ON OPTIMAL SEGMENTATION OF SEQUENTIAL DATA

Jens Kohlmorgen
Fraunhofer FIRST.IDA
Kekuléstr. 7, D-12489 Berlin, Germany
E-mail: jek@first.fraunhofer.de

Abstract. We present an algorithm that efficiently computes optimal partitions of sequential data into 1 to N segments and propose a method to determine the most salient segmentation among them. As a by-product, we obtain a regularization parameter that can be used to compute such salient segmentations – also on new data sets – even more efficiently.

INTRODUCTION

Consider the problem of assigning each data point x_t of a sequence of data points $[x_1, ..., x_T]$ to a class s_t of a given set $\Psi = \{1, ..., S\}$ such that the resulting sequence of class labels $[s_1, ..., s_T]$ has minimal costs with respect to the cost function

$$c([s_1, ..., s_T]) = \sum_{t=1}^{T} D[s_t, t], \tag{1}$$

where D is a given $S \times T$ matrix with elements $D[s, t]$ – or, equally well, a given function of s and t – that assigns individual costs (or distances) for each class s to each data point by way of the data point index t. The resulting sequence should furthermore consist of exactly n segments, i.e. the number of *class label changes* in the sequence should be exactly $n - 1$.

This problem is closely related to the well-known Viterbi algorithm for hidden Markov models (HMMs) [5]. The Viterbi algorithm finds the optimal state (class) sequence in an HMM that best represents a sequence of observations $[x_1, ..., x_T]$. The difference to the problem considered in this work is that there is no constraint in the Viterbi algorithm with respect to the exact number of visited states, which is a *global* property of a sequence. Instead, state changes are controlled by transition probabilities (or, equivalently, transition costs) and a *local* decision that is independent of the number of state changes already performed before or to be performed after the transition. Such a local Markovian decision process constitutes a typical dynamic programming problem [1], which in this case is solved efficiently by the Viterbi algorithm.

In contrast to the Viterbi problem, the task that we consider here exhibits a global constraint, which at first sight does not allow for an incremental dynamic programming approach. We will show, however, that the problem can be transformed into a local decision problem. In addition, we consider the more general case where optimal state sequences are to be returned for all n in a given set $\{1, ..., N\}$. This allows us to quantify the robustness of each of the N obtained segmentations in a subsequent step, which in turn gives an indication which of the segmentations is the most salient one. Such an evaluation is helpful in unsupervised tasks where the significance of a segmentation can not be assessed otherwise, in particular if there is no target segmentation available. Moreover, the method presented here provides a segmentation mechanism that does not depend on any hyperparameter[1], in contrast to the Viterbi segmentation that depends – potentially to a large extent – on the transition probability matrix that must be chosen in advance. On the contrary: we show that the method can actually be used to *determine* a hyperparameter such that a simpler, Viterbi-like algorithm produces the same segmentation.

THE N–SEGMENTATIONS ALGORITHM

Let D be an $S \times T$ matrix with elements $D[s,t]$, where $s \in \Psi = \{1, ..., S\}$ and $t \in \{1, ..., T\}$. Without loss of generality, we call s the state index and t the time index of the matrix, and $D[s,t]$ the cost of state s at time t. We consider the problem of finding $n = 1, ..., N$ shortest paths $p_n(T) = [s_1^n, ..., s_T^n]$ through the matrix D, with $s_t^n \in \Psi$, such that (a) the n-th path consists of exactly n segments, i.e. the number of state changes in the path is $n-1$,

$$\#\{\ s_t^n \mid s_t^n \neq s_{t-1}^n\} \ = \ n-1, \tag{2}$$

and (b) the sum of costs (the length of the path),

$$c([s_1, ..., s_T]) \ = \ \sum_{t=1}^{T} D[s_t, t], \tag{3}$$

is minimal for each path. This combinatorial optimization problem can be summarized as follows:

$$p_n(T) \ = \ \underset{[s_1, ..., s_T]}{\operatorname{argmin}} \left\{ \ c([s_1, ..., s_T]) \ \right\} \tag{4}$$

s.t.

$$\#\{\ s_t \mid s_t \neq s_{t-1}\} \ = \ n-1 \tag{5}$$

[1]if we neglect N, which merely denotes the number of segmentations to be computed.

