Online Algorithm for the Self-Organizing Map of Symbol Strings

Panu J. Somervuo
Neural Networks Research Centre
Helsinki University of Technology
P.O.Box 5400, FI-02015 HUT, Finland
tel. +358 9 451 3284, fax +358 9 451 3277
email: panu.somervuo@hut.fi

Acknowledgement: This work was supported by Academy of Finland, project no. 44886 "New information processing principles" (Finnish Centre of Excellence Programme 2000-2005).

Online Algorithm for the Self-Organizing Map of Symbol Strings

Abstract: In this work an online algorithm is presented for the construction of the Self-Organizing Map (SOM) of symbol strings. Each node of the SOM grid is associated with a model string which is a variable-length vector sequence. Smooth interpolation method is applied in the training which performs simultaneous adaptation of the symbol content and the length of the model string. The efficiency of the method is demonstrated by the clustering of a 100.000-word English dictionary.

Keywords: dynamic time warping, self-organizing map, subsymbolic representation, symbol string, string average

Nomenclature:

$\mathcal{A}$	symbol alphabet
$\alpha(t)$	learning rate parameter
$\sigma(t)$	effective width of the neighborhood function
$c(X_t), c$	index of best-matching unit
$D(X_t, M_k)$	distance between two vector sequences
$d(\cdot,\cdot), d[z]$	distance between two vectors
F	warping function
$g(i, j)$	trellis variable, cumulative distance
$h(c, k, t), h$	neighborhood function
L	length of sequence
$l(p)$	sequence element position
M_k	model associated with map node k, vector sequence
$\mathbf{m}_{kj}$, $M_k(j(p))$	jth vector of M_k
p	index for a point in warping function
q	interpolation weight
S_t	symbol string
t	time index
X_t	encoded symbol string, vector sequence
$\mathbf{x}_{ti}$, $X_t(i(p))$	ith vector of X_t
z	index pair

1 Introduction

Symbol strings are encountered in many application fields like speech recognition and bioinformatics. This work describes an online algorithm which allows the construction of the Self-Organizing Map (SOM) for symbol strings with smooth symbol representation and averaging.

The SOM is an unsupervised method for forming a representation of data [6, 8]. It consists of local data models located at the nodes of the low-dimensional map grid. The models are adapted via competitive learning process.

There are two approaches for constructing the SOM for a new data set. One approach is to convert the data into fixed-dimensional feature vectors so that the comparison and averaging can be done using familiar Euclidean distance and arithmetic averaging. Another possibility is to modify the SOM algorithm itself. In the current work, the SOM algorithm is very close to its original online formulation. The only difference is that instead of single fixed-dimensional feature vectors as input and models, the models associated with the SOM nodes are vector sequences with variable lengths [17].

In document clustering, a common approach is to use a "bag-of-words" representation. In this method, the word order in the document is ignored and each document is represented by a plain word histogram. Symbol strings could be also represented in the similar way using "bag-of-letters" type feature vectors or histograms of consecutive letters, N-grams. This would give a fixed-dimensional feature vector and the original SOM algorithm could be then used without any modifications. However, the approach in the present work is to take the order of the data string symbols into account. Another objective is to maintain the simplicity and elegance of the original online SOM algorithm.

A common method for comparing symbol strings is to use Levenshtein distance [13]. This takes both the content and the order of the symbols in the string into account. However, it is defined for discrete symbol strings only and does not allow soft subsymbolic representations. In order to compute averages between arbitrary symbols and perform gradual changes to the model strings on the SOM, a vector representation is used in the current work. Each data string is represented by a sequence of the vectors where each individual vector encodes one symbol position. In the current work an orthogonal symbol representation is used. Each symbol of the data string is converted into a vector whose dimension corresponds to the size of the alphabet. The length of the vector sequence representing the data string equals to the length of the original symbol sequence. Vector representation allows arithmetic averaging between the symbols and results in smooth model adaptation.

Sequences with varying lengths are compared using dynamic time warping (DTW) algorithm [15, 16]. Models are updated by computing a weighted average between the model string and the data string. The average of two vector sequences with unequal lengths is well-defined [14, 16]. Although the length of the model string can be considered as a model structure parameter, the current method allows it to be adapted simultaneously with the symbol contents.

The paper is organized as follows: Sec. 2 explains the background and related work to the SOM of symbol strings, Sec. 3 illustrates different string averages, Sec. 4 describes the online training algorithm, and Sec. 5 illustrates the new method with two examples which show how the symbol strings can be smoothly interpolated between sparse data and how the method scales well to large data sets.

2 Previous work related to the SOM of symbol strings

In [9] it was shown for the first time that the SOM can be constructed for any data set for which a similarity or dissimilarity measure between its elements is defined. This was based on the use of batch-SOM training and the definition of the generalized median as the model associated with each SOM node. The first application of this method was the construction of the SOM for symbol strings [9, 10, 11]. Two types of medians were used: the set median, which is an existing element of the data set, and the median, which does not have to be an exact replica of any element in the data set. The set median is applicable to dense data sets, while the latter kind of median may interpolate sparse data better. If the set medians are used, the models are always copies of some data items. The benefit of this is that one algorithm can be applied without any modifications to the new data sets. However, if the type of the data is restricted, e.g. the SOM is to be constructed for symbol strings as in the current work, more data-type oriented interpolation methods can be used.

The SOMs can be trained using the online or batch algorithm. Batch algorithm for the SOM of symbol strings has been used in [9, 10, 11, 12, 18] and online algorithms have been experimented in [1, 4]. The novelty of the present work contrasted with the previous works is the combination of vectorial symbol representation and the computation of the average between two symbol strings with unequal lengths. This method fits naturally to the online training.

