Neural circuits for dynamics-based segmentation of time series

Tiberiu Teşileanu[1], Siavash Golkar[1], Samaneh Nasiri[2], Anirvan M. Sengupta[1,4], and Dmitri B. Chklovskii[1,3]

[1]Center for Computational Neuroscience, Flatiron Institute
[2]Department of Neurology, Harvard Medical School
[3]Department of Physics and Astronomy, Rutgers University
[4]Neuroscience Institute, NYU Medical Center

Abstract

The brain must extract behaviorally relevant latent variables from the signals streamed by the sensory organs. Such latent variables are often encoded in the dynamics that generated the signal rather than in the specific realization of the waveform. Therefore, one problem faced by the brain is to segment time series based on underlying dynamics. We present two algorithms for performing this segmentation task that are biologically plausible, which we define as acting in a streaming setting and all learning rules being local. One algorithm is model-based and can be derived from an optimization problem involving a mixture of autoregressive processes. This algorithm relies on feedback in the form of a prediction error, and can also be used for forecasting future samples. In some brain regions, such as the retina, the feedback connections necessary to use the prediction error for learning are absent. For this case, we propose a second, model-free algorithm that uses a running estimate of the autocorrelation structure of the signal to perform the segmentation. We show that both algorithms do well when tasked with segmenting signals drawn from autoregressive models with piecewise-constant parameters. In particular, the segmentation accuracy is similar to that obtained from oracle-like methods in which the ground-truth parameters of the autoregressive models are known. We provide implementations of our algorithms at `https://github.com/ttesileanu/bio-time-series`.

1 Introduction

Detecting changes in the environment is essential to a living organism's survival (Koepcke et al., 2016). To a first approximation, environmental stimuli reaching our senses are generated by switching between different dynamical systems driven by a stochastic source. This implies that the temporal dependency structure of the stimuli, rather than their exact time evolution, contains information about the dynamics, which allows to detect important changes in the environment. For example, does the AC sound different today? Is your conversational partner's voice sad or cheerful? Is the recent surge in electrical activity in the brain indicative of an imminent seizure?

In this paper we develop unsupervised, biologically plausible neural architectures that segment time series data in an online fashion, clustering the underlying generating dynamics. Our focus here is different from typical applications of change-point detection in neuroscience (*e.g.,* (Beck et al., 2001; Yu, 2006)), as we are focusing on changes in the temporal correlation structure of the data, in contrast to changes in instantaneous statistics like the mean or the variance. We are also addressing a different question compared to sequence learning or identification (*e.g.,* (Brea et al., 2011, 2013; Memmesheimer et al., 2014)), since we aim to segment time series based on the dynamical processes that generated them, rather than the precise patterns that they contain in any given instance. Methods based on hidden Markov models (HMMs) have been used to segment spike-train data (Abeles et al., 1995; Jones et al., 2007; Mazzucato et al., 2015; Escola et al., 2011), but these generally do not work online and are not implemented using biologically plausible circuits. Methods similar to ours, but without a neural substrate, have also been used in econometrics (Ni and Yin, 2009; Ouyang and Yin, 2014) and for analysis of EEG data (Camilleri et al., 2015).

In order to build circuits that perform time-series clustering in a biologically plausible way, we require that learning occurs online and uses only local update rules. The first constraint is because biological learning and inference tend to happen in a streaming setting, with decisions taken as the sensory data is received and without the possibility of reprocessing the same data.[1] The second constraint reflects the fact that synaptic plasticity involves chemical processes that only have access to the local environment of the synapse. Synaptic updates thus typically depend only on the activity of pre- and post-synaptic neurons, potentially modified by a modulator, using a Hebb-like mechanism (Hebb, 2005; Kuśmierz et al., 2017).

Apart from constraints like the ones above, which one would expect to hold across all brain circuits, there are also limitations specific to certain areas. For instance, the retina in mammals does not receive feedback connections from the rest of the brain (Kandel et al., 2000), and so the results of computations performed downstream cannot be used to inform learning in the retina. In particular, algorithms involving the calculation of prediction errors seem implausible at this level. In other parts of the brain, however, neural correlates for prediction errors have been found (Schultz et al., 1997; Cohen, 2007; Egner et al., 2010; Tang et al., 2018), and thus this constraint can be lifted in those cases.

Here we show how the brain can implement time-series segmentation using two different biologically plausible architectures. If prediction error calculations are allowed, a model-based algorithm related to online k-means learning (Pehlevan et al., 2017) can solve the task effectively. The model predicts the existence of multiple independent modules in the brain, one for each learned generating process, and one global inhibitory neuron that silences all but the module that most accurately represents the data at any given time. Because of this latter feature we will typically refer to the model-based algorithm as "winner-take-all". We note also that this approach is related to developments in machine learning, particularly in the field of causal learning (Bengio et al., 2013; Schölkopf, 2019; Parascandolo et al., 2018; Locatello et al., 2019, 2018; Goyal et al., 2019).

In a second, model-free approach, a running estimate of the autocorrelation structure of the signal is clustered using a non-negative similarity-matching algorithm (Hu et al., 2014; Minden et al., 2018). By employing a metric that focuses on the similarity structure instead of encoding error, the model-free approach provides an architecture that does not require any feedback connections. This approach can therefore model brain areas in which such feedback is not available. The distinction between our model-based and model-free algorithms is similar to the difference between parametric and non-parametric models (Deng et al., 1997).

In the following sections, we will formally define the segmentation task; we will then introduce the winner-take-all algorithm and the autocorrelation-based algorithms together with their biological implementations; next we will compare their segmentation performance with each other and with an oracle-like method with roots in control theory; and we will end with a summary and discussion of future work.