In principle, the problem can be solved in exponential time by evaluating all possible S^T paths through the matrix, for all n. This calculation is computationally unfeasible however, even for small values of S and T, e.g. for $S = 10$ and $T = 100$ one would need to evaluate 10^{100} paths. We here present an algorithm that can be implemented with only $O(TNS)$ time and space complexity. To achieve this, we make use of the following property of a shortest path with n segments.

Theorem 1 (shortest-path property).
Let $p_n(t) = [s_1, ..., s_t]$, $t > 1$, be the shortest path with n segments that ends at time t. Then, if $s_{t-1} = s_t$,

> *$[s_1, ..., s_{t-1}]$ is the shortest path with n segments that ends in state s_{t-1} at time $t - 1$,*

otherwise, i.e. if $s_{t-1} \neq s_t$,

> *$[s_1, ..., s_{t-1}]$ is the shortest path with $n - 1$ segments that ends in state s_{t-1} at time $t - 1$.*

Proof. First case ($s_{t-1} = s_t$):

– Because $s_{t-1} = s_t$, the path $[s_1, ..., s_{t-1}]$ has the same number of segments as $[s_1, ..., s_t]$ (cf. Eq. (5)).

– Assume there exists a path $[r_1, ..., r_{t-1}]$ with n segments that ends in the state s_{t-1} at time $t - 1$, i.e. $r_{t-1} = s_{t-1}$, and that is shorter than $[s_1, ..., s_{t-1}]$. Then, $[r_1, ..., r_{t-1}, s_t]$ would be a path with n segments at time t that is shorter than $[s_1, ..., s_{t-1}, s_t]$. This would contradict the premise of theorem 1, therefore $[s_1, ..., s_{t-1}]$ must be the shortest path.

Second case ($s_{t-1} \neq s_t$):

– Because $s_{t-1} \neq s_t$, the path $[s_1, ..., s_{t-1}]$ has one segment less than $[s_1, ..., s_t]$ (cf. Eq. (5)).

– Assume there exists a path $[r_1, ..., r_{t-1}]$ with $n - 1$ segments that ends in state s_{t-1} at time $t - 1$, i.e. $r_{t-1} = s_{t-1}$, and that is shorter than $[s_1, ..., s_{t-1}]$. Then, $[r_1, ..., r_{t-1}, s_t]$ would be a path with n segments at time t that is shorter than $[s_1, ..., s_{t-1}, s_t]$. Again, this would contradict the premise of theorem 1 and therefore $[s_1, ..., s_{t-1}]$ must be the shortest path.

$\square$

Theorem 1 tells us that the shortest path $p_n(t)$ with n segments always consists of a sub-path $[s_1, ..., s_{t-1}]$ that again is a shortest path, however, only with respect to all paths that also end in the same state s_{t-1} at time $t - 1$. A way to construct the shortest path is therefore to track all shortest paths constrained to particular ending states $s = 1, ..., S$ at times t, denoted by $p_n^s(t)$. We can then obtain the shortest unconstrained path $p_n(t)$ simply by taking the minimum over all possible states,

$$p_n(t) \;=\; \operatorname*{argmin}_{s\in\Psi} \left\{ \; c(p_n^s(t)) \; \right\} . \tag{6}$$

Theorem 2 (recursion property).
The shortest-path property in theorem 1 also holds for a path $p_n^s(t)$ that is the shortest path only with respect to all paths that exhibit the same ending state s at time t.

Proof. Analogous to the proof of theorem 1. □

Theorem 2 readily leads us to the following shortest-path algorithm. The algorithm recursively computes the cost $c_n^s(t)$ for each constrained shortest path $p_n^s(t)$, for all $t = 1, ..., T$ and all n and s, and finally obtains the costs $c_n(T)$ of all unconstrained shortest paths $p_n(T)$.

Optimal N–Segmentations Algorithm

1. Initialization ($t = 1$):

$$\forall n, s\colon \;\; c_n^s(1) \;:=\; \begin{cases} D[s,1]\,; & \text{if} \quad n = 1 \\ \quad \infty \quad ; & \text{if} \quad 2 \le n \le N \end{cases} \tag{7}$$

Paths with only one element ($t = 1$) can contain only one segment and have costs $c_1^s(1) = D[s, 1]$ (Eq. (3)). The costs of the (unrealizable) paths with more than one segment are set to ∞, which is consistent with the following recursion.