In earlier work with the SOM of symbol strings [9, 10, 11, 12, 18] the data and the models in the SOM were considered as discrete items. Levenstein distance [13] and Redundant Hash Addressing (RHA) method based distance [5, 8] were then used. Quantized data and models may result that there emerge duplicate models on the SOM [12]. The proposed method for computing string average has no such problems. In the present work, the models in the SOM are able to interpolate smoothly between discrete data clusters. Only when the soft symbol representations are converted back to discrete symbols after SOM training, duplicates may occur.

3 Examples of string averages

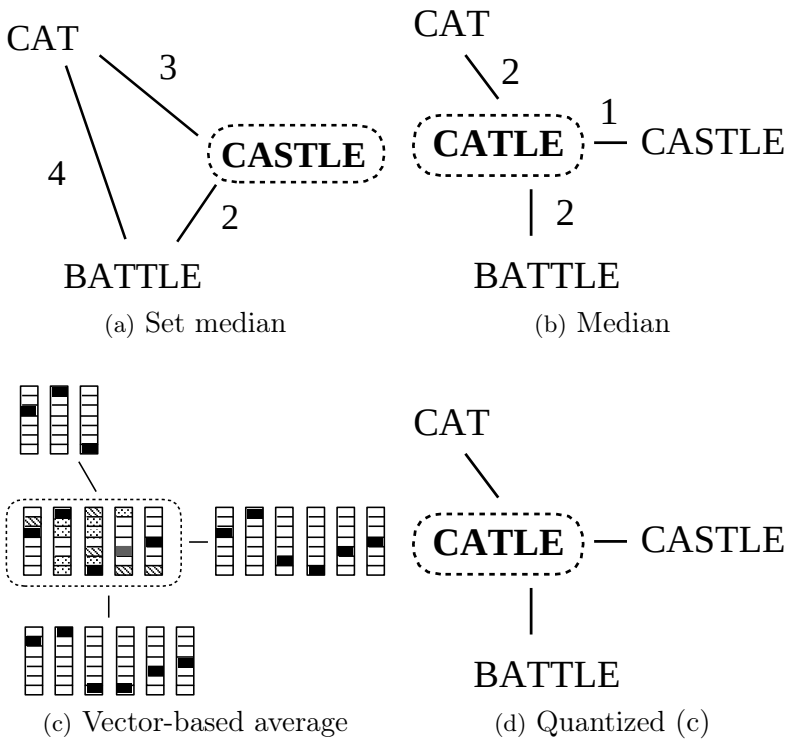
Two approaches for computing the string average has been presented in [7]. One of them, the set median, is defined to be the string which has the smallest sum of distances to other strings in the set. Set median is defined to be the string which is a member of the original data set while the general median string is defined as a symbol string which can be taken outside of the original data set. This fine-tuned median string can be obtained by systematically varying the symbols of the set median string position by position. The idea is to find the string which has the smallest sum of distances to the strings in the data set. The search can be computationally heavy if large data sets are used, since after each symbol variation, the distances between the new median candidate and the original data strings must be recomputed in order to evaluate if the change of symbol in the median string is beneficial. Set medians and general medians have been utilized in the construction of the SOM of symbol strings in [9, 10, 11]. Some modifications for computing the string average have been proposed in [4].

If the data item can be converted into feature vector, the interpolation and averaging can be computed by arithmetic means. This was the case in [2], where the symbol string data were clustered by the SOM. The data strings which represented protein sequences were first multiple-aligned and then converted into fixed-dimensional feature vectors. Multiple string alignment is a process where the parts of the strings are shifted to the left or right in respect to other strings by inserting gaps between symbols so that the maximum mutual coherence between the aligned

string set is attained. The coherence is measured by the position-wise symbol similarity. In [2] sequences consisted of amino-acid symbols. Because of the global string set alignment, all data could be converted into fixed-dimensional feature vectors. Each symbol was converted into 20-dimensional vector (there were 20 symbols in the amino-acid alphabet) and the length of the feature vector representing the entire string was 20 times the length of the global alignment. Since fixed-dimensional vector representation was used for all strings, Euclidean distance and arithmetic averaging could be used in the SOM training.

In [14], two clustering methods were considered for feature vector sequences. The methods were applied to speech recognition where the feature vector sequences represented isolated word utterances. The representative sequence of the cluster, the minimax center, was defined to be the sequence which had the smallest maximum distances to other sequences in the cluster. That approach was called clustering without averaging (UWA). Another method was also presented, clustering with full averaging (UFA), where the averaged sequence was obtained by averaging feature vectors of two sequences along their dynamic time warping alignment.

The current work combines the two abovementioned approaches by utilizing the vector coding of the symbols [2] and dynamic time warping when computing the distances and averages between the vector sequences [14].



(a) Set median (b) Median

(c) Vector-based average (d) Quantized (c)

Figure 1: Examples of string averages for the data set containing three strings: "CAT", "CASTLE", and "BATTLE". String average is shown encircled in the middle of each subfigure. The numbers in (a) and (b) are the Levenshtein distances between the strings. Set median (a) is the element of the data set which has the smallest sum of distances to other elements. Median string (b) is defined to be the string which has the smallest sum of distances to other strings of the data set but it does not have to be the member of the original data set. Vector-sequence based average (c) is computed by means of dynamic time warping where each symbol is represented by a vector. Vector-based string average can be converted into a discrete symbol string (d) by taking the symbol which corresponds to the largest vector element in each sequence position.

Different string averages are illustrated in Fig. 1. Set median in Fig. 1(a) and median in Fig. 1(b) are computed according to [7]. Vector-representation based average is shown in Fig. 1(c). Since there are seven symbols in the alphabet: A, B, C, E, L, S, and T, each symbol of the data string is converted into a seven-dimensional vector. The average vector sequence is computed by means of dynamic time warping [14, 16]. In Fig. 1(d), the vector-sequence average of Fig. 1(c) is converted into a discrete symbol string by selecting the symbol in each sequence position which corresponds to the largest vector element in that position. In this example, the result is equal to the median string of Fig. 1(b).