2 Piecewise stationary autoregressive dynamics

More formally, we consider time series data $y(t)$ whose structure at any given time t is induced by one of a number of different stationary autoregressive (AR) processes (see Figure 1). Mathematically,

$$y(t) = \begin{cases} w_{11}y(t-1) + \cdots + w_{1p}y(t-p) + \epsilon(t)\,, & \text{if } \hat{z}(t) = 1, \\ \vdots & \\ w_{M1}y(t-1) + \cdots + w_{Mp}y(t-p) + \epsilon(t)\,, & \text{if } \hat{z}(t) = M, \end{cases} \tag{1}$$

where $\hat{z}(t)$ indicates the process that generated the sample at time t, and $\epsilon(t) \sim \mathcal{N}(0, \sigma^2)$ is white Gaussian noise. We can write this more compactly as

$$y(t) = \sum_{k=1}^{M} z_k(t) \left[\boldsymbol{w}_k^\top \boldsymbol{x}(t) + \epsilon(t) \right], \tag{2}$$

where we introduced the notation $\boldsymbol{x}(t)$ for the *lag vector* with components

$$x_i(t) = y(t-i)\,, \qquad i \in \{1, \ldots, p\}\,, \tag{3}$$

[1] Hippocampal replay is a notable exception (Pavlides and Winson, 1989; Buhry et al., 2011)).

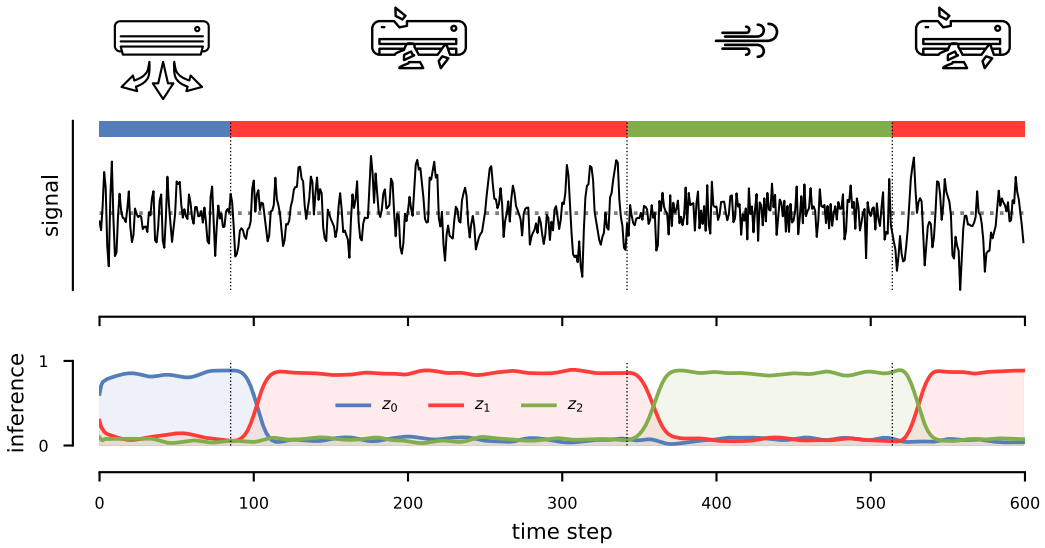

Figure 1: Sketch of the inference task. A signal (black line in top panel) is generated by alternating processes—*e.g.,* an intact AC, a broken AC, a gust of wind, as shown in the icons on the top, and also indicated by the colored ribbon between the icons and the signal line. The segmentation task amounts to identifying the transitions and clustering the sources (bottom panel). The z_k curves sketched in the bottom panel perform a soft clustering. We also sketched a delay in recognizing each transition, which is inherent in an online setting.

and used a one-hot encoding for the latent-state variable $\hat{z}_k(t)$,

$$z_k(t) = \begin{cases} 1 & \text{if } \hat{z}(t) = k, \\ 0 & \text{else.} \end{cases} \tag{4}$$

Our aim is thus to develop a biologically plausible mechanism that assigns each sample to a particular generative process (segmentation) and, in the model-based case, infer the process parameters (system identification). More specifically, this amounts to inferring the latent states $z_k(t)$ for the segmentation task and estimating the coefficients $\boldsymbol{w}_k$ for the system identification task.

There are several generalizations that we will not address in detail, but are straightforward to implement: handling processes with non-zero mean; allowing for more complex dependencies on past samples; and working with multidimensional time series. For example, we can lift the assumption on the mean by using an adaptation mechanism to subtract the mean from the data before segmentation. We can use arbitrary (but fixed) functions of the past, $x_i(t) = g_i(y(t-1), y(t-2), \dots)$, instead of time-lagged samples $x_i(t) = y(t-i)$, in eq. (2), with minimal changes to our algorithm. We can similarly extend our methods to continuous time using a set of fixed kernels, $x_i(t) = K_i(t) * y(t)$. And the generalization to multi-dimensional $\boldsymbol{y}(t) \in \mathbb{R}^d$ is also straightforward. In this work, we focus on the special case described in eq. (2), and leave these extensions for future work.

3 Model-based, winner-take-all algorithm

3.1 Basic framework

A natural approach for solving both the segmentation and system-identification tasks outlined above is to find the latent states $z_k(t)$ and AR coefficients $\boldsymbol{w}_k$ that minimize the discrepancy between the values of the signal predicted from eq. (2) and the actual observed values $y(t)$,

$$\min_{\boldsymbol{w}_k} \min_{\boldsymbol{z}} \frac{1}{2\sigma^2} \sum_t \sum_{k=1}^{M} z_k(t) \left| y(t) - \boldsymbol{w}_k^\top \boldsymbol{x}(t) \right|^2 \quad \text{such that } z_k(t) \text{ is one-hot,} \tag{5}$$

where σ is the standard deviation of the noise $\epsilon(t)$ from eq. (2).

The optimal $z_k(t)$ values at fixed $\boldsymbol{w}_k$ are given by the following expression (see Appendix A):

$$z_k(t) = \delta_{kk^*(t)}\,, \qquad k^*(t) = \arg\max_k -\frac{1}{2\sigma^2}\big|y(t) - \boldsymbol{w}_k^\top \boldsymbol{x}(t)\big|^2\,, \tag{6}$$

where $\delta_{kk^*} = 1$ if $k = k^*$ and 0 otherwise is the Kronecker delta. Intuitively, the best estimate for the latent state at time t is the one that produces the lowest prediction error. This depends both on the current estimate for the model coefficients $\boldsymbol{w}_k$ and on the recent history of the signal, represented by the lag vector $\boldsymbol{x}(t)$.

The optimal latent state assignments $z_k(t)$ and the optimal process parameters $\boldsymbol{w}_k$ depend on each other, which means that a full solution to the optimization problem (5) requires iteratively re-evaluating all the latent state variables $z_k(1), \ldots, z_k(t)$ and the coefficients $\boldsymbol{w}_k$. This is analogous to Lloyd's iterative solution for k-means clustering (Lloyd, 1982), but is unsuitable for an online algorithm where samples are presented one at a time and are generally not stored in memory.