2. Recursion (for $t = 2, ..., T$):

$\forall n, s\colon$

$$c_n^s(t) \;:=\; \begin{cases} \qquad D[s,t] + c_n^s(t-1) & ; \;\text{if} \quad n = 1 \\ D[s,t] + \min \Big\{ \; c_n^s(t-1), & \\ \qquad\qquad \min\limits_{\bar{s}\neq s}\{c_{n-1}^{\bar{s}}(t-1)\} \; \Big\} & ; \;\text{if} \quad 2 \le n \le N \end{cases} \tag{8}$$

Eq. (8) determines the shortest of all paths that might constitute $p_n^s(t)$ up to time $t - 1$ (according to theorem 2) and adds the cost $D[s, t]$ of the new ending state s.

3. Termination:

$$\forall n\colon \;\; c_n(T) \;:=\; \min_{s\in\Psi} \left\{ \; c_n^s(T) \; \right\} \tag{9}$$

Eq. (9) finally determines the (costs of the) shortest unconstrained paths $p_n(T)$ through the matrix D. The actual state sequences $p_n(T)$ must be obtained by backtracking through the sequence of states that make up each $c_n(T)$.

The above algorithm has $O(TNS^2)$ time complexity due to the $\min_{\bar{s}\neq s}$-operation that runs over all states except the current state s for $(T–1)(N–1)S$ evaluations of Eq. (8). We can, however, easily convert it into an $O(TNS)$ algorithm, if we instead use an unconstrained min-operation once for all s, for a given n and t, and verify later if the condition $\bar{s} \neq s$ applies. This is how we actually implemented the algorithm. Finally note that the presented algorithm should be applied for $S \geq 2$ and $N \leq T$. Otherwise it returns $c_n(T) = \infty$ for all paths that are unrealizable.

CHOOSING THE BEST SEGMENTATION

After having obtained N optimal segmentations with 1 to N segments, the question is how to decide which one of them is the most appropriate segmentation of the underlying data sequence. If one can not make an assessment by using additional external knowledge, it will in general not just be the one with the smallest cost, $\min_n\{c_n(T)\}$, because typically, as e.g. in [2], the costs will simply monotonically decrease when n – the number of segments – is increased, since a larger n provides more flexibility for modeling the data. This criterion would then always favor the segmentation with the largest number of segments, regardless of the sequential structure of the data. One therefore needs to take the increasing model complexity into account. A common way is to add a regularization term to the cost function c,

$$o(\mathbf{s}) \;=\; c(\mathbf{s}) \;+\; C\, n(\mathbf{s}) \tag{10}$$

which in this case explicitly penalizes the number of segments $n(\mathbf{s})$ in a sequence, $\mathbf{s} = [s_1, ..., s_T]$, weighted by a regularization parameter $C > 0$. The point is then to choose a suitable regularization parameter. If we would start with $C = 0$ and then successively increase C, one can easily see that the number of segments in the segmentation that minimizes o would stepwise increase until there is only one segment left. The interesting point is: how stable are the intermediate segmentations, i.e. how large is each interval in which C yields the same particular segmentation? In the following we show that these intervals and the corresponding optimal segmentations can be computed via the N–segmentations algorithm.

Theorem 3. *For any C, there is an $n \in \{1, ..., T\}$ such that the sequence $p_n(T)$ minimizes the regularized cost function o.*

Proof. For any given C, there is a sequence (at least one) that minimizes the cost function o. Let us denote such a sequence as $\mathbf{s}_C$ and its number of segments as n_C. The number n_C necessarily must be an integer between 1 and T. Then the sequence $p_{n_C}(T)$ also minimizes o for the given C, since it has the same number of segments as $\mathbf{s}_C$, and therewith the same regularization term, and it has minimal costs c with respect to all sequences with the same number of segments as $p_{n_C}(T)$.[2] □

[2]Typically, there is only one global minimum, and thus $p_{n_C}(T) = \mathbf{s}_C$.

