AN ON–LINE METHOD FOR SEGMENTATION AND IDENTIFICATION OF NON–STATIONARY TIME SERIES

Jens Kohlmorgen, Steven Lemm

GMD FIRST.IDA
German National Research Center for Information Technology
Institute for Computer Architecture and Software Technology
Kekuléstr. 7, D-12489 Berlin, Germany
E-mail: {jek, lemm}@first.gmd.de
Web: http://www.first.gmd.de

Abstract. We present a method for the analysis of non-stationary time series from dynamical systems that switch between multiple operating modes. In contrast to other approaches, our method processes the data incrementally and without any training of internal parameters. It straightaway performs an unsupervised segmentation and classification of the data on-the-fly. In many cases it even allows to process incoming data in real-time. The main idea of the approach is to track and segment changes of the probability density of the data in a sliding window on the incoming data stream. An application to a switching dynamical system demonstrates the potential usefulness of the algorithm in a broad range of applications.

INTRODUCTION

Alternating dynamics is ubiquitous in real-world systems like, for example, speech, climatological data, physiological recordings (EEG/MEG), industrial processes or financial markets. Methods for the analysis of time-varying dynamical systems, which, aside from being non-stationary, might possibly be also non-linear, are therefore an important issue for many application areas. In [10], we introduced the annealed competition of experts (ACE) method for time series from non-linear switching dynamics, where an ensemble of neural network predictors specializes on different dynamical regimes by increasing the competition among the predictors through a deterministic annealing scheme. Related approaches for switching dynamics were presented, e.g., in [2, 5, 7, 11]. For a brief review of some of these models, their advantages and drawbacks, see [4].

Compared to the aforementioned work, we here present a different approach in two respects. First, the segmentation does not depend on the predictability of the system. Instead, we merely estimate the density distribution of the data and track its changes. This is particularly an improvement for systems where data is hard to predict, like, for example, EEG recordings [6] or financial data. Here, prediction based methods might possibly fail to provide an adequate representation of the underlying system, although they nevertheless might yield reasonable segmentation results in some cases [6]. Second, and this was actually the main motivation for this work, it is an *on-line* method. An incoming data stream is processed incrementally while keeping the computational effort limited by a fixed upper bound, i.e. the algorithm is able to perpetually segment and classify data streams with a fixed amount of memory and CPU resources. It even permits to continuously monitor measured data in *real-time*, as long as the sampling rate is not too high.[1] The main reason for achieving a high on-line processing speed is the fact that the method, in contrast to the approaches above, does not involve any training, i.e. iterative adaptation of parameters. Instead, it optimizes the segmentation on-the-fly by means of dynamic programming [1], thereby allowing for an automatic correction or fine-tuning of previously estimated segmentation bounds.

THE ALGORITHM

We consider the problem of continuously segmenting a data stream on-line and simultaneously labeling the segments. The data stream is supposed to have a *temporal structure* as follows: it is supposed to consist of consecutive blocks of data in such a way that the data points in each block exhibit a common feature, e.g., they might belong to a common cluster or, and this is where we focus on, they might stem from the same underlying distribution. The segmentation task is to be performed in an unsupervised fashion, i.e. without any a-priori given labels or segmentation bounds. In order to allow for a perpetual processing of data, potentially even in real-time, the problem should moreover be solved with a fixed amount of memory and computation time.

Feature extraction with pdfs

Let $\vec{y}_1, \vec{y}_2, \vec{y}_3, \ldots$, with $\vec{y}_t \in R^n$, be an incoming data stream to be analyzed. The sequence might have already passed a pre-processing step like filtering or sub-sampling.[2] As a first step of further processing we exploit an idea from dynamical systems theory and *embed* the data into a higher-dimensional

[1] In our reported application we can process data at 550 Hz (250 Hz including display) on a 750-MHz Pentium-III under Linux, which we expect is sufficient for a large number of applications.

