    0.97   5.0    0.02
15000  1.4565       5      1000   92.0   0.8435 5.5812 0.93   0.5    0.0050
Weights      Contexts
    ...      .........
    .o.      .....o....
    .o       .oo.oo..
 ...o.       .oooo....
oOOOo        ...OOo....
Oo.oo        ...OOo....
.  .o.       ...oOo....
    .o       o...ooo...
    .o.      ooo..oo...
    ..       .oo.....
 1 1 1 1 1 4 6 6 6 6
 1 1 1 1 1 4 4 5 6 6
 1 1 1 1 1 4 4 5 5 6
 1 1 1 1 1 4 4 5 5 5
 1 1 2 1 4 4 4 5 5 5
 2 2 2 2 4 4 4 5 5 5
 2 2 2 2 2 4 4 5 5 5
 2 2 2 2 2 4 4 5 5 5
 2 2 2 2 2 4 4 5 5 5
 2 2 2 2 2 4 5 5 5 5
5Weight Clusters: (Water-Level:0.3150969741158941)100
6Context Clusters: (Water-Level:5.258908188860101)

file = 08mat-01noise.gsomsd
c      Depth  #Clusters    Pats   Wins   CE     CR     alpha  sigma  LR
0      2.8261       1      1000   23.0   2.0226 0.0    0.97   5.0    0.02
15000  1.1573       3      1000   89.0   0.8865 7.2392 0.93   0.5    0.0050
Weights      Contexts
     .o      .o...oo.
     oo.     oOOo.oo...
    .Oo      oOo..Oo..
...oOo       .o..oo....
oOOOO        ...oo.....
.oooo.       ooo.oo..o.
   .o.       .oooOoooo.
   .o.       ..oOo.....
   .o.       oooo.o....
   .o.       .oo... .
 1 1 1 1 1 1 3 3 3 3
 1 1 1 1 1 1 3 3 3 3
 1 1 1 1 1 1 3 3 3 3
 1 1 1 1 1 3 3 3 3 3
 1 1 1 3 3 3 3 3 3 3
 2 2 2 2 2 3 3 3 3 3
 2 2 2 2 2 3 3 3 3 3
 2 2 2 2 3 3 3 3 3 3
 2 2 2 2 3 3 3 3 3 3
 2 2 2 2 2 3 3 3 3 3
3Weight Clusters: (Water-Level:0.3018334292410147)100
8Context Clusters: (Water-Level:5.490486708797992)
```

Clustering Information used for section 10

For each map, a combined representation of weight and context clusters is shown. The letter gives the (unified) weight cluster, the number denotes the context cluster. Middle values for the clusters are listed.

```
00mat-01noise2.som Order:2
 c1 c1 c1 c1 c1 c4 a4 a4 a4 a4
 c1 c1 c1 c1 c1 a4 a4 a4 a4 a4
 c1 c1 c1 c1 c1 a4 a4 a4 a4 a4
 c1 c1 c1 c1 c1 a4 a4 a4 a4 a4
 c1 c1 a1 a1 a1 a4 a4 a4 a4 a4
 a1 a1 a1 a1 a1 a1 a3 b3 b3 a3
 a1 a1 a1 a1 a1 b3 b3 b3 b3 b5
 a1 a1 a1 a1 b2 b3 b3 b3 b3 b5
 a1 a1 a1 a1 b2 b2 b3 b3 b3 b2
 a1 a1 a1 b1 b2 b2 b3 b2 b2 b2
Weights:
1'c': (-0.008356266723424996, 0.9359888804259986)
2'a': (-0.001292869870699878, 0.06296552699054445)
3'b': (0.9142898214083094, -0.003021512393604514)
4'a': (0.038020495124488206, 0.004528753658591943)

Contexts:
1'.': (4.287951135145407, 6.846795990335733)
2'.': (6.258723312912225, 2.9999924929517165)
3'.': (7.087757988102371, 1.5417985848084876)
4'.': (1.7615787285035873, 1.7611851457055698)
5'.': (8.284799119624292, 2.935065471697438)