4 Online SOM algorithm for symbol strings

The SOM training consists of two steps which are iterated [6, 8]: 1) search of the best-matching unit (BMU) and 2) adaptation of the models based on the input.

The index of the BMU to the given input X_t is defined as:

$$c(X_t) = \arg\min_k D(X_t, M_k) \ , \tag{1}$$

where M_k denotes the model of the kth map node and $D(X_t, M_k)$ is the distance measure. This definition is valid regardless of the model and input representation. In the current work, where X_t is a data string converted into a vector sequence and M_k is the model of the data string which is also a vector sequence, the distance is computed according to Eq. (3).

4.1 Symbol string encoding

Symbol strings are encoded into vector sequences in the following way. Let $S_t = [S_t(1)S_t(2)\dots S_t(L)]$ denote the tth symbol string with length L. We now define the mapping $S_t \rightarrow X_t$, where X_t is the vector sequence: $X_t = [X_t(1)X_t(2)\dots X_t(L)]$. Let $\mathcal{A}$ be the symbol alphabet. $|\mathcal{A}|$ denotes the number of the elements in $\mathcal{A}$. Each symbol $S_t(i)$, $i = 1 \dots L$ is then encoded into $|\mathcal{A}|$-dimensional vector $X_t(i)$ so that all elements in $X_t(i)$ are zeros except one corresponding to the index of the symbol $S_t(i)$ in $\mathcal{A}$.

For example, if $\mathcal{A} = \{A, B, \dots, Z\}$ and $|\mathcal{A}| = 26$, the symbols in the alphabet are converted into vectors in the following way: $A \rightarrow [100\dots0]$, $B \rightarrow [010\dots0]$, $\dots$, $Z \rightarrow [000\dots1]$ and the converted symbol string is the sequence of the corresponding vectors. E.g., the string $S_t =$ "BAD" would be converted to $X_t = [0100\dots0]^T[1000\dots0]^T[0001\dots0]^T$.

4.2 Distance computation

Dynamic time warping addresses to the problem of finding an optimal alignment between two sequences. This can be efficiently computed by dynamic programming [15, 16]. By means of an optimal alignment, it is possible to define the distance between the sequences with unequal lengths.

Let X_t and M_k denote two vector sequences, i.e., two encoded strings, and F the warping function which performs the alignment. F is a sequence of index pairs $[i(p), j(p)]$ which defines the mapping between the elements i of sequence X_t and elements j of sequence M_k:

$$F = [z(1), z(2), \dots, z(p), \dots, z(P)] \ , \tag{2}$$

where $z(p) = [i(p), j(p)]$ and P is the number of the points in the alignment. DTW-based distance is computed as a sum of distances between the sequence elements along the alignment:

$$D(X_t, M_k) = \sum_{p=1}^{P} d[z(p)] \ , \tag{3}$$

where $d[z(p)]$ is the distance between the vectors of the sequences M_k and X_t indexed by $z(p)$. Euclidean distance $||X_t(i(p)) - M_k(j(p))||$ can be used. Here $X_t(i(p))$ denotes the single vector whose position is $i(p)$ in the vector sequence X_t and $M_k(j(p))$ is the single vector whose position is $j(p)$ in the vector sequence M_k. The cumulative distance in Eq. (3) can be divided by P or the sum of the sequence lengths. The number of the points in the alignment, P, depends on the contents of the sequences and it is determined by the dynamic programming process. Its maximum value is the sum of the two sequence lengths.

The warping function can be computed in a two-dimensional trellis. Let $g(i, j)$ denote the variable which stores the cumulative distance in each trellis point (i, j). The trellis is first initialized:

$$\begin{aligned} g(0, j) &= \begin{cases} 0 & j = 0 \\ \infty & j > 0 \end{cases} \\ g(i, 0) &= \begin{cases} 0 & i = 0 \\ \infty & i > 0 \end{cases} \end{aligned} \tag{4}$$

Dynamic programming then follows:

$$g(i, j) = \min \begin{cases} g(i, j-1) + d(i, j) \\ g(i-1, j-1) + d(i, j) \\ g(i-1, j) + d(i, j) \end{cases} , \tag{5}$$

where $d(i, j)$ is the distance between the vectors $X_t(i)$ and $M_k(j)$. Index i goes from 1 to the length of X_t and j from 1 to the length of M_k. In accordance with Eq. (3) we can now define:

$$D(X_t, M_k) = g(L_{X_t}, L_{M_k}) \ , \tag{6}$$

where L_{X_t} and L_{M_k} denote the lengths of X_t and M_k, respectively. Eq. (6) and Eq. (3) are equivalent. From $g(i, j)$ we can obtain the warping function $F = \{z(p)\}_{p=1}^{P}$ which minimizes the sum of distances in Eq. (3). As an initialization we set:

$$z(P) = (L_{X_t}, L_{M_k}) \ . \tag{7}$$

The rest of the warping function is obtained by backtracking the trellis:

$$z(p) = \arg \min_{(x,y) \in \{(i-1,j),(i-1,j-1),(i,j-1)\}} g(x, y) \tag{8}$$

where index p goes from $P - 1$ to 1. Because of the initialization of $g(i, 0)$ and $g(0, j)$, $z(1)$ will be $(1, 1)$. All values of $z(p)$ are number pairs refering to the integer-valued coordinates of the two-dimensional trellis. Fig. 2 illustrates the distance computation and determination of warping function.