[2] As long as this can be done on-the-fly in case of an on-line scenario.

space, which in principle aims to reconstruct the state space of the underlying system,

$$\vec{x}_t = (\vec{y}_t, \vec{y}_{t-\tau}, \ldots, \vec{y}_{t-(m-1)\tau}). \tag{1}$$

The parameter m is called the embedding dimension and τ is called the delay parameter of the embedding. The dimension of the vectors $\vec{x}_t$ thus is $d = m\ n$. The idea behind embedding is that the measured data might be a potentially non-linear projection of the systems state or phase space. In any case, an embedding in a higher-dimensional space might help to resolve structure in the data, a property which is exploited, e.g., in scatter plots. After the embedding step one might perform a sub-sampling of the embedded data in order to reduce the amount of data for real-time processing.[3]

As the next step, we want to track the density distribution of the embedded data. For this purpose we estimate the probability density function in a sliding window of length W by using a standard density estimator with multivariate Gaussian kernels [3] centered on the data points[4] in the window $\{\vec{x}_{t-w}\}_{w=0}^{W-1}$,

$$p_t(\mathbf{x}) = \frac{1}{W} \sum_{w=0}^{W-1} \frac{1}{(2\pi\sigma^2)^{d/2}}\ \exp\left(-\frac{(\mathbf{x} - \vec{x}_{t-w})^2}{2\sigma^2}\right). \tag{2}$$

The kernel width σ acts as a smoothing parameter and its value is important to obtain a good representation of the underlying distribution. We propose to choose σ proportional to the mean distance of each $\vec{x}_t$ to its first k nearest neighbors, averaged over a sample set $\{\vec{x}_t\}$.

Measuring similarity between pdfs

Once we have sampled enough data points to compute the first pdf according to eq. (2), we can compute a pdf for each successive point in time as soon as a new data point is available. In order to quantify the difference between two such functions, f and g, we use the squared L_2-Norm, sometimes also called *integrated squared error* (ISE), which can be calculated analytically if f and g are mixtures of Gaussians as in our case. Let us first consider the case of two general mixtures $f = \sum_{i=1}^{F} \alpha_i f_i$ and $g = \sum_{j=1}^{G} \beta_j g_j$. We then get

$$\begin{aligned} d(f,g) &= \int (f-g)^2\, d\mathbf{x} \qquad (3)\\ &= \sum_{i,k}^{F} \alpha_i \alpha_k \int f_i f_k\, d\mathbf{x} - 2\sum_{i=1}^{F}\sum_{j=1}^{G} \alpha_i \beta_j \int f_i g_j\, d\mathbf{x} + \sum_{j,l}^{G} \beta_j \beta_l \int g_j g_l\, d\mathbf{x}. \end{aligned}$$

[3] In that case, our further notation of time indices would refer to the subsampled data.

[4] In the following we use $\vec{x}$ to denote a specific vector-valued *point* and $\mathbf{x}$ to denote a vector-valued *variable*.

The integral of the product of two multivariate, spherical Gaussian distributions, $f_i \sim \mathcal{N}(\vec{\mu}_i, \sigma_i^2 I_d)$ and $f_j \sim \mathcal{N}(\vec{\mu}_j, \sigma_j^2 I_d)$, is given by

$$\int f_i f_j \, d\mathbf{x} = \frac{1}{(2\pi(\sigma_i^2 + \sigma_j^2))^{d/2}} \exp\left(-\frac{(\vec{\mu}_i - \vec{\mu}_j)^2}{2(\sigma_i^2 + \sigma_j^2)}\right). \tag{4}$$

Inserting eq. (2) in (3) and using (4) yields the particular distance function for our case of pdfs estimated from data windows,

$$d\left(p_{\bar{t}}(\mathbf{x}), p_t(\mathbf{x})\right) = \frac{1}{W^2\left(4\pi\sigma^2\right)^{d/2}} \sum_{w,v=0}^{W-1} \left[\exp\left(-\frac{(\vec{x}_{\bar{t}-w} - \vec{x}_{\bar{t}-v})^2}{4\sigma^2}\right) - 2\exp\left(-\frac{(\vec{x}_{\bar{t}-w} - \vec{x}_{t-v})^2}{4\sigma^2}\right) + \exp\left(-\frac{(\vec{x}_{t-w} - \vec{x}_{t-v})^2}{4\sigma^2}\right)\right] \tag{5}$$

The segmentation algorithm

Before we introduce the on-line algorithm, it is necessary to discuss the respective off-line algorithm first.