01mat-01noise2.som Order:2
 b1 b1 b1 b1 c3 c3 c3 c3 c3 c3
 b1 b1 b1 b1 b3 c3 c3 c3 c3 c3
 b1 b1 b1 b3 b3 c3 c3 c3 c3 c3
 b1 b1 b3 b3 b3 c3 c3 c3 c3 c5
 b1 b1 b1 b3 a2 c2 c2 c2 c5 c5
 b2 b2 b2 a2 a4 a5 a5 a5 a5 a5
 a2 a2 a2 a2 a4 a4 a5 a5 a5 a5
 a2 a2 a2 a2 a4 a4 a5 a5 a5 a5
 a2 a2 a2 a2 a4 a4 a4 a5 a5 a5
 a2 a2 a2 a2 a2 a4 a5 a5 a5 a5
Weights:
1'b': (1.0558187579829825, -0.010476779296380263)
2'a': (0.006631935199328441, 2.397458821967359E-4)
3'b': (0.7652403553775214, 0.087610992407598)
4'c': (0.16636756585566323, 0.9643139783771884)
5'c': (0.007653507532642167, 0.8658900216549744)
6'c': (-0.07265270654364106, 1.1097474451785376)

Contexts:
1'.': (2.302867683293782, 7.060411847855459)
2'.': (6.91428845718414, 2.2022691476646266)
3'.': (6.815194629844695, 7.126760082785038)
4'.': (4.809546976780184, 1.560450690935124)
5'.': (1.6087880250352866, 2.412826455495859)
```

```
02mat-01noise2.som Order:2
 a1 a1 a1 a1 a1 a4 a4 a4 a4 a4
 a1 a1 a1 a1 a1 a4 a4 a4 a4 a4
 a1 a1 a1 a1 a1 a4 a4 a4 a4 a4
 a1 a1 a1 a1 a1 a4 a4 a4 a4 a4
 c1 c1 a1 a1 a1 a4 a4 a4 a4 a4
 c1 c1 c3 c3 a4 b3 b3 b3 b5 b5
 c3 c3 c3 c3 a3 b3 b3 b5 b5 b5
 c3 c3 c3 c3 b3 b3 b3 b5 b5 b5
 c3 c2 c3 c2 c3 b3 b5 b5 b5 b5
 c2 c2 c2 c2 b3 b5 b5 b5 b5 b5
Weights:
1'a': (0.10536358861745965, 0.10938175075912818)
2'c': (0.014988830231233092, 0.9389413644498548)
3'a': (-0.033243192396090894, -0.045735013905217635)
4'b': (0.9209938873430241, 0.040042759478397236)
5'a': (0.08437835571460163, 0.04199590369916233)
6'a': (-0.06635564852930395, 0.08129740324561974)

Contexts:
1'.': (7.276905162268173, 6.722092301779163)
2'd': (0.9534709459289685, 1.3099140722193248)
3'.': (3.134193037326341, 2.0762734923800465)
4'.': (1.7170587797259331, 6.997094384443843)
5'.': (6.622775316484559, 2.0657472792777742)

03mat-01noise5.som Order:2
 b1 b1 b1 b4 a4 a4 a4 a4 a4 a4
 b1 b1 b1 b4 a4 a4 a4 a4 a4 a4
 b2 b2 b2 b4 a4 a4 a4 a4 a4 a4
 b2 b2 b2 b2 a4 a4 a4 a4 c5 c5
 b2 b2 b2 a2 a4 a4 c5 c5 c5 c5
 b2 b2 b2 a2 a3 a5 c5 c5 c5 c5
 b2 b2 b3 a3 a3 a3 c5 c5 c5 c5
 b3 a3 a3 a3 a3 a3 c3 c5 c5 c5
 a3 a3 a3 a3 a3 a3 c3 c6 c6 c6
 a3 a3 a3 a3 a3 a3 c6 c6 c6 c6
Weights:
1'b': (0.8981778007777791, -0.012402205712438986)
2'a': (0.041791925455790795, -0.06787261996347896)
3'a': (0.020878726152319393, 0.05293878254392084)
4'c': (0.002103257046607663, 0.8946393436577149)

Contexts:
1'.': (3.043518209876521, 7.469609179008686)
2'.': (6.307964573050545, 1.6252555816043404)
3'.': (1.484578888447978, 2.866876868481626)
4'.': (7.611850640837112, 6.631182770696962)
5'.': (3.290443105500681, 7.370773860689866)
6'.': (5.925755500140937, 1.561101573190284)
```