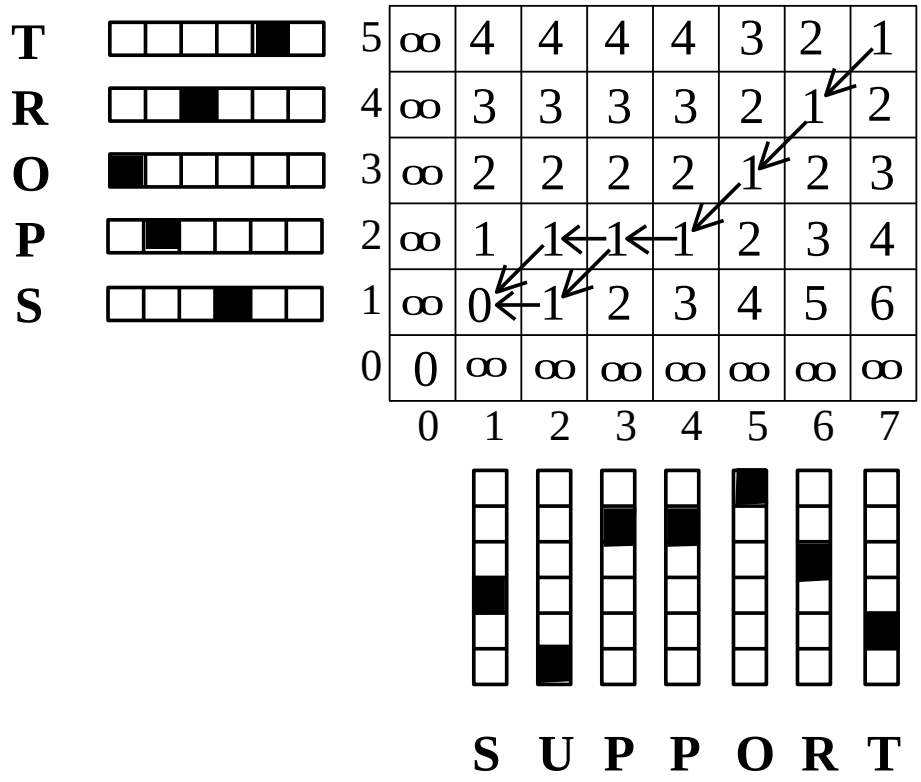


Figure 2: Distance computation between two strings "SPORT" and "SUPPORT" on a two-dimensional trellis. For simplicity each vector contains here only one 1 and the rest of the elements are 0. In the SOM training one of the vector sequences is a model associated with the map node and all of its elements can be nonzero. The values of the trellis grid are cumulative distances based on Eq. (5). The arrows starting from the top-right corner of the trellis illustrate the backtracking of the trellis according to Eq. (8). In the current case there are two equal optimal alignments (either z(p) = (2,1) or z(p) = (2,2) belongs to the warping function).

There are several modifications to the basic scheme described here which can be implemented, e.g., various slope constraints and weightings can be added to the warping function [15].

In case of symbols instead of feature vectors as the elements of the sequences, the corresponding distance is called Levenshtein or edit distance [13, 16]. The purpose of using vectors in this work is that they allow smooth symbol interpolation. This is explained in the following.

4.3 Model adaptation

In the model adaptation the length of the new model vector sequence is determined first and then the matching vectors of the input X_t and old model vector sequence M_k are averaged along the warping function F, Eq. (2).

Let us first investigate two vectors $\mathbf{m}_{kj} = M_k(j(p))$ and $\mathbf{x}_{ti} = X_t(i(p))$ from sequences M_k and X_t. The average of two vectors is simple to compute and we can use the model update formula of the original SOM algorithm:

$$\mathbf{m}'_{kj} = \mathbf{m}_{kj} + h(c,k,t)(\mathbf{x}_{ti} - \mathbf{m}_{kj}) \ , \tag{9}$$

where $h(c, k, t)$ denotes the neighborhood function which determines the learning rates of the models, t denotes the training cycle. Symmetric Gaussian neighborhood function can be used whose center is located at the BMU, $c = c(X_t)$. The entire sequences M_k and X_t are averaged by computing the sequence of weighted vector averages $(1 - h)\mathbf{m}_{kj} + h\mathbf{x}_{ti}$ along the alignment $F = \{z(p)\}_{p=1}^{P}$ of M_k and X_t . Shorthand notation h is used for $h(c, k, t)$.

Let $l(p)$ denote the "position" of the averaged sequence element. It is defined as the weighted average of the two indices in $z(p) = [i(p), j(p)]$:

$$l(p) = (1 - h)j(p) + hi(p) \ . \tag{10}$$

It is desirable that the length of the average sequence is in proportion to L_{M_k} and L_{X_t}, the lengths of the sequences M_k and X_t. Let us denote the updated model M_k by M'_k. We can determine its length:

$$L_{M'_k} = (1 - h)L_{M_k} + hL_{X_t} \ , \tag{11}$$

i.e., the length of the updated model sequence is a weighted average of its old length and the length of the current input. A reasonable sampling interval from $\{z(p)\}_{p=1}^{P}$ is that $l(n+1) - l(n)$ is one, where index n goes from 1 to $L_{M'} - 1$. The first value of l is set $l(1) = 1$.

If $l(n)$ does not coincide with any integer point of $z(p)$ it can be interpolated between two nearest integer points in the warping function. Let us denote the indices of these points by p_1 and p_2. They must satisfy:

$$l(p_1) < l(n) < l(p_2) \ . \tag{12}$$

The nth element of the new model sequence can then be interpolated between two vector averages $\mathbf{m}_1$ and $\mathbf{m}_2$ corresponding to the warping points $z(p_1)$ and $z(p_2)$:

$$\mathbf{m}'_{kj}(n) = \frac{q_1\mathbf{m}_1 + q_2\mathbf{m}_2}{q_1 + q_2} \ , \tag{13}$$

where linear interpolation weights q_1 and q_2 are determined as $q_1 = l(p_2) - l(n)$ and $q_2 = l(n) - l(p_1)$, and $\mathbf{m}_1$ and $\mathbf{m}_2$ are computed according to Eq. (9). Fig. 3 illustrates the averaging along the warping function.