– **The off-line algorithm.**
For a given a data sequence, $\{\vec{x}_t\}_{t=1}^T$, we can obtain the corresponding sequence of pdfs $\{p_t(\mathbf{x})\}_{t\in\mathcal{S}}$, $\mathcal{S} = \{W, ..., T\}$, according to eq. (2). The idea behind unsupervised segmentation is to find a compact (or compressed) representation of the sequence of pdfs in terms of a small set of pdfs, $Q = \{q_i(\mathbf{x})\}_{i=1}^N$, called *prototypes*, and a sequence $\mathbf{s} = \{s(t)\}_{t\in\mathcal{S}}$, $s(t) \in \{1, ..., N\}$, called *segmentation*, that assigns a prototype pdf to each pdf in the given sequence in such a way that $\mathbf{s}$ exhibits only a few number of prototype changes, $n(\mathbf{s}) = \sum_{t=W}^{T-1}(1 - \delta_{s(t),s(t+1)})$.[5] Moreover, such a sequence should have a small residual compression error,

$$\varepsilon(\mathbf{s}, Q) = \sum_{t=W}^{T} d(q_{s(t)}(\mathbf{x}), p_t(\mathbf{x})), \tag{6}$$

which is the sum of distances between given pdfs and corresponding prototype pdfs according to eq. (5). We can now formulate the overall segmentation problem in terms of an *objective* or *cost function* to be minimized with respect to $\mathbf{s}$ and Q as follows:

$$O(\mathbf{s}, Q) = \varepsilon(\mathbf{s}, Q) \; + \; C_1 \; n(\mathbf{s}) \; + \; C_2 \; N, \tag{7}$$

where C_1 and C_2 are weighting factors that act as regularization constants.

In fact, eq. (7) constitutes a complex optimization problem. However, if we simplify it and constrain the set of prototype pdfs to be a subset of

[5] In this definition, we use the usual Kronecker delta function $\delta_{i,j}$, i.e. $\delta_{i,j} = 1$, if $i = j$, and $\delta_{i,j} = 0$ otherwise.

the pdfs obtained from the data, $q_i(\mathbf{x}) \in \{p_t(\mathbf{x})\}_{t\in\mathcal{S}}$, and furthermore just penalize the number of prototype changes, $n(\mathbf{s})$, but not explicitly the total number of prototypes used, N, we can efficiently *compute* a global optimum of the simplified cost function by *dynamic programming* [1]. This does not mean that we assume a fixed number of prototypes. However, we restrict the number of resulting prototypes only indirectly by penalizing the number of segments, which is given by $n(\mathbf{s})+1$. In other words, using the same prototype for more than one segment is not rewarded. The problem of minimizing the total number of prototypes might be approached in a subsequent step though (see labeling section below).

The cost function to be minimized in the simplified case is given by

$$o(\mathbf{s}) = \sum_{t=W}^{T} d(p_{s(t)}(\mathbf{x}), p_t(\mathbf{x})) \; + \; C \; n(\mathbf{s}) \tag{8}$$

where now $\mathbf{s} = \{s(t)\}_{t\in\mathcal{S}}$, $s(t) \in \mathcal{S}$, denotes a sequence of pdfs from the given set $\{p_t(\mathbf{x})\}_{t\in\mathcal{S}}$. Finding the optimal segmentation $\mathbf{s}$ with respect to $o(\mathbf{s})$ corresponds to finding an optimal path in the matrix of pairwise distances between each of the given pdfs, $D = (d_{\bar{t},t})_{\bar{t},t\in\mathcal{S}}$, with

$$d_{\bar{t},t} = d(p_{\bar{t}}(\mathbf{x}), p_t(\mathbf{x})), \quad \bar{t}, t \in \mathcal{S}, \tag{9}$$

where the path runs from left to right along the time index t selecting a pdf $\bar{t} = s(t)$ as a prototype at each time step. The index $\bar{t} \in \mathcal{S}$ can therefore be understood as a *state* the system is in. We will denote $\bar{t}$ with s from now on in order to make a distinction between time and state. Since there is only a finite number of possible paths in the matrix, one might in principle obtain an optimal path $\mathbf{s}^*$ by computing $o(\mathbf{s})$ for *all* $(T - W + 1)^{(T-W+1)}$ possible paths $\mathbf{s}$ and choosing a path $\mathbf{s}^*$ for which $o(\mathbf{s})$ has a global minimum $o^* = o(\mathbf{s}^*) = \min_{\mathbf{s}}\{o(\mathbf{s})\}$.[6]