According to theorem 3, it is sufficient to consider all segmentations $p_n(T)$, $n \in \{1, ..., T\}$, of the N–segmentations algorithm – with $N = T$ in this case – in order to find the minimum of the regularized cost function o, for any C. The optimization problem can therefore be reduced to finding the optimal n for a given C:

$$o^* \ = \ \min_{1 \leq n \leq N} \left\{ \ c_n(T) \ + \ C\,n \ \right\} \tag{11}$$

If required, one can easily constrain the optimization problem in Eq. (11) to segmentations with not more than a certain number of segments, simply by setting N to that number, instead of using $N = T$. This is actually the situation where we started from. Also if no such restriction is required, one might consider to set N to an expected maximum number of segments ($N \ll T$), since this largely reduces memory requirements and computation time of the N–segmentations algorithm. One should be aware, however, that all segmentations with more than N segments are ignored in that case, even if they would have yielded lower costs.

The minimization problem in Eq. (11) consists of a set of linear functions of C, one for each $n \in \{1, ..., N\}$, where n constitutes the slope of the functions. Figures 1 and 2 illustrate this.

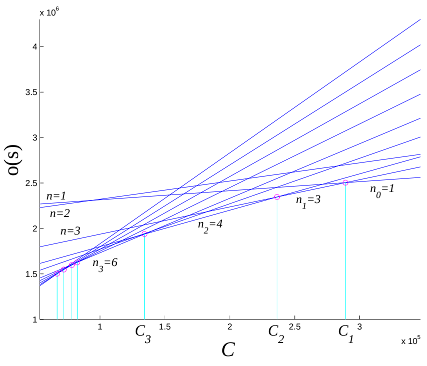

Figure 1: An example of a set of linear functions that constitute a minimization problem according to Eq. (11). The task is to determine the line segments at the bottom.

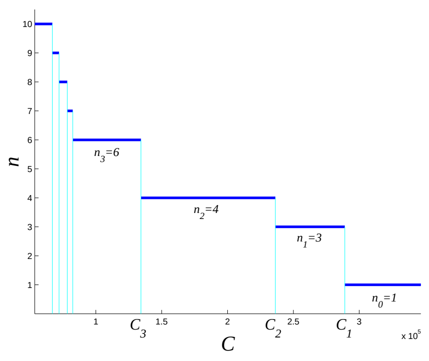

Figure 2: The intervals for the parameter C, in which the regularized minimization task in Eq. (11) would yield a segmentation with the depicted number of segments, n.

An efficient way to determine the line segments that constitute the lower border of the set is to traverse the lines from smallest to largest slope, starting with $n = 1$, and to update the solution incrementally. At each update step, the line with the next largest slope is added and previously obtained line segments are removed, if they are completely above the newly added line (see, e.g., line $n = 2$ in Fig. 1). The lines with the smallest and (currently) largest slope, however, always constitute the outermost segments. The exclusion of a line with a particular n means that the regularized minimization problem (Eq. (10) or (11)) would never yield a segmentation with that particular number of segments, no matter how large C is, because it is never optimal.

The following algorithm computes the sequence of lower intersections, more precisely: the series of C-values, C_k, and the corresponding slopes, n_k.

Lower Intersections Algorithm

```
(* k: number of intersections *)
$k := 0$
$n_0 := 1$
for $n = 2, ..., N$
    $k := k + 1$
    $C_k := \frac{c_{n_{k-1}}(T) - c_n(T)}{n - n_{k-1}}$
    while $k > 1$ and $C_k > C_{k-1}$
        $k := k - 1$
        $C_k := \frac{c_{n_{k-1}}(T) - c_n(T)}{n - n_{k-1}}$
    end
    $n_k := n$
end
```

The algorithm runs in $O(N)$ time: the inner part of the while–loop is executed at most $N - 2$ times *in total*, since this is the maximum number of lines that can possibly be excluded. All computed intersection points C_k are positive, if (and only if) the segmentation with the largest number of segments has the smallest unregularized cost, i.e. if $c_N(T) = \min_{n \leq N}\{c_n(T)\}$. Otherwise at least the (smallest) C_k with the largest index is negative. In that case, all negative C_k-values must henceforth be ignored, since we want to *penalize* the number of segments and not reward it. We then propose to use the distance between two successive and positive C_k-values to quantify how salient and stable a segmentation is. The k-index of the segmentation with the largest C-interval is given by

$$k^* \;=\; \underset{1 \leq k < K}{\operatorname{argmax}} \left\{ \, C_k \;-\; C_{k+1} \, \right\} \tag{12}$$