To obtain an online approximation, we assume that the latent-state estimates $z_k(t)$ do not change once they are made. We thus apply eq. (6) only once for each time step t, and we then use stochastic gradient descent to update the coefficients $\boldsymbol{w}_k$ given a new sample. This yields:

$$\boldsymbol{w}_k(t+1) = \boldsymbol{w}_k(t) + \eta_w z_k(t)\boldsymbol{x}(t)\big[y(t) - \boldsymbol{x}(t)^\top \boldsymbol{w}_k(t)\big]\,, \tag{7}$$

where η_w is a learning rate.

We call the algorithm based on eqns. (6) and (7) "winner-take-all" because only one $z_k^*(t)$ is non-zero and only the weights associated with the inferred latent state, $\boldsymbol{w}_{k^*(t)}$, are updated at each step. Below we will relax this condition by softening the clustering and allowing several $z_k(t)$ to be non-zero, but it will generally hold true that the weight updates are strongest for the process that yields the best prediction for the sample. We will therefore sometimes refer to the algorithm as "soft" or "enhanced" winner-take-all.

3.2 Enhancements to the basic method

Soft clustering. Instead of forcing the latent-state vector $\boldsymbol{z}$ to be one-hot, as in eq. (6), we can soften the clustering by employing a soft max nonlinearity instead,

$$z_k(t) = \text{soft max}_T\left\{-\frac{1}{2\sigma^2}\big|y(t) - \boldsymbol{w}_k(t)^\top \boldsymbol{x}(t)\big|^2\right\}\,, \tag{8}$$

where

$$\text{soft max}_T \Delta_k = \frac{e^{\Delta_k/T}}{\sum_{k'} e^{\Delta_{k'}/T}}\,. \tag{9}$$

Here T is a "temperature" parameter controlling the softness of the clustering. In the $T \to 0$ limit, this reduces to the arg max solution.

This is equivalent to the following change in the objective function (*cf.* eq. (5); see derivation in Appendix A):

$$\min_{\boldsymbol{w}_k} \min_{\substack{\boldsymbol{z}\geq 0,\\ \sum z_k = 1}} \frac{1}{2\sigma^2}\sum_t \sum_{k=1}^M z_k(t)\big|y(t) - \boldsymbol{w}_k^\top \boldsymbol{x}(t)\big|^2 + T\left[\sum_{k=1}^M z_k \log z_k - 1\right]\,. \tag{10}$$

Persistence of latent states: penalizing transitions. In many realistic situations, the latent states exhibit some level of persistence: if the signal was generated by model $k^*(t)$ at time t, we can assume that the signal at time $t+1$ will likely be generated by the same model. Assuming persistence helps to avoid spurious switches in the inferred latent states that are due to noise. The downside is that actual switches in the state are more likely to be dismissed as noise.

One way to encourage persistence of the inferred latent states is to add a pairwise interaction term to the loss function:

$$\min_{\boldsymbol{w}_k} \min_{\boldsymbol{z}\geq 0} \frac{1}{2\sigma^2}\sum_t \sum_{k=1}^M z_k(t)\big|y(t) - \boldsymbol{w}_k^\top \boldsymbol{x}(t)\big|^2 - J\sum_k \sum_t z_k(t-1)z_k(t)\,, \tag{11}$$

where J controls the strength of the persistence correction. The extra term can be seen as a regularizer, or equivalently, as imposing a prior on the structure of the latent states.

In the online algorithm, this regularization has the effect of penalizing states that are different from the state at time $t-1$ by adding a term proportional to J to their squared prediction errors. Mathematically, eq. (6) is replaced by

$$z_k(t) = \delta_{kk^*(t)}\,, \qquad k^*(t) = \arg\max_k \left\{ -\frac{1}{2\sigma^2}\left|y(t) - \boldsymbol{w}_k(t)^\top \boldsymbol{x}(t)\right|^2 + J z_k(t-1) \right\}. \tag{12}$$

Averaging the squared error. A different approach that combines the signal across several consecutive samples is to replace the instantaneous squared prediction error $|y(t) - \boldsymbol{w}_k(t)^\top \boldsymbol{x}(t)|^2$ in eq. (12) with a time-average,

$$\begin{aligned} z_k(t) &= \delta_{kk^*(t)}\,, \\ k^*(t) &= \arg\max_k -\frac{\eta_\Delta}{2\sigma^2}\Big\{ |y(t) - \boldsymbol{w}_k(t)^\top \boldsymbol{x}(t)|^2 \\ &\qquad + (1-\eta_\Delta)|y(t-1) - \boldsymbol{w}_k(t-1)^\top \boldsymbol{x}(t-1)|^2 \\ &\qquad + (1-\eta_\Delta)^2|y(t-2) - \boldsymbol{w}_k(t-2)^\top \boldsymbol{x}(t-2)|^2 \\ &\qquad + \cdots \Big\} \\ &\equiv \arg\max_k -\frac{1}{2\sigma^2}\Delta_k(t)\,, \end{aligned} \tag{13}$$

where we used the notation $\Delta_k(t)$ for the exponential moving average (EMA) of the squared prediction error with smoothing factor η_Δ. This can be calculated online using

$$\Delta_k(t+1) = \eta_\Delta |y(t) - \boldsymbol{w}_k(t)^\top \boldsymbol{x}(t)|^2 + (1-\eta_\Delta)\Delta_k(t)\,. \tag{14}$$

Averaging the squared error mitigates the effect of noise on latent-state inference much like the penalty on state transitions described above does.