Fortunately, such an optimal sequence can be obtained much more efficiently in a single sweep from $t = W$ to T, since the task constitutes a typical dynamic programming problem [1]. At each time step t, we compute for all $s \in \mathcal{S}$ the cost $c_s(t)$ of the optimal path from W to t, subject to the constraint that it ends in state s at time t. We will call these constrained optimal paths c-paths and the unconstrained optimum o^*-path. The iteration can now be formulated as follows:

Initialization, $\forall s \in \mathcal{S}$:

$$c_s(W) := d_{s,W}, \tag{10}$$

Induction, $\forall s \in \mathcal{S}$:

$$c_s(t) := d_{s,t} + \min_{\bar{s}\in\mathcal{S}} \Big\{ \; c_{\bar{s}}(t-1) + C \; (1 - \delta_{s,\bar{s}}) \; \Big\}, \quad \text{for } t = W+1, ..., T, \tag{11}$$

[6]Note that it is generally very unlikely to find more than one global minimum, except in cases where one explicitly designs a problem in this respect.

Termination:

$$o^* := \min_{s \in \mathcal{S}} \left\{ \; c_s(T) \; \right\}, \tag{12}$$

where $\delta_{s,\bar{s}}$ again denotes the Kronecker delta function. The regularization constant C can thus be interpreted as transition cost for switching from one pdf to another in the path. The optimal prototype sequence with minimal cost o^* for the complete series of pdfs, which is determined in the last step, is obtained by logging and updating the c-paths for all states s during the iteration and finally choosing the one with minimal costs according to eq. (12).

– The on-line algorithm.

The first step to make the above segmentation algorithm on-line is to allow for incremental updates when new data points arrive. We neglect at this first stage that memory and CPU resources might be limited.

Suppose that we have already processed data up to $T-1$. When a new data point $\vec{y}_T$ comes in at time T, we can generate a new embedded vector $\vec{x}_T$ (once we have sampled enough initial data points for the embedding) and we have a new pdf $p_T(\mathbf{x})$ (once we have sampled enough embedded vectors $\vec{x}_t$ for the first pdf window). Next, we can easily obtain a new pdf distance matrix $D_T = (d_{\bar{t},t})_{\bar{t},t \in \mathcal{S}}$ from the previous one, D_{T-1}, simply by computing the new row and column vector for time T. Since our distance function d is symmetric, both vectors are identical and we are left with computing one vector of distances between the new pdf and each of the previous pdfs according to eq. (5).

Unlike the distance matrix, the segmentation can not readily be updated incrementally. The newly added pdf might affect all the optimal c-paths with minimal costs $c_s(T-1)$ obtained so far, since they do not take the new pdf as a potential state into account. Updating the c-paths would therefore in principle require a re-computation from scratch, for which we could simply use the off-line algorithm again. We can perform a simplified update, however, if we reconsider the cost function. We simplified the original cost function in eq. (7) in order to make the problem efficiently solvable by dynamic programming. The restriction in the simplified cost function in eq. (8) not to penalize the number of prototypes and to choose prototypes only from pdfs obtained from the data results in optimal segmentations where each segment typically has a different pdf prototype taken from within the respective segment. This is because such a prototype optimally explains its own pdf in the segment, i.e. the distance is zero, and it also has a small distance to neighboring pdfs, since it is partially constructed from the same data. Hence there is an inherent bias in the cost function towards choosing prototypes from pdfs within the segment, which therefore generally results in a temporally ascending order of the prototype pdfs. Vice versa, a path that exhibits a switch back to a previous pdf is unlikely to be optimal with respect to the cost function. Thus, if we neglect to consider c-paths that contain a switch back, we still obtain