In the model adaptation, there are two averages to be made. One is the average between the vectors and another is for the sequence lengths. The vector elements of the sequences will always get adapted even when very small learning rate values h are used. But if the length of the updated model sequence is rounded to the nearest integer then the length adaptation does not have any effect if the learning rate is small. However, the length of the updated model sequence M_k need not be integer-valued. This was found to be beneficial in the experiments. If the non-integer value $L_{M'_k}$ is used as the real value of the sequence length, the model length adaptation behaves smoothly. The sequence content must be then stored up to the rounded-up symbol position $\lceil L_{M'_k} \rceil$.

5 Experiments

The algorithm has similar behavior as the original SOM algorithm for single vectors. Using a wide updating neighborhood in the beginning of the training shrinks the map in the data space

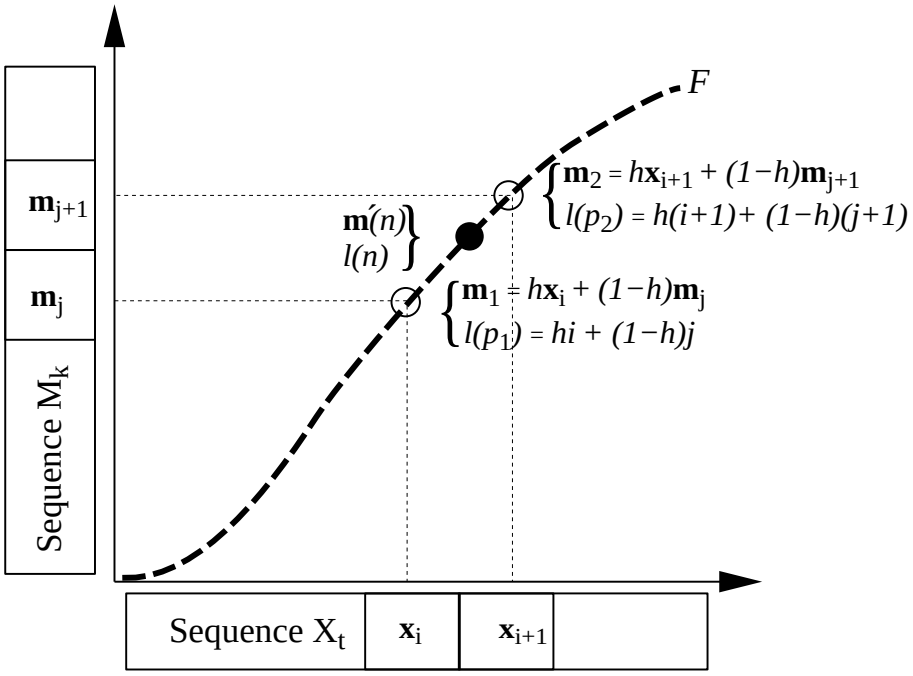


Figure 3: The alignment between two sequences is computed by DTW in a two-dimensional trellis. Warping function F is shown by dashed line. The updated model is obtained as a sequence of weighted averages sampled along F. Vector average $\mathbf{m}'(n)$ corresponding to the symbol position $l(n)$ can be interpolated between two vector averages $\mathbf{m}_1$ and $\mathbf{m}_2$ which correspond to the integer valued symbol positions of the sequences M_k and X_t.

and results in the initial ordering. The models become more specific and spread smoothly along the data manifold when the effective width of the neighborhood is decreased during the training.

In the following, two examples are given which illustrate the self-organization and clustering properties of the method for symbol strings.

5.1 SOM based string interpolation

In the first example, the idea is to show that the method results in a smooth SOM even in the case of sparse data. Input data consist of five words "CAT", "CATTLE", "BAT", "BATTLE", and "BATTLEFIELD". These strings can be considered as five separate data clusters which SOM tries to connect by a flexible grid.

Hexagonal SOM with 92 map units was used. The models were initialized by 26-dimensional random vectors whose values were between one and zero. Each vector element corresponded to one symbol in alphabet A...Z. The initial models were symbol sequences with the length of only one symbol. The map was trained in two stages with 1000 training cycles in each. The Gaussian neighborhood function was used in Eq. (9):

$$h(c(t),k,t) = \alpha(t)\exp[0.5d(c(t),k)^2/\sigma^2(t)] \ , \tag{14}$$

where $d(c(t),k)$ is the Euclidean distance between the coordinates of the nodes $c(t)$ and k on the map grid. Parameter $\alpha(t)$ controls the learning rate. In the first 1000 training cycles the effective width $\sigma(t)$ of the Gaussian neighborhood function decreased linearly from 10 to 3, and

$\alpha(t)$ decreased linearly from 0.5 to 0.01. In the second stage the neighborhood width was kept fixed, $\sigma(t)$ was 3, and another 1000 training cycles were performed during which $\alpha(t)$ decreased from 0.1 to 0.01.

The resulting SOM is shown in Fig. 4. After training, the models in the adjacent map nodes are similar and they span smoothly the data space. Because the lengths of the model sequences are adapted in addition to the vector elements, the models on the trained SOM correspond to the data strings despite their initialization.

The similarities between model strings in the neighboring map nodes are easy to observe when the subsymbolic string representations, i.e. vector sequences, are converted back into hard symbol strings. Each vector is represented by a symbol which corresponds to its largest component, see Fig. 4(b). The font size is used for representing the value of the largest vector component in each symbol position of the string.

Because of the neighborhood function, the SOM generates observations between disjoint data clusters and smoothly fills the gaps in the original data space. For instance, in Fig. 4(b), we see the model strings "CATTE" and "CATLE" between the model strings "CAT" and "CATTLE" which were not in the training data. By investigating adjacent model strings on the SOM and forming continuous paths on the map surface, we can create chains of small changes which convert one string smoothly into another.