Final expression for latent-state estimates. Combining all the techniques in this section, we obtain the following expression for inferring the identity of the latent states:

$$z_k(t) = \text{soft max}_T \left\{ -\frac{1}{2\sigma^2}\Delta_k(t) + J z_k(t-1) \right\}. \tag{15}$$

3.3 Biologically plausible implementation

Algorithm 1 Biological winner-take-all method

function ProcessSample($\boldsymbol{x}$, y, z_k^{prev})
  $\Delta_k \leftarrow (1-\eta_\Delta)\Delta_k + \eta_\Delta \left|y(t) - \boldsymbol{w}_k^\top \boldsymbol{x}\right|^2$. ▷ averaged reconstruction error
  repeat ▷ output-neuron dynamics
    $z_k \leftarrow z_k - \eta_z\left[\frac{1}{2\sigma^2}\Delta_k - J z_k^{\text{prev}} + n + T\log z_k\right]$,
    $n \leftarrow n + \eta_n\left(\sum_k z_k - 1\right)$.
  until convergence
  $\boldsymbol{w}_k \leftarrow \boldsymbol{w}_k + \eta_w z_k \boldsymbol{x}\left[y - \boldsymbol{w}_k^\top \boldsymbol{x}\right]$. ▷ synaptic updates
  return z_k.
end function

We now construct a circuit that can implement the winner-take-all algorithm defined in eqns. (15) and (7) in a biologically plausible way. The key observation is that the latent state $z_k(t)$ can be obtained from the

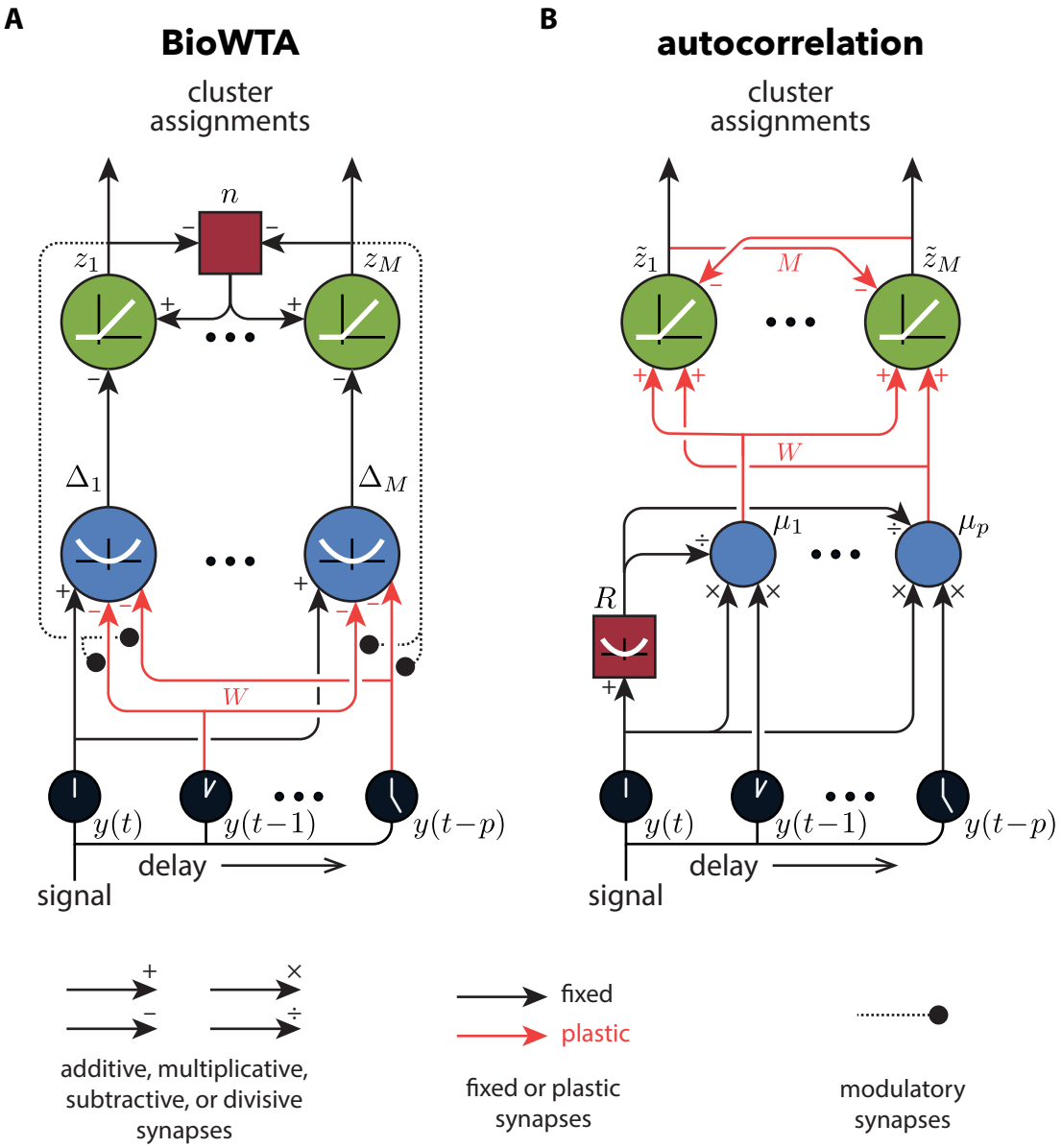

Figure 2: Biological implementations of our algorithms. (A) The model-based winner-take-all algorithm. (B) The model-free autocorrelation-based algorithm. The neurons are linear unless an activation function is shown. The neurons in blue are leaky integrators with timescales related to the appropriate learning rates in the models (see text); the other neurons are assumed to respond instantaneously.

following optimization problem (see Appendix A):

$$\min_{\boldsymbol{z}\geq 0}\max_{n}\sum_{k=1}^{M} z_k(t)\left[\frac{1}{2\sigma^2}\Delta_k(t) - Jz_k(t-1) + T\big(\log z_k(t) - 1\big) - n(t)\right] + n(t)\,, \tag{16}$$

where $\Delta_k(t)$ is the EMA of the squared prediction error (eq. (14)) and $n(t)$ is a Lagrange multiplier enforcing the constraint $\sum_k z_k(t) = 1$. Similar to (Pehlevan et al., 2017), we can solve this min-max optimization

objective with gradient descent-ascent dynamics:

$$\begin{aligned}\dot{z}_k(t) &= \eta_z \left(-\frac{1}{2\sigma^2}\Delta_k(t) + J z_k(t-1) + n(t) - T\log z_k(t) \right) , \\ \dot{n}(t) &= \eta_n \Big(1 - \sum_k z_k(t) \Big) .\end{aligned} \tag{17}$$

Note that here t refers to the sample index, while the dynamics happens on a fast timescale that must achieve convergence before the next sample can be processed.

Running the neural program above until convergence recovers the soft max solution of eq. (8). The $T\log z_k(t)$ term enforces the non-negativity constraint on $z_k(t)$ by going to negative infinity as $z_k(t) \to 0$.