```
04mat-01noise17.som Order:2
 a1 a1 a1 a1 a1 b5 b5 b5 b5 b5
 a1 a1 a1 a1 a1 b5 b5 b5 b5 b5
 a1 a1 a1 a1 a1 a1 b1 b6 b6 b6
 a1 a1 a1 a1 a1 a1 b6 b6 b6 b6
 c2 c2 a1 a1 a1 a6 b6 b6 b6 b6
 c2 c2 c2 c2 a4 a4 a4 a4 b6 b6
 c2 c2 c2 c2 a4 a4 a4 a4 a4 a4
 c2 c2 c3 c3 a4 a4 a4 a4 a4 a4
 c3 c3 c3 c3 c3 a4 a4 a4 a4 a4
 c3 c3 c3 c3 c3 a4 a4 a4 a4 a4
Weights:
1'a': (-0.048946749652464934, 0.1000986965426045)
2'c': (-0.0064775629278793646, 0.9018123646064607)
3'b': (0.9357008791204846, -0.020272181750462984)
4'a': (0.10000983835065635, 0.023724008512199152)

Contexts:
1'.': (7.301942500900722, 2.0867521097234176)
2'.': (6.612875990950398, 6.14498555948755)
3'.': (2.1664801135101355, 2.3815792912546874)
4'.': (1.4793657227605237, 6.824459712046185)
5'.': (6.807390987220609, 7.017719194520284)
6'.': (2.3380203364751444, 2.4402023392545695)


05mat-01noise2.som Order:2
 b1 b1 b1 b4 b3 b3 b3 a6 a6 a6
 b1 b1 b1 b4 b4 b3 a6 a6 a6 a6
 b1 b1 b1 b4 b4 b4 a6 a6 a6 a6
 b1 b1 b1 b4 b4 b4 a6 a6 a6 a6
 a1 a2 b2 a2 a4 a6 a6 a6 a6 a6
 a2 a2 a2 a2 a2 a4 a6 a6 a6 a6
 a2 a2 a2 a2 a2 c5 a6 a6 c6 c7
 a2 a2 a2 a2 c5 c5 c5 c7 c7 c7
 a2 a2 a2 a5 c5 c5 c5 c7 c7 c7
 a2 a2 a2 c5 c5 c5 c5 c7 c7 c7
Weights:
1'b': (0.9373413558661678, -0.0014756026899730018)
2'a': (0.04163304457008619, -0.1020714291791832)
3'c': (-0.0033794173354229585, 0.9302723799508502)
4'a': (0.04017429231871855, 0.05015227905411029)

Contexts:
1'.': (7.549944584763594, 2.9192212106118607)
2'.': (6.831052508066119, 7.815083627229312)
3'.': (0.7747102920198449, 7.475167756688364)
4'.': (2.546574420416277, 5.4635578211505065)
5'.': (7.285142169122498, 3.488851629546919)
6'.': (2.4644406872674187, 1.701439733834245)
7'.': (1.725720172113903, 6.519714620849484)
```

```
06mat-01noise2.som Order:2
 a1 a1 a1 b3 b3 b3 b3 b3 b6 b6
 a1 a1 a1 a1 b3 b3 b3 b3 b6 b6
 a1 a1 a1 a1 b3 b3 b3 b3 b6 b6
 a1 a1 a1 a1 a1 a3 b3 b3 b6 b6
 a1 a1 a1 a1 a2 a5 a5 a5 a5 b6
 a1 a1 a1 a2 a5 a5 a5 a5 a5 a5
 c2 c2 c2 c2 a5 a5 a5 a5 a5 a5
 c2 c2 c2 c4 c4 c4 a5 a5 a5 a5
 c2 c2 c2 c4 c4 c4 a5 a5 a5 a5
 c2 c2 c2 c4 c4 c4 c4 a5 a5 a5
Weights:
1'a': (-0.013355854558105083, -0.07680111704327852)
2'c': (-0.004302918734329331, 0.9673045875147569)
3'a': (0.05858458308066061, 0.1701789443805327)
4'b': (0.9335647419092231, -3.907935848390734E-4)
5'a': (0.20169612441704812, 0.009606628915757642)
6'a': (-0.032206165483771004, -0.04768035213339161)