5.2 SOM based string clustering

As a demonstration that the method scales well to large maps and large amounts of data, a SOM was trained for clustering an English dictionary with 100.000 words. This addresses the problem of how to generate a string taxonomy. The task is to divide a large dictionary into subdictionaries based on word similarities.

In the following experiment, a rectangular SOM with 50×50 nodes was used. This map size was arbitrarily chosen. In case of larger map, the model strings would be closer to the actual data strings (larger map yields lower quantization error) but the clustering tendency can be demonstrated adequately enough using this map size.

Symbols in the alphabet A...Z were converted into 26-dimensional unit-vectors. Those rare symbols which were outside the alphabet were converted into 26-dimensional vectors with all elements set to zero. Models were initialized to contain only one symbol position in the beginning of the training. Vector components were initialized by random numbers between zero and one. In the first stage of the training, 10.000 training cycles were performed during which the width $\sigma(t)$ of the Gaussian neighborhood function decreased linearly from 50 to 5 and $\alpha(t)$ decreased from 0.2 to 0.1. In the second training stage $\sigma(t)$ decreased from 5 to 1 and $\alpha(t)$ from 0.1 to 0.01 during 100.000 training cycles. Data strings were picked randomly from the data set in each training cycle.

After training, the entire data set was clustered by finding the BMU for each data string. Fig. 5 illustrates some of the clusters. Similar data strings tend to be mapped into neighboring map nodes. The gray scale image illustrates the lengths of the model strings. Since these regions are smooth, it can be seen that there is self-organization also based on the string lengths. The average model string length is 6.9 and the average length of data strings was 7.3. The distribution of the model string lengths has the shape of Gamma distribution with clipped tails. The shortest model string on the SOM has 4 symbols and the longest model string has 12 symbols. The lengths of the data strings were between 1 and 20 symbols.

6 Discussion

The distance between two sequences was defined by the cumulative distance between the sequence elements. In the present work, the individual symbols were converted into vectors using orthogonal representation and Euclidean distance was used for computing their dissimilarity. Other representations and distance functions are also possible. By changing the definition of similarity between data items leads to different kinds of clusterings and SOM representations. The choice for the best similarity or dissimilarity measure depends on the application.

Levenshtein distance is defined for discrete symbol strings to be the number of symbol changes required to convert one string into another. Some applications favor the use of weighted Levenshtein distance, where different symbol substitutions, deletions and insertions may have different weightings [19]. Similar approach can be embedded also in the vectorial representations of the symbols. Instead of the plain Euclidean distance, the weighted distance can be computed between two symbol representing vectors $\mathbf{a}$ and $\mathbf{b}$:

$$d(\mathbf{a}, \mathbf{b}) = [\mathbf{a} - \mathbf{b}]^T W [\mathbf{a} - \mathbf{b}] \ , \tag{15}$$

where W denotes the weighting matrix.

Another issue is the symbol representation. Instead of using vectors whose dimensionality corresponds to the size of the symbol alphabet, random projection methods can be used for reducing the dimensionality [3].

7 Conclusions

This work presented an online algorithm for the SOM with symbol strings as data. Symbol strings were converted into vector sequences which enabled the computation of smooth averages. Dynamic time warping was used for comparing the input strings against the model sequences and arithmetic averaging was used for updating the models.

The SOM method provides an ordered display by means of representative local averages of the data. Besides string taxonomy and clustering, the trained model strings on the SOM can be used as prototypes in approximative string matching.

The main advantage of the proposed method over the previous algorithms for constructing the SOM for symbol strings is the smooth interpolation ability due to the vectorial representation of the symbols. The online algorithm proposed in this work is computationally lighter compared to the batch-SOM algorithm used in earlier works which required the computation of pairwise distances between the data strings. Arithmetic averaging of the vector representations of the symbols along the DTW-based alignment is fast to compute. The current approach has the same benefits as the original online vector-SOM algorithm.

References

[1] Aaltonen, S. & Torkkola, K. (1989). Merkkijonojen itseorganisaatio. *Technical report Tik-61.198* (in Finnish), Helsinki University of Technology.

[2] Andrade, M., Casari, G., Sander, C., & Valencia, A. (1997). Classification of protein families and detection of the determinant residues with an improved self-organizing map. *Biological Cybernetics*, **76**, 441–450.

[3] Bingham, E. & Mannila, H. (2001). Random projection in dimensionality reduction: applications to image and text data. *Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2001)* (pp. 245–250).

[4] Fischer, I. & Zell, A. (2000). String Averages and Self-Organizing Maps for Strings", *Proceedings of the Second ICSC Symposium on Neural Computation (NC'2000)* (pp. 208-215).

[5] Kohonen, T. & Reuhkala, E. (1978). A very fast associative method for the recognition and correction of misspelt words, based on redundant hash addressing. *Proceedings of the 4th International Joint Conference on Pattern Recognition* (pp. 807–809).

[6] Kohonen, T. (1982). Self-Organized formation of topologically correct feature maps. *Biological Cybernetics*, **43**, 59–69.

[7] Kohonen, T. (1985). Median strings. *Pattern Recognition Letters*, **3**, 309–313.

[8] Kohonen, T. (1995). Self-Organizing Maps. Springer-Verlag.

[9] Kohonen, T. (1996). Self-Organizing Maps of Symbol Strings. *Technical Report A 42*, Helsinki University of Technology, Laboratory of Computer and Information Science.

[10] Kohonen, T. & Somervuo, P. (1997). Self-Organizing Maps of Symbol Strings With Application to Speech Recognition. *Proceedings of the Workshop on Self-Organizing Maps (WSOM'97)* (pp. 2–7).

[11] Kohonen, T. & Somervuo, P. (1998). Self-Organizing Maps of Symbol Strings. *Neurocomputing*, **21**, 19–30.