Combining eq. (17) with the synaptic plasticity rule from eq. (7),

$$\boldsymbol{w}_k(t+1) = \boldsymbol{w}_k(t) + \eta_w z_k(t) \boldsymbol{x}(t) \big[y(t) - \boldsymbol{x}(t)^\top \boldsymbol{w}_k(t) \big] , \tag{18}$$

yields the basis of our biological winner-take-all neural circuit. The method is summarized in Algorithm 1, and the resulting circuit, providing a biological implementation of a neural attention mechanism modulated via competition, is sketched in Figure 2A .

The neurons labeled z_k in Figure 2A represent the cluster assignments and compete with each other via the interaction with the interneuron n. The "winning" clusters get their parameters updated following a three-factor Hebbian learning rule (Kuśmierz et al., 2017) (eq. (18)) at the x–Δ synapses, where the outputs from the z_k neurons are used as modulators. The circuit uses leaky integrator neurons with a quadratic nonlinearity (Δ_k) to estimate the average squared error from eq. (14). These neurons project to the output neurons (z_k) that implement the soft max function in conjunction with the normalizing (n) neuron, as in eq. (17).

4 Model-free, autocorrelation-based algorithm

Algorithm 2 Biological autocorrelation method

function ProcessSample($\boldsymbol{x}$, y)
  $R \leftarrow R + \eta_R(y^2 - R)$. ▷ autocorrelation update
  $\boldsymbol{\mu} \leftarrow \boldsymbol{\mu} + \eta_\mu\big(y\boldsymbol{x}/R - \boldsymbol{\mu}\big)$.

  $M_d \leftarrow$ diagonal part of M . ▷ output-neuron dynamics
  $M_o \leftarrow$ off-diagonal part of M .
  $\tilde{\boldsymbol{z}}_d \leftarrow M_d^{-1}(W\boldsymbol{\mu})$.
  $\tilde{\boldsymbol{z}} \leftarrow \tilde{\boldsymbol{z}}_d - M_d^{-1}(M_o\tilde{\boldsymbol{z}}_d)$.

  $W \leftarrow W + \alpha\,[\tilde{\boldsymbol{z}}\boldsymbol{\mu}^\top - W]$. ▷ synaptic updates
  $M \leftarrow M + \tau^{-1}\alpha\,[\tilde{\boldsymbol{z}}\tilde{\boldsymbol{z}}^\top - M]$.
  return $\tilde{z}_k$.
end function

In this section, we propose a network that operates without an explicit error calculation, in contrast to the winner-take-all circuit which relies on an estimate of the prediction error for both inference and learning. To do so, we combine a running estimate of the autocorrelation structure of the signal with a biologically plausible clustering algorithm.

The key observation is that the dynamical characteristics of a signal can be summarized through its autocorrelation structure. Indeed, the coefficients defining an autoregressive process are related to the autocorrelation function of the signal it generates through the Yule-Walker equations (Shumway et al., 2000), although the precise relationship is not important here. We summarize the autocorrelation structure using a

p-dimensional vector $\boldsymbol{\mu}$ with components

$$\begin{aligned} \mu_k &= \frac{1}{R}\mathbb{E}[y(t)y(t+k)]\,, \qquad \text{where } R \text{ is the variance,} \\ R &= \mathbb{E}[y(t)^2]\,. \end{aligned} \tag{19}$$

Time-series segmentation then reduces to calculating short-time estimates of the autocorrelation vectors $\boldsymbol{\mu}(t)$, and clustering the vectors obtained at different moments in time.

The following set of update rules can be used to estimate the autocorrelation structure in a streaming setting:

$$\begin{aligned} \Delta R(t) &= \eta_R\big(y(t)^2 - R(t)\big)\,, \\ \Delta\boldsymbol{\mu}(t) &= \eta_\mu\left[\frac{1}{R(t)}y(t)\boldsymbol{x}(t) - \boldsymbol{\mu}(t)\right]\,, \end{aligned} \tag{20}$$

where η_R and η_μ are learning rates. Note that, as in the winner-take-all algorithm, we are assuming that the input signal has mean zero. If it does not, a simple adaptation mechanism could be used to subtract the mean.

To cluster the vectors $\boldsymbol{\mu}(t)$ that summarize the dynamics at time t, we use non-negative similarity matching (NSM), an online algorithm that admits a simple neural interpretation (Hu et al., 2014; Minden et al., 2018). The NSM algorithm is based on the idea that signals with similar autocorrelation structure should be mapped to similar outputs,

$$\underset{\boldsymbol{u}(t)\geq 0}{\arg\min} \sum_{t,t'} \left|\boldsymbol{\mu}(t)^\top\boldsymbol{\mu}(t') - \tilde{\boldsymbol{z}}(t)^\top\tilde{\boldsymbol{z}}(t')\right|^2\,, \tag{21}$$

where we force the outputs to be non-negative, $\tilde{z}_k(t) \geq 0$. With this constraint, the outputs of the network perform soft clustering (Pehlevan and Chklovskii, 2014), such that $\tilde{z}_k(t)$ can act as an indicator function for whether the k^{th} generating process is responsible for the output at time t.

Note that unlike in the case of BioWTA, the outputs $\tilde{z}_k(t)$ from the autocorrelation-based algorithm do not in general sum to 1. This is why we use a slightly different notation here, $\tilde{z}_k(t)$ instead of $z_k(t)$, for the outputs. We can still recover the best guess for the latent state at time t, $z_k(t)$, by finding the largest $\tilde{z}_k(t)$:

$$z_k(t) = \delta_{kk^*(t)}\,, \text{with } k^*(t) = \arg\max_k \tilde{z}_k(t)\,. \tag{22}$$

The optimization from eq. (21) can be implemented using the following equations (Minden et al., 2018):

$$\begin{aligned} \tilde{\boldsymbol{z}}(t) &= \big[M(t)^{-1}W(t)\boldsymbol{\mu}(t)\big]_+\,, \\ W(t+1) &= W(t) + \alpha\big[\tilde{\boldsymbol{z}}(t)\boldsymbol{\mu}(t)^\top - W\big]\,, \\ M(t+1) &= M(t) + \tau^{-1}\alpha\big[\tilde{\boldsymbol{z}}(t)\tilde{\boldsymbol{z}}(t)^\top - M\big]\,, \end{aligned} \tag{23}$$

where α and $\tau^{-1}\alpha$ are learning rates and $[u]_+$ denotes a rectifying nonlinearity. Note how the synaptic weights W and M undergo Hebb-like dynamics. As in (Minden et al., 2018), in practice we use a more biologically plausible two-step approximation instead of calculating the inverse $M(t)^{-1}$ in the equation for $\tilde{\boldsymbol{z}}(t)$; see Algorithm 2.