the optimal o^*-path in most cases.[7] We therefore simply reuse the c-paths from $T-1$ even though a new pdf state is given, since using that pdf as a prototype in past segments would require a switch back in the respective path and thus is unlikely to be optimal. The on-line update at time T for these restricted paths, that we henceforth denote with a tilde, can be performed as follows: For $T = W$, D is a 1×1 matrix and $\tilde{c}_W(W) := \tilde{o}^*(W) := d_{W,W} = 0$. For $T > W$:

1. Compute the cost $\tilde{c}_T(T-1)$ for the new state $s = T$ at time $T-1$: For $t = W, ..., T-1$, compute

$$\tilde{c}_T(t) := d_{T,t} + \begin{cases} 0 & , \quad \text{if} \quad t = W \\ \min\left\{\tilde{c}_T(t-1);\ \tilde{o}^*(t-1) + C\right\}, & \text{else} \end{cases} \tag{13}$$

   and update

$$\tilde{o}^*(t) := \tilde{c}_T(t), \quad \text{if} \quad \tilde{c}_T(t) < \tilde{o}^*(t). \tag{14}$$

   Here we use all previous optimal segmentations $\tilde{o}^*(t)$, so we don't need to keep the complete matrix $\{\tilde{c}_s(t)\}$ and repeatedly compute the minimum over all states. However, we must store and update the history of optimal segmentations $\tilde{o}^*(t)$.

2. Update from $T-1$ to T and compute $\tilde{c}_s(T)$ for all states $s \in \mathcal{S}$, and also $\tilde{o}^*(T)$: For $s = W, ..., T$, compute

$$\tilde{c}_s(T) := d_{s,T} + \min\left\{\tilde{c}_s(T-1);\ \tilde{o}^*(T-1) + C\right\} \tag{15}$$

   and then get the optimal path

$$\tilde{o}^*(T) := \min_{s \in \mathcal{S}} \left\{\tilde{c}_s(T)\right\}. \tag{16}$$

   Note that if the min-operation in eq. (15) results in a switch from the last state in the optimal $\tilde{o}^*(T-1)$-path to the current state s, it might be a switch back from a succeeding pdf. We will make use of this below to limit memory requirements and computation time.

As for the off-line case, the above algorithm only shows the update equations for the *costs* of the $\tilde{c}$- and $\tilde{o}^*$-paths. The associated path sequences must be logged simultaneously during the computation. Note that this can be done by just storing the sequence of switching points for each path.

So far we have presented the incremental update version of the segmentation algorithm, which only needs to traverse the newly added row (1. step) and column (2. step) of the distance matrix D. It does not consider some paths with prototypes in non-ascending order, it does, however, neither exclude all of them due to the unconstrained min-operation in eq. (15). This algorithm still needs an amount of memory and CPU time that is increasing with each new data point. Note, however, that we do not need to keep the full matrices D and $\{\tilde{c}_s(t)\}$ for the update, the most recent columns are sufficient.

[7] In fact, one rarely finds a deviation with respect to the resulting o^*-path in practice.

In order to limit both resources to a fixed amount, we must remove old pdfs at some point. We propose to do this by discarding all pdfs with time indices smaller or equal to s each time a path associated with $\tilde{c}_s(T)$ in eq. (15) exhibits a switch back. In the above algorithm this can simply be done by setting $W := s + 1$ in that case, which also allows us to discard the corresponding old $\tilde{c}$- and $\tilde{o}^*$-paths. In addition, the "if $t = W$" initialization clause in eq. (13) must be ignored after the first such cut and the $\tilde{o}^*(W-1)$-path must therefore still be kept to compute the else-part also for $t = W$ now. Also, we do not have $\tilde{c}_T(W-1)$ and we therefore assume $\min\{\tilde{c}_T(W-1);\ \tilde{o}^*(W-1) + C\} = \tilde{o}^*(W-1) + C$ (in eq. (13)).