[12] Kohonen, T. & Somervuo, P. (2002). How to make large self-organizing maps for nonvectorial data. *Neural Networks*, **15**, 945–952.

[13] Levenshtein, V. (1966). Binary codes capable of correcting deletions, insertions, and reversals. *Cybernetics and Control Theory*, **10**, 707–710.

[14] Rabiner, L. & Wilpon, J. (1979). Considerations in applying clustering techniques to speaker-independent word recognition. *Journal of Acoustical Society of America*, **66**, 663–672.

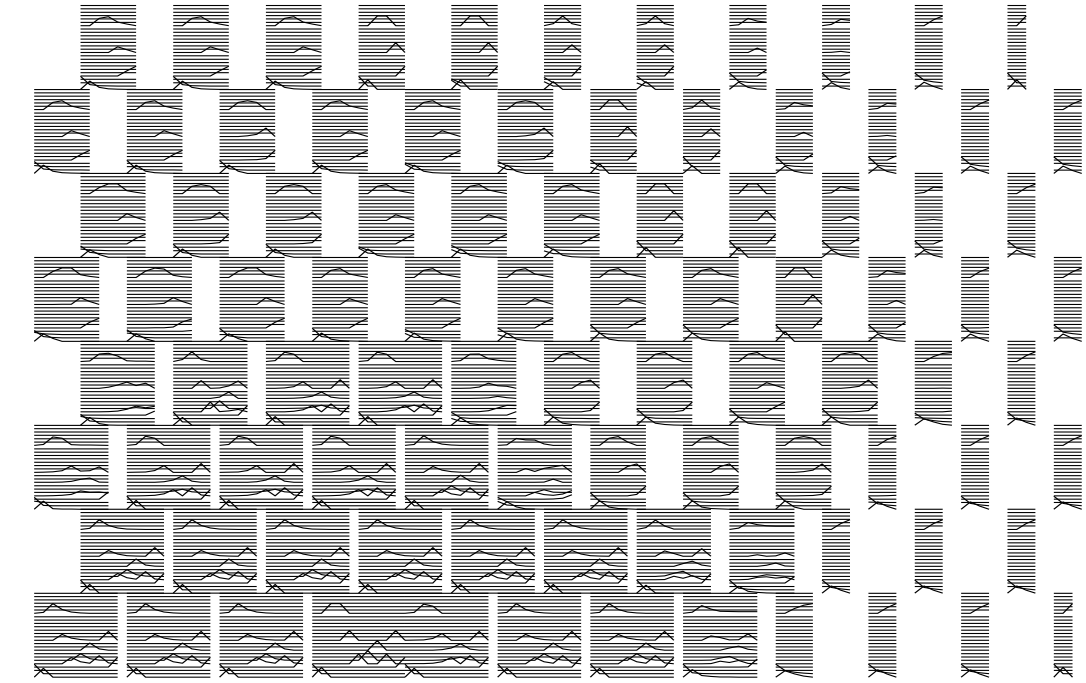
[15] Sakoe, H. & Chiba, S. (1978). Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, **26**, 43–49.

[16] Sankoff, D. & Kruskal, J. (1983). Time warps, string edits, and macromolecules: the theory and practice of sequence comparison. Addison-Wesley.

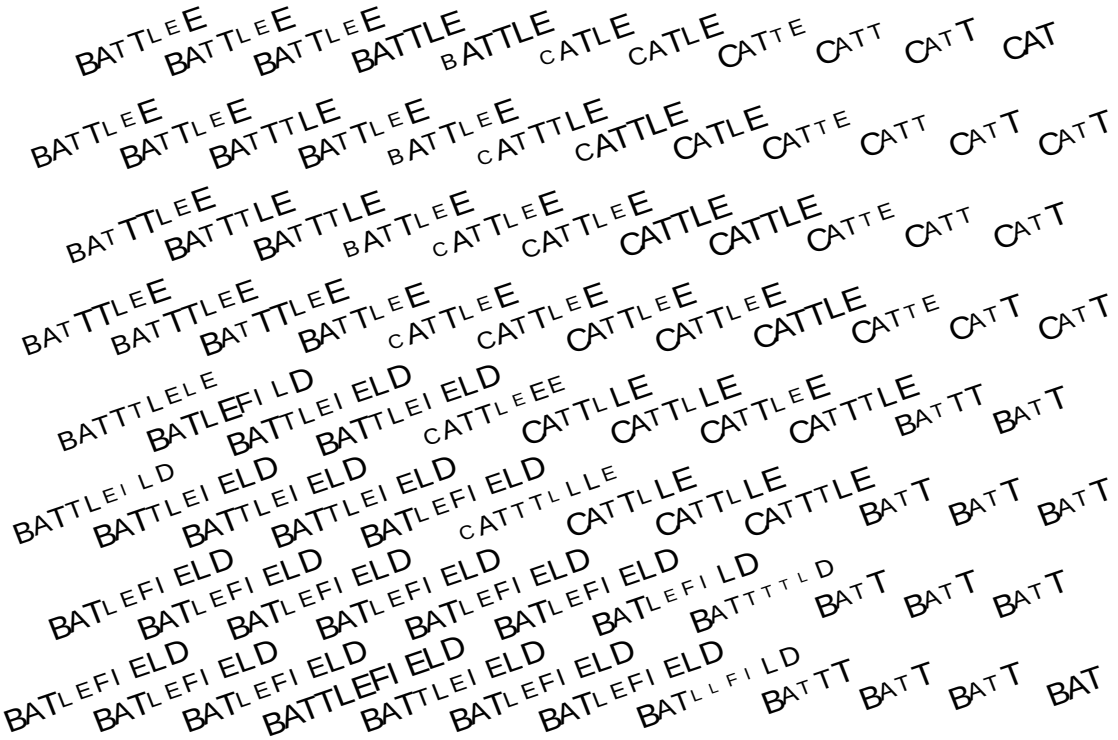
[17] Somervuo, P. & Kohonen, T. (1999). Self-organizing maps and learning vector quantization for feature sequences. *Neural Processing Letters*, **10**, 151–159.