A downside of the autocorrelation approach is that the coefficients $\boldsymbol{w}_k$ describing the generating processes are difficult to recover. Information about them is in principle contained in the weight matrices M and W, which encode information about the autocorrelation structure characteristic of each process. However, obtaining the AR coefficients from the weight matrices is a non-trivial task that our circuit does not perform.

We note also that the model-free algorithm implicitly assumes a level of persistence of the latent states because of the updating rules from eq. (20): the system needs a number of samples of order $1/\eta_\mu$ before it can detect a change in the autocorrelation structure, and so transitions happening on timescales faster than this will typically go undetected. The learning rate η_μ in the autocorrelation model thus plays a similar role as the η_Δ parameter used in the averaging step of the BioWTA algorithm, eq. (14).

The overall dynamics of the autocorrelation model combined with the clustering model can be represented using the neural architecture shown in Figure 2B. It is assumed that a quadratic nonlinearity from the signal

neurons y to the interneuron R implements the necessary squaring operation, with leaky integration responsible for averaging over recent samples. Inhibitory connections from the interneuron to the autocorrelation neurons μ_k perform the divisive normalization from eq. (20). The multiplications needed for updating the covariances can be the result of the synergistic effects of simultaneous spikes reaching the same neuron (Bugmann, 1991). A rectifying nonlinearity ensures the non-negativity of the outputs, and the synaptic updates from eq. (23) follow Hebbian and anti-Hebbian rules for the feedforward and lateral connections, respectively.

5 Numerical results

Table 1: Performance measures for our algorithms. Results are summarized over 100 runs, each run using a different 200,000-sample long signal generated using alternating AR(3) processes. The same 100 signals were used across the different algorithms. The plain BioWTA algorithm assumes hard clustering and no relation between latent states $z(t)$ at different times. The enhanced BioWTA algorithm uses soft clustering (eq. (8)) and the persistence correction (eq. (12)) described in the text. The cepstral algorithm assumes that the ground-truth AR coefficients are known and uses a running estimate of a cepstral norm to identify the generating process at each time (see text and Appendix H).

	autocorr.	plain BioWTA	enh. BioWTA	cepstral
Mean seg. score	0.75	0.73	0.88	0.89
Fraction well-segmented	0.40	0.21	0.70	0.79
Seg. score of bottom 5%	0.52	0.56	0.59	0.72
Mean weight error	—	0.79	0.76	—
Mean convergence time	620	7370	5040	—

We consider several ways to assess the effectiveness of our algorithms: segmentation accuracy; speed of convergence; and accuracy of system identification. We measure segmentation accuracy using a score equal to the fraction of time steps for which the inferred latent state is equal to the ground truth, up to a permutation.[2] We measure the speed of convergence by the number of steps needed to reach 90% of the final segmentation score. And we measure the accuracy of system identification by the root-mean-squared difference between the learned AR coefficients and the ground-truth coefficients, normalized by the size of the difference between the ground-truth coefficients. See Appendix B for details.

We use artificially generated time series to test our algorithms, as this gives us access to unambiguous ground-truth data. More specifically, we generate time series data by stochastically alternating several AR generative processes, themselves having coefficients that are chosen randomly for each signal. The switch between processes is governed by a semi-Markov model, ensuring a given minimum dwell time in every state. Beyond that minimum time, we use a constant probability of switching at each step, which is chosen to achieve a given average dwell time. To source the signals, we use a constant noise scale ($\epsilon(t)$ in eq. (2)), and we normalize the entire signal's variance to 1 before feeding it into the segmentation algorithms. See Appendix C for details.

In the simulations below, unless otherwise indicated, we use signals 200,000-samples long generated from two alternating AR(3) processes, each with a minimum dwell time of 50 steps and an average of 100. Typically only a fraction of the samples are needed for learning.

Table 1 summarizes the performance of our algorithms on a few different metrics, and compares it to an oracle-like cepstral method rooted in control theory. The sections below provide some detail and context for these results.

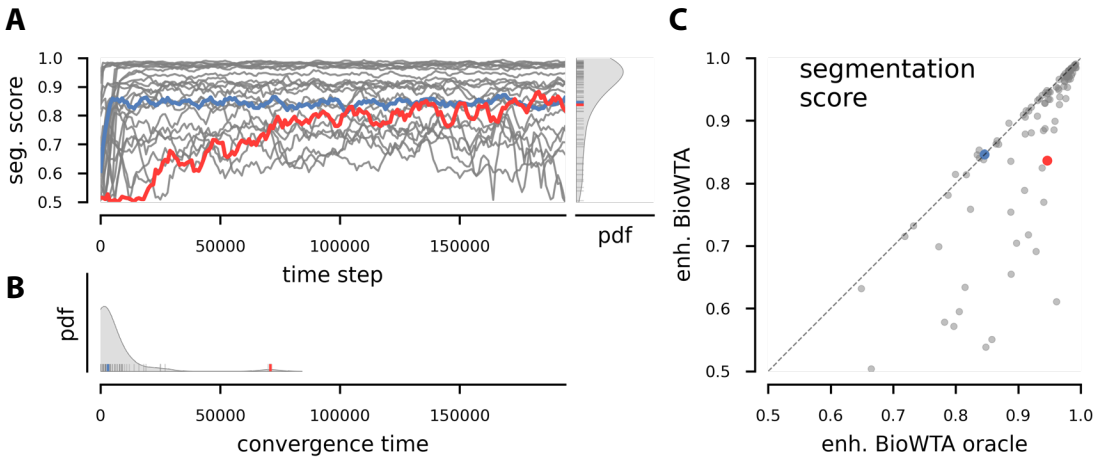

Figure 3: Segmentation accuracy for our BioWTA algorithm. (A) Rolling segmentation score for a subset of 10 of the 100 simulated runs. The blue and red traces single out runs that are used in subsequent figures. A kernel-density estimate of the distribution of segmentation scores for all 100 runs is shown on the right. (B) Kernel-density estimate of the convergence times for all 100 runs. Convergence is defined as reaching a segmentation score that is at least 90% of the final score. (C) Comparison of segmentation scores from our BioWTA learning algorithm (on the y-axis) with the scores obtained from a BioWTA "oracle"—an otherwise identical inference procedure in which the weights are kept fixed and equal to the ground-truth values.