A switch back in eq. (15) indicates that a new mode is established, such that the $\tilde{c}$-path that ends in a pdf state s from an old distribution starts to route its path through one of the more recent states that represent the new distribution, which in that case has lower costs despite of the incurred additional transition. Vice versa, a newly obtained pdf is unlikely to properly represent the previous mode then, which justifies our above assumption about $\tilde{c}_T(W-1)$. The effect of the proposed cut-off strategy is that we discard paths that end in pdfs from old modes but still allow to find the optimal pdf prototype within the current segment.

Cut-off conditions occur shortly after mode changes in the data. If no mode change takes place, however, no pdfs will be discarded. We therefore still need to set a fixed upper limit κ for the number of candidate paths/pdfs that are simultaneously under consideration if we only have limited resources available. When this limit is reached because no switches are detected, we must successively discard the oldest path/pdf stored, which finally might result in choosing a sub-optimal prototype for that segment, however. Moreover, continuous discarding even *enforces* a change of prototypes after 2κ time steps if no switching is induced by the data until then. The buffer size κ should therefore be as large as possible. In any case, the buffer overflow condition can be recorded along with the segmentation, which allows us to identify such artificial switchings.

Labeling the prototypes

We need a labeling algorithm on top of the segmentation algorithm to identify segments that stem from the same underlying distribution and thus have similar pdf prototypes. The labeling algorithm generates labels for the segments and assigns identical labels to segments that are similar in this respect. We do this by on-line clustering of the prototypes. To this end, we use a simple clustering scheme since we expect the prototypes obtained from the same underlying distribution to be already well-separated from the other prototypes as a result of the segmentation algorithm. We assign a new label to a segment if the distance of its associated prototype to all preceding prototypes exceeds a certain threshold θ, and we assign the existing label of the closest preceding prototype otherwise.

We developed algorithms that compute values for θ and the other hyper-parameters, C, σ, and κ, from a sample set $\{\vec{x}_t\}$. Due to the limited space, we will present them in a forthcoming publication.

APPLICATION

To illustrate our approach, we generated a time series from switching dynamics using the Mackey-Glass delay differential equation,

$$\frac{dx(t)}{dt} = -0.1x(t) + \frac{0.2x(t-t_d)}{1+x(t-t_d)^{10}}. \tag{17}$$

It describes a high-dimensional chaotic system that was originally introduced as a model of blood cell regulation [8]. In our example, three stationary operating modes, A, B and C, are established by using different delays, $t_d =$ 17, 23, and 30, respectively. The dynamics operates stationary in one mode for a certain number of time steps, which is chosen at random between 100 and 300 time steps (referring to sub-sampled data with a step size $\Delta = 6$). It then switches to one of the other modes with probability 0.5 for choosing either mode. This procedure is repeated 20 times, thus generating a switching chaotic time series with 20 stationary segments. We then added a relatively large amount of "measurement" noise to the series: zero-mean Gaussian noise with a standard deviation of 1/4 of the standard deviation of the original series.

Next, we applied the on-line segmentation algorithm to the data, i.e. processing was performed on-line although the full data set was already available in this case. We embedded the scalar time series on-the-fly by using $m = 6$ and $\tau = 1$ (on the sub-sampled data) and used a pdf window of size $W = 50$. The algorithm processed 253 data points per second on a 750-MHz Pentium-III in MATLAB, including display of the ongoing segmentation. The final segmentation is depicted in Fig. 1. Surprisingly, the bounds of the segments are almost perfectly recovered from the very noisy data set. The only two exceptions are the fourth segment from the right, which is noticeably shorter than the original mode, and the segment labeled with a 5, which is a bit too long. The on-line labeling algorithm, however, assigns six labels instead of three. In particular, the segments from the most chaotic mode, C, get three different labels, 2, 5, and 6. This can be explained by the fact that the segments comprise only 100–300 very noisy data points, such that the data distributions of such small sample sets are likely to differ to some extent. The next complex mode, B, received two labels, 1 and 4, and mode A only one, label 3.