[18] Somervuo, P. & Kohonen, T. (2000). Clustering and Visualization of Large Protein Sequence Databases by Means of an Extension of the Self-Organizing Map. *Proceedings of the Discovery Science (DS'2000), Lecture Notes in Artificial Intelligence 1967*, (pp. 76–85).

[19] Tanaka, E. & Kasai, T. (1976). Synchronization and substitution error-correcting codes for the Levenshtein metric. *IEEE Transactions on Information Theory*, **22**, 156–162.



(a) Soft symbol sequence models



(b) Quantized symbol sequence models

Figure 4: String interpolation using two-dimensional 92-unit SOM. The hexagonal map grid has eight units along vertical axis and 11 and 12 units in alternating rows along the horizontal axis. Input data consisted of five strings: "CAT", "CATTLE", "BAT", "BATTLE", and "BATTLEFIELD". Subfigure (a) shows the string models, 26-dimensional vector sequences, where each column vector element corresponds to one symbol in the alphabet A...Z and each vector corresponds to one symbol position. Heights of the lines reflect the values of 26 symbols in each sequence position. Vector sequences have been converted into discrete symbol strings (b) by selecting the symbol with the largest value in each vector. Font size indicates the dominance of the symbol in the sequence position and corresponds to the height of the line in (a).

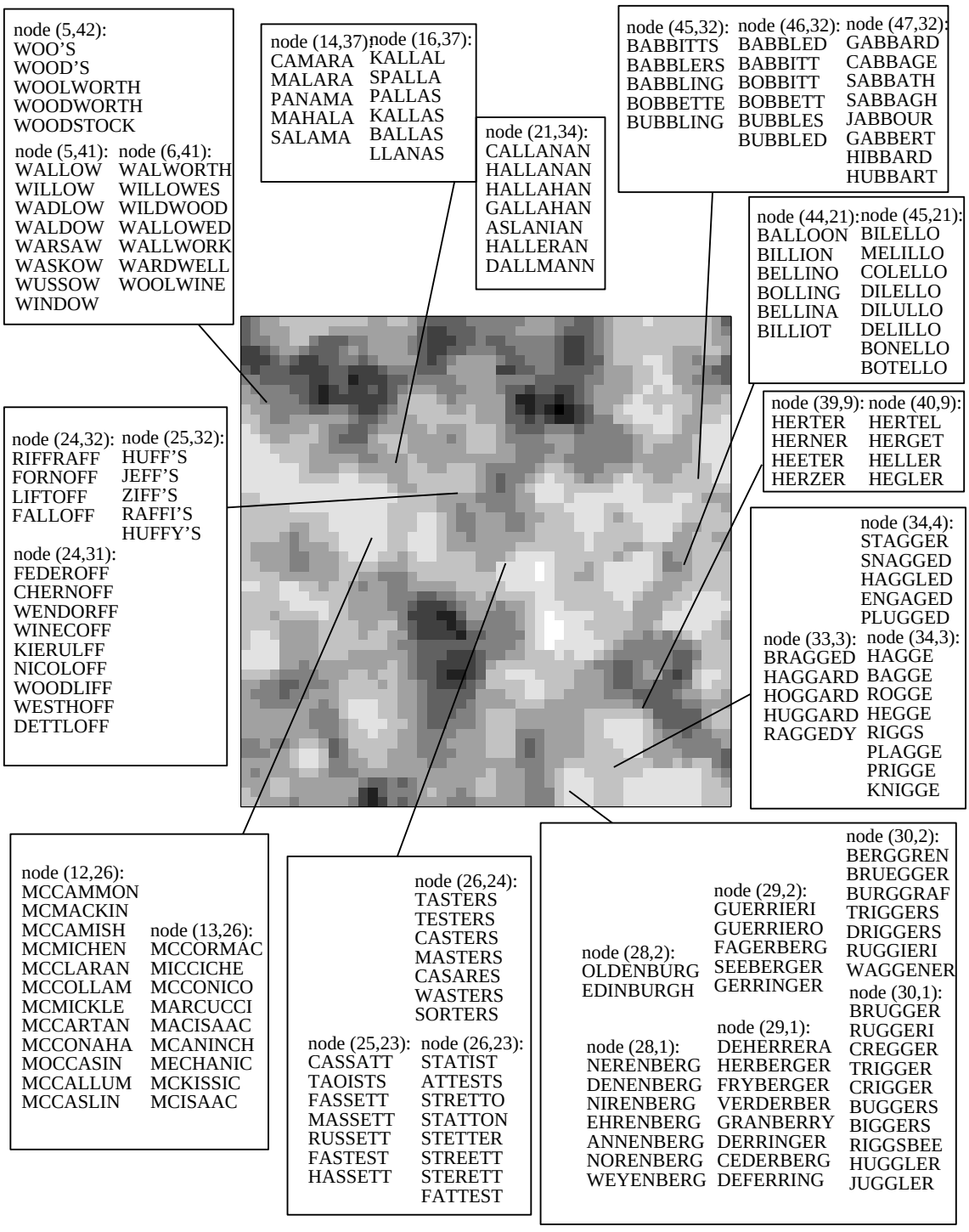


Figure 5: String clustering on a 50-by-50-unit SOM. Input data were a 100.000-word English dictionary. Shades of gray on the map represent the lengths of model strings which were between 4 (light gray) and 12 (dark gray). The data were mapped to their BMUs and some example clusters are shown from different parts of the SOM. In each data list, under the x-y coordinate of the node, there are data strings sorted according to their quantization error to the model string of the node. It can be seen that similar data strings are mapped to the neighboring map nodes.