5.1 Winner-take-all algorithm is highly accurate

To test the results of the winner-take-all algorithm, we first had to choose values for the learning rates η_w, η_Δ, the persistence parameter J, and the "temperature" from the soft max function, T. We did this by generating 200 random signals and running the simulation on these signals for 2000 randomly generated hyperparameters choices. We then selected parameter values that maximized the fraction of successful runs (defined as runs reaching a segmentation score of at least 85%). We obtained the best performance by using soft clustering $T > 0$ and a non-zero persistence parameter J, but no averaging, $\eta_\Delta = 1$. We call this the "enhanced" BioWTA algorithm. See Appendix D for details.

Figure 3 shows the performance of this enhanced winner-take-all algorithm on a new batch of 100 simulated time series. We find that segmentation is typically very accurate, reaching a median score after learning of 93%, with more than two thirds of runs exhibiting scores over 85%. Learning is relatively fast, too: almost 90% of runs converge to 90% of their final segmentation scores in less than 10,000 time steps.

5.2 Model coefficients are also learned by winner-take-all algorithm

One of the advantages of the model-based BioWTA algorithm compared to the model-free, autocorrelation-based one is that BioWTA learns the coefficients of the generating autoregressive processes. This should in principle allow the system to predict future inputs. But how well does weight learning actually work?

In Figure 4A, we show that most runs learn a noisy version of the AR weights, with deviations from the ground-truth that are smaller than the differences between the two sets of ground-truth coefficients. The accuracy of the weight reconstruction can become quite good in some cases, such as for the run highlighted in blue in the figure.

Good weight reconstruction implies high segmentation accuracy, but interestingly, the converse is often not true, as seen in Figure 4B. Consider, for instance, the run highlighted in red in Figure 4. The accuracy of the segmentation is essentially as good as that for the run highlighted in blue, but its weight reconstruction is much worse.

[2]Since ours is an unsupervised learning task, the ordering of models in the simulations can be different from the ground-truth ordering. We choose the mapping between simulation labels and ground-truth labels that maximizes the segmentation score. This implies that the segmentation score is always $\geq 1/M$, where M is the number of clusters.

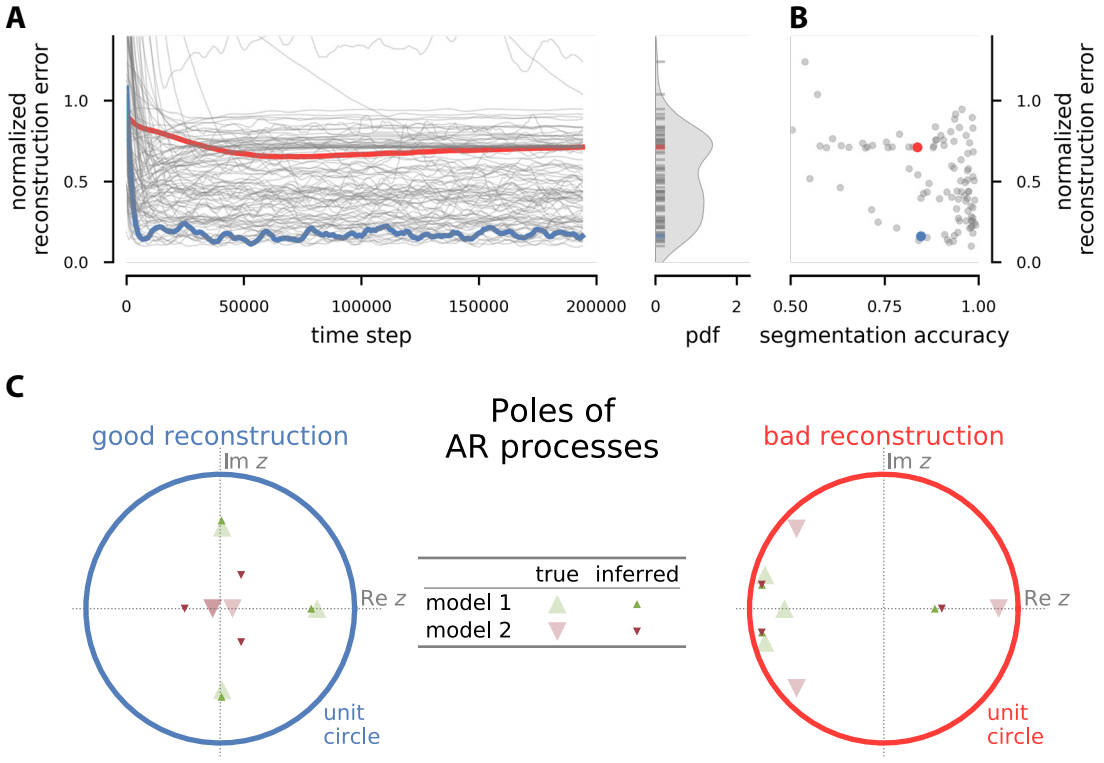

Figure 4: Weight reconstruction in the BioWTA algorithm. (A) Time evolution of the reconstruction error in the weights normalized by the difference between the ground-truth coefficients. The kernel-density plot on the right shows the distribution of final normalized reconstruction errors. (B) Relation between final normalized reconstruction error of the AR weights and segmentation accuracy. (C) Poles of the ground-truth (larger, faded triangles) and inferred (smaller, more saturated triangles) models for the runs shown in blue and red in panel A. The poles are the complex roots of the polynomial $z^p - w_1 z^{p-1} - \cdots - w_p$ and are a convenient representation of the properties of an autoregressive model (see Appendix G for details). Note how in the blue run, each inferred model is close to one ground-truth model, and very different from each other. In contrast, in the red run, both inferred models are very similar to each other and interpolate between the ground-truth models.