where K denotes the k-index of the smallest positive C_k. The number of segments of this most robust segmentation is then given by n_{k^*} and the segmentation itself is given by $p_{n_{k^*}}(T)$. Furthermore, we can determine a robust regularization parameter C^* that would yield this segmentation with maximum tolerance by taking the mean of the two corresponding boundary values,

$$C^* \;=\; \frac{1}{2}\,(C_{k^*} \;+\; C_{k^*+1}). \tag{13}$$

The computation of C^* is very useful if one wants to compute segmentations – at the same level of granularity – on related data sets of potentially different size, since once we have obtained a suitable value for the hyperparameter C, we can compute a segmentation that minimizes $o(\mathbf{s})$ in Eq. (10) – for that particular C – very efficiently *without* using the N–segmentations algorithm.

This can be accomplished by a standard dynamic programming approach, namely by recursively computing the quantity $o^s(t)$, which denotes the cost of the sequence that minimizes Eq. (10) with respect to all sequences that end in state s at time t. In the following, we present a formulation of the algorithm that only has $O(TS)$ time (and space) complexity:

Optimal C–Segmentation Algorithm

1. Initialization ($t = 1$):

$$\forall s \in \Psi: \;\; o^s(1) \;:=\; D[s, 1] \tag{14}$$

2. Recursion (for $t = 2, ..., T$):

$$h \;:=\; \min_{s \in \Psi} \left\{ \; o^s(t-1) \; \right\} + C \tag{15}$$

$$\forall s \in \Psi: \;\; o^s(t) \;:=\; D[s, t] + \min \left\{ \; o^s(t-1), \; h \; \right\} \tag{16}$$

3. Termination:

$$o^* \;:=\; \min_{s \in \Psi} \left\{ \; o^s(T) \; \right\} \tag{17}$$

The actual state sequence that corresponds to the minimum o^* must be obtained by backtracking through the sequence of states that make up o^*. An on-line variant of this algorithm, which is able to process unlimited data streams on-the-fly, was recently proposed in [2, 3].

AN EXAMPLE

As a proof of concept, we apply the proposed method to a fundamental problem in signal processing and statistics: the analysis of non-stationary data. To this end, we employ a data set that was used in [3] for the purpose of on-line segmentation. The data consists of a noisy time series from a complex dynamical system with alternating modes of operation. It is based on the Mackey-Glass delay differential equation [4],

$$\frac{dx(t)}{dt} = -0.1x(t) + \frac{0.2x(t-t_d)}{1 + x(t-t_d)^{10}}. \tag{18}$$

Four stationary operating modes, A, B, C, and D, are established by using four different delays, $t_d = 17$, 23, 30, and 35, respectively. The dynamical system is run stationary in one mode for a certain number of time steps, which is chosen at random between $\Delta t = 200$ and 300 (referring to sub-sampled data with a step size $\delta = 6$). The system is then randomly switched to one of the other modes with uniform probability. In addition, a relatively large amount of “measurement” noise is added to the series: zero-mean Gaussian noise with a standard deviation of 30% of the standard deviation of the original signal. The task is then to detect the different operating modes in the very noisy data sequence.

To keep the example clear, we only use the first four generated segments of the time series. That sequence contains 875 data points from modes B, C, A, and B (in that order). As a first step of preprocessing, the data is embedded into a six-dimensional space ($d = 6$) by using time-delay coordinates with a delay $\tau = 1$ on the sub-sampled data,

$$\vec{x}_t = (x_t, x_{t-1}, \ldots, x_{t-5}). \tag{19}$$

The next step can be interpreted as feature extraction: a sequence of Parzen window density estimates is obtained by shifting a window of size $W = 50$ stepwise over the embedded data,

$$p_t(\mathbf{x}) = \frac{1}{W} \sum_{w=0}^{W-1} \frac{1}{(2\pi\sigma^2)^{d/2}} \, \exp\left(-\frac{(\mathbf{x} - \vec{x}_{t-w})^2}{2\sigma^2}\right). \tag{20}$$

Finally, a square matrix D of all pairwise distances between the obtained density estimates is computed. To this end, the *integrated squared error* (ISE) is used as distance measure (see [3] for further details),

$$D[s,t] \; = \; \int (p_s(\mathbf{x}) - p_t(\mathbf{x}))^2 \, d\mathbf{x}. \tag{21}$$

This matrix D is the input to our algorithm. The segmentation task can be interpreted as a search for n prototypical density estimates (the “states” s) that best represent all density estimates in their corresponding segment, i.e. in this case the sequence to be segmented is actually a sequence of density estimates and each density estimate in that sequence at the same time constitutes a possible class or state.