Contexts:
1'.': (2.2731092291212787, 7.50853917713189)
2'.': (2.0014688066039845, 2.6508473442396716)
3'.': (6.466894412304316, 6.055348826882835)
4'.': (7.164114969665171, 6.242279780578055)
5'.': (6.800999029795852, 1.446601940055267)
6'.': (1.7189946570229413, 2.4203267321185584)

07mat-01noise2.som Order:2
 c1 c1 c3 c3 c3 c3 c3 a4 a4 a4
 c1 c1 c3 c3 c3 c3 c3 a4 a4 a4
 c1 c1 c3 c3 c3 c3 a3 a4 a4 a4
 a2 c2 c2 c3 c3 c4 a4 a4 a4 a4
 a2 a2 a2 a2 a3 a4 a4 a4 a4 a4
 a2 a2 a2 a2 a2 a4 a4 a4 a4 a4
 a2 a2 a2 a2 a2 b5 b5 b5 a4 a4
 a2 a2 a2 a2 b5 b5 b5 b5 b6 b6
 a2 a2 a2 b2 b5 b5 b5 b5 b6 b6
 a2 a2 a2 b2 b5 b5 b5 b5 b6 b6
Weights:
1'c': (-0.014066343924137043, 0.9291089366417165)
2'a': (-0.08536835550411886, -0.04110793949369925)
3'b': (0.9060852077204713, -0.006083234873623799)
4'a': (0.05232031805600305, 0.05037132493564258)

Contexts:
1'.': (7.713765753328667, 2.7942206339210536)
2'.': (3.199387833518634, 1.2349576811269491)
3'.': (1.6775832395379218, 6.271576556298676)
4'.': (6.174974947381409, 7.955074715083675)
5'.': (7.54488387071345, 3.617433211444732)
6'.': (1.6723295708118222, 6.170620408302824)
```

```
08mat-01noise6.som Order:2
 a1 a1 a1 a1 a1 a4 a4 a4 a5 a5
 a1 a1 a1 a1 a1 a4 a4 a4 a5 a5
 a1 a1 a1 a1 a1 a4 a4 a5 a5 a5
 a1 a1 a1 a1 a1 a4 a4 a4 a5 a5
 b2 b2 b2 b1 a1 a4 a4 a4 a5 a5
 b2 b1 b2 b2 b1 a4 a4 a4 a4 c4
 b2 b2 b2 b2 b2 c3 c3 c3 c3 c3
 b2 b2 b2 b2 b2 c3 c3 c3 c3 c3
 b2 b2 b3 b3 c3 c3 c3 c3 c3 c3
 b2 b3 b3 b3 b3 c3 c3 c3 c3 c3
Weights:
1'b': (0.9281796757788578, 0.05412447734998191)
2'c': (0.02498093155122629, 0.9689316681128546)
3'a': (0.01772841608059241, 0.031871440288560565)

Contexts:
1'.': (1.2896368805343343, 5.391968425245038)
2'.': (3.297915617265121, 1.3959968479664897)
3'.': (6.9201886695410115, 1.7648029667947798)
4'.': (4.7477116212538775, 6.861503032689587)
5'.': (7.976948469640918, 7.520947928321123)
```

Clustering-Algorithm for Cartesian grids:

recall that N is the set of neurons on the SOMSD
naming convention: if U is a matrix of size |N|, $u(n_j)$ denotes the value of the matrix for n_j for all $j \leq |N|$

let U be the U-Matrix of a SOMSD
let C be a matrix of size |N|, initialized with zeros.

// choice of the 'water level'
let L be a list of all $u(n_j)$ in U in increasing order
let act be 0.5 * (Number of winning neurons)
let waterLevel bet he act'th entry in L.
// clustering of 'flooded' neurons
set nextCluster = 1
for each neuron n_j starting in the top left corner of the map
    if $u(n_j)$ < waterLevel
        set l = the cluster of the left neighbour if present, 0 otherwise
        set o = the cluster of the above neighbour, 0 otherwise
        if l = o = 0
            set $c(n_j)$ = nextCluster
            nextCluster = nextCluster + 1
            continue with the next neuron
        if l = 0
            set $c(n_j)$ = o
            continue with the next neuron
        if o = 0
            set $c(n_j)$ = l
            continue with the next neuron
        if l ≠ o
            replace all occurrences of o in C by l