DISCUSSION

An on-line method for the unsupervised segmentation and identification of time series from non-stationary switching dynamics was presented. In contrast to other approaches, it processes the data on-line and potentially even

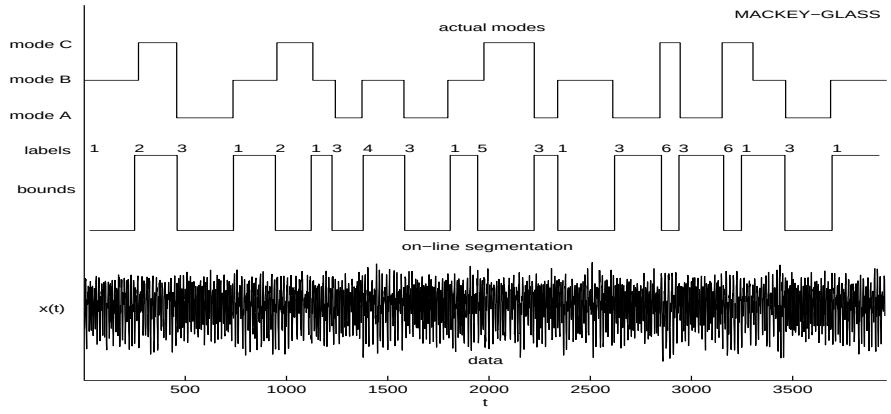

Figure 1: Segmentation of a switching Mackey-Glass time series with noise (bottom) that operates in three different modes (top). The resulting on-line segmentation (middle), which receives the data points one by one but not the mode information, exhibits almost perfect segmentation bounds. The on-line labeling algorithm, however, assigns more labels (6) than desired (3), presumably due to the fact that the segments are very short and noisy.

in real-time without training of any parameters. The method provides and updates a segmentation each time a new data point arrives. In effect, past segmentation bounds and labels are automatically re-adapted when new incoming data points are processed. The number of prototypes and labels that identify the segments is not fixed but determined by the algorithm and in most cases the segmentation is exactly the same as the respective optimal off-line segmentation. We expect useful applications of this method in fields where complex non-stationary dynamics plays an important role, like, e.g., in physiology (EEG, MEG), climatology, in industrial applications, or in finance.

REFERENCES

[1] Bellman, R. E. (1957). *Dynamic Programming*, Princeton University Press, Princeton, NJ.

[2] Bengio, Y., Frasconi, P. (1995). An Input Output HMM Architecture. In: *NIPS'94: Advances in Neural Information Processing Systems* 7 (eds. G. Tesauro, D.S. Touretzky, T.K. Leen), Morgan Kaufmann, 427–434.

[3] Bishop, C. M. (1995). *Neural Networks for Pattern Recognition*, Oxford Univ. Press, NY.

[4] Husmeier, D. (2000). Learning Non-Stationary Conditional Probability Distributions. *Neural Networks* 13, 287–290.

[5] Kehagias, A., Petridis, V. (1997). Time Series Segmentation using Predictive Modular Neural Networks. *Neural Computation* 9, 1691–1710.

[6] Kohlmorgen, J., Müller, K.-R., Rittweger, J., Pawelzik, K. (2000). Identification of Nonstationary Dynamics in Physiological Recordings, *Biol Cybern* 83(1), 73–84.

[7] Liehr, S., Pawelzik, K., Kohlmorgen, J., Müller, K.-R. (1999). Hidden Markov Mixtures of Experts with an Application to EEG Recordings from Sleep. *Theo Biosci* 118, 246–260.

[8] Mackey, M., Glass, L. (1977). Oscillation and Chaos in a Physiological Control System. *Science* 197, 287.

[9] Packard, N. H., Crutchfield J. P., Farmer, J. D., Shaw, R. S. (1980). Geometry from a Time Series. *Phys. Rev. Letters* **45**, 712–716.

[10] Pawelzik, K., Kohlmorgen, J., Müller, K.-R. (1996). Annealed Competition of Experts for a Segmentation and Classification of Switching Dynamics. *Neural Comput* 8(2), 340–356.

[11] Ramamurti, V., Ghosh, J. (1999). Structurally Adaptive Modular Networks for Non-Stationary Environments. *IEEE Tr. Neural Networks* 10(1), 152–160.