Ideally, one would expect that the algorithm finds out that a segmentation with four segments is the most salient one, and that the three obtained segment boundaries in that segmentation precisely match the true ones. Figure 3 shows the unregularized costs $c_n(T)$ of $N = 10$ segmentations computed by the N–segmentations algorithm. The costs are decreasing with increasing n. Moreover, one can not see any preference for $n = 4$, because the model complexity is not taken into account. Figure 2 shows which one of the computed segmentations $p_n(T)$ minimizes the *regularized* cost function o, in dependence of C (compare also with Fig. 1). Clearly, the most salient segmentation with the largest C-interval is the one with $n = 4$ segments. The respective robust regularization parameter C^* is in the middle of that interval: $C^* = 1.85 \cdot 10^5$.

Figure 4 shows the actual segmentations obtained by the N–segmentations algorithm. For $n = 4$, the differences between true and estimated segment boundaries amount to only about one wavelength of the signal. Considering the large amount of noise in the data, this is an excellent result. It can also be seen in Fig. 4 that a larger number of segments ($n > 4$) causes a finer segmentation into sub-modes, whereas a smaller number of segments leads to a pooling of neighboring modes and therewith to a coarser representation.

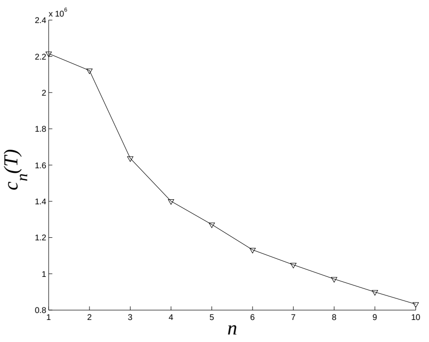

Figure 3: The costs $c_n(T)$ of the optimal segmentations with $n = 1, ..., 10$ segments, obtained by the N–segmentations algorithm for the example data set.

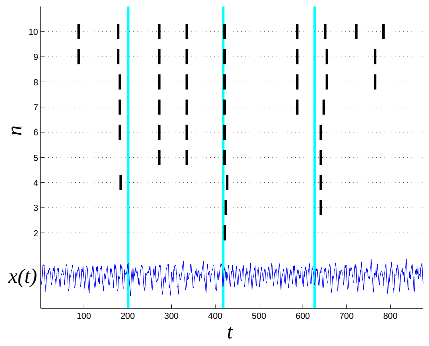

Figure 4: The boundaries of the optimal segmentations $p_n(T)$, for $n = 2, ..., 10$ segments, computed by the N–segmentations algorithm for the example data set $x(t)$. (True boundaries in grey.)

DISCUSSION

We presented an algorithm that efficiently computes optimal segmentations of sequential data into 1 to N segments. It can be implemented with only $O(TNS)$ time and space complexity, and, to the best of our knowledge, it has not been published so far, despite of the rather general nature of the problem it solves. We furthermore showed how the result can be used to determine the most salient segmentation among the obtained ones. As a by-product, a robust regularization parameter can be obtained to compute segmentations of new data sets – at the same level of granularity – even more efficiently. We expect useful applications of this method in fields where sequence segmentation plays an important role, like, e.g., in medicine (EEG, MEG), bioinformatics, text segmentation, or industrial applications.

REFERENCES

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

[2] Kohlmorgen, J., Lemm, S. (2001). An On-line Method for Segmentation and Identification of Non-stationary Time Series. In: *Neural Networks for Signal Processing XI*, IEEE, NJ, 113–122.

[3] Kohlmorgen, J., Lemm, S. (2002). A Dynamic HMM for On-line Segmentation of Sequential Data. In: *Advances in Neural Information Processing Systems 14*, (eds. T.G. Dietterich, S. Becker, and Z. Ghahramani), MIT Press.

[4] Mackey, M., Glass, L. (1977). Oscillation and Chaos in Physiological Control Systems. *Science* **197**, 287–289.

[5] Rabiner, L. R. (1989). A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition, *Proceedings of the IEEE* **77**(2), 257–286.