```
                    set c(n_j) = l
                    continue with the next neuron
//clustering of the above-level neurons
  let m(i) for each cluster number i occurring in C be the average weight over all
     neurons n_j with c(n_j) = i.
let changed = true;
while changed = true:
        changed = false
        for each neuron n_j starting in the top left corner of the map
                set d = w_j.distanceTo(m(c(n_j)) //Euclidian distance
                set l = the cluster of the left neighbour if present, 0 otherwise
                set r = the cluster of the right neighbour if present, 0 otherwise
                set a = the cluster of the above neighbour if present, 0 otherwise
                set b = the cluster of the below neighbour if present, 0 otherwise
                for all x in {l, r, a, b}
                        if d > w_j.distanceTo(m(x))
                                changed = true
                                c(n_j)= x
                                setze d = w_j.distanceTo(m(c(n_j)
                repeat with the next x
        repeat with the next neuron
repeat
```

15. Acknowledgments

I thank Barbara Hammer for interested, motivated and motivating mentoring, Marc Strickert for helpful advice as well as material for Figure 9 and Nicolas Neubauer for providing the GSOMSD Java classes and giving support on them.

16. References

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

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

[3] Eyesenck, Keene. *Cognitive Psychology - A Student's Handbook*. Psychology Pres, 2000.

[4] N. Chomsky. *Strukturen der Syntax*. Mouton & Co., 1965.

[5] Gazzaniga, A, editor. *Cognitive Neuroscience: A Reader*. 2000.

[6] Hebb, D. *The Organization of Behavior: A Neuropsychological Theory*. John Wiley and Sons, 1949.

[7] T. Martinetz, S.G. Berkovich and K.J. Schulten. 'Neural-gas' networks for vector quantization and its application to time-series prediction. *IEEE Transaction on Neural Networks*, 4(4):558-569, 1993.

[8] M. Hagenbuchner, A. Sperduti and A.C. Tsoi. A Self-Organizing Map for Adaptive Processing of Structured Data. *IEEE Transactions on Neural Networks*. 14(3):491-505, 2003.

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

[10] B. Hammer, A. Micheli, M. Strickert, A. Sperduti. A general framework for unsupervised processing of structured data. To appear in *Neurocomputing*.

[11] Kandel, E.-R., Schwartz, J., Jessel, T. Priciples of Neural Science. McGraw-Hill, 2000.

[12] T. Koskela, M. Varsta, J. Heikkonen and K. Kaski. Recurrent SOM with local linear models in time series prediction. In M. Verleysen, editor, *6th European Symposium on Artificial Neural Networks*, 167-172, De facto, 1998.

[13] T. Kohonen. Self-Organizing Maps. Springer, 1995.

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

[15] H. Ritter, T. Martinetz and K. Schulten. *Nerual Computation and Self-Organizing Maps: An Introduction*, Addison-Wesley, 1992

[16] M.Strickert, B.Hammer. Unsupervised recursive sequence processing. In M.Verleysen, editor, *11th European Symposium on Artificial Neural Networks'2003,* 27-32, D-side publications, 2003.

[17] M. Strickert, B. Hammer. Neural Gas for Sequences. Submitted to the *Workshop on Self-Organizing Networks (WSOM'03)*, 2003.

[18] J. Tsien, P. Huerta, S. Tonegawa. The Essential Role of Hippocampal CA1 NMDA Receptor-Dependent Synaptic Plasticity in Spatial Memory. *Cell.* 87: 1327-1338, 1996

[19] Ultsch, A.. Data Mining and Knowledge Discovery with Emergent Self-Organizing Feature Maps for Multivariate Time Series . In: Oja, E., Kaski, S., editors, *Kohonen Maps*, 33 - 46, 1999.

[20] T. Voegtlin. Recursive self-organizing maps. *Neural Networks* 15(8-9):979-991, 2002.

[21] G. Wunsch, H. Schreiber. *Stochastische Systeme*. VEB Verlag Technik, 1984.

[22] E. Pampalk, A. Rauber, and D. Merkl. Content-based Organization and Visualization of Music Archives. In: *Proceedings of the ACM Multimedia 2002*, pp 570-579, 2002.

[23] F. Ollermann, K. Hamborg, S. Reinecke. Software Evaluation with Eye Tracking. Conference tutorial at *The European Cognitive Science Conference EuroCogSci 03*. 2003.