To understand how a run can exhibit poor weight reconstruction but good segmentation accuracy, it is convenient to look at the complex roots of the characteristic polynomial $z^p - w_1 z^{p-1} - \cdots - w_p$ where $\boldsymbol{w}$ are the AR coefficients for the inferred and ground-truth models. These roots are called *poles* and they correspond to different modes of the dynamical system.[3]

Figure 4C shows the poles for the inferred and ground-truth AR processes in a run with successful weight reconstruction (highlighted in blue) compared to a run where weight reconstruction failed (highlighted in red). The left panel shows good weight reconstruction: the poles for the inferred models are relatively close to the respective ground-truth values. In contrast, the right panel shows how the inferred models for the red run converged to almost a single point that interpolates between the two ground-truth models. Despite the bad weight reconstruction, the segmentation accuracy can still be high as long as the inferred models are different enough that samples from ground-truth model 1 are typically just a little bit more accurately described by inferred model 1 than inferred model 2, and *vice versa*.

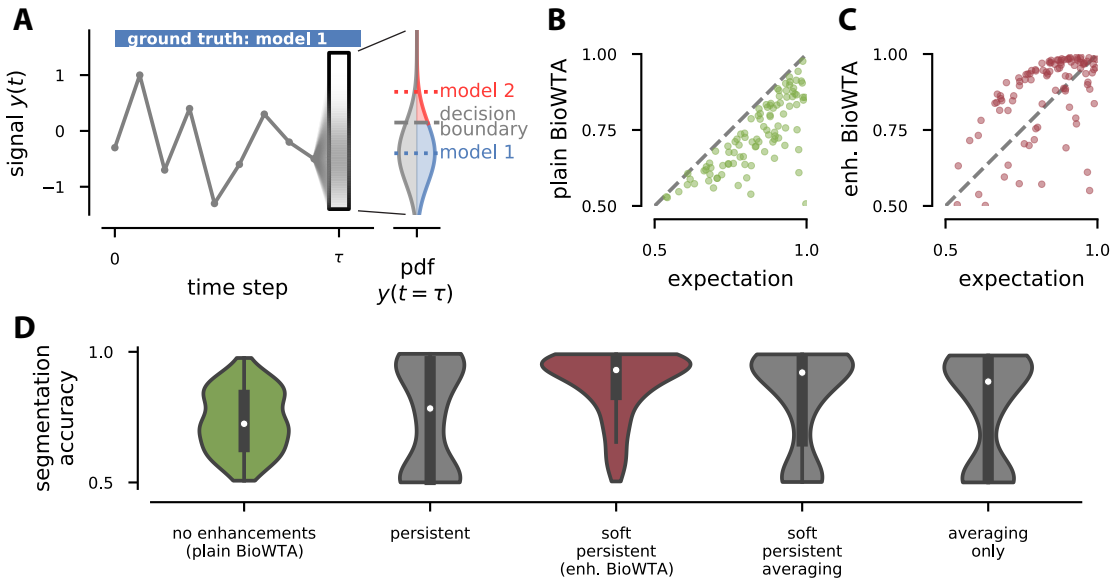

Figure 5: Understanding when and why BioWTA has trouble, and how enhancements help. (A) Sketch showing how latent-state inference can fail even if the ground-truth weights are exactly known. A fraction of samples from model 1 will predictably end up closer to the prediction from model 2 due to noise, and thus be misclassified. (B), (C) Comparison of segmentation score from our learning procedure to the naive prediction based on the argument from panel (A) (see eq. (24)). Panel (B) shows the results from the plain BioWTA algorithm (eq. (5)); panel (C) shows the improvement when using "enhanced" BioWTA, that is, when adding soft clustering (eq. (8)) and a persistence correction (eq. (12)). (D) Kernel-density estimates of the distribution of segmentation accuracies for several variations of our algorithms. The green and dark red violins correspond to the plain and enhanced BioWTA algorithms from panels (B) and (C), respectively.

5.3 Some segmentation problems are intrinsically harder

Although the BioWTA algorithm performs very well, it is clear from Figure 3 that a number of runs are not so successful. In some cases, such as when the segmentation accuracy stays close to chance level, this is due to a failure in learning, which in turn can happen if, for instance, the learning rate is too large for that particular case. There is, however, significant variability even among the runs that do converge—as can be seen from Figure 3C, which shows on the x-axis the segmentation scores of the BioWTA algorithm when the weights are kept fixed at their ground-truth values. Why is this?

The explanation for much of the variability seen in the segmentation accuracy of the BioWTA algorithm is that each randomly generated pair of AR processes can be more or less similar to each other. In the limit in which the two AR processes are identical, there would of course be no way to perform better than chance in the segmentation task. It is thus reasonable to expect that the segmentation accuracy depends on how different the two AR processes are.

This is indeed the case: even with perfect knowledge of the generating processes, segmentation will fail when a noise sample is large enough to move the signal into a range that is closer to the prediction from the wrong model. Indeed, our BioWTA algorithm infers which process the sample came from by choosing the one with the smallest prediction error.[4] This guess will often be correct, but it will inevitably also fail if the predictions are close enough or the noise large enough—even if in the "oracle" case where we have perfect knowledge of the parameters describing the generating models (Figure 5A).

We can derive an analytical expression for how frequently we would expect a segmentation error to occur in the "oracle" case, and use that to predict the segmentation accuracy. The result is the following (see

[3]Note that because the characteristic polynomial is real, complex roots always appear in complex-conjugate pairs.

[4]This is also the best possible way to make the inference if the noise scales are the same for the two processes and we cannot assume anything about the relation between the states at different times.

Appendix E):

$$\text{expected segmentation score} = \frac{1}{2} + \frac{\arctan\left(|\boldsymbol{w}_1 - \boldsymbol{w}_2|/\sigma\sqrt{\pi/8}\right)}{\pi}\,, \tag{24}$$

where $\boldsymbol{w}_1$ and $\boldsymbol{w}_2$ are the ground-truth coefficient vectors for the two processes and σ is the standard deviation of the noise, which is chosen here such that the standard deviation of the whole signal equals 1. This guess in fact does a great job of estimating an upper bound for the segmentation accuracy of our "plain" BioWTA algorithm—which uses hard clustering (*i.e.*, $T = 0$), no persistence correction ($J = 0$), and no error averaging ($\eta_\Delta = 1$); see Figure 5B.

5.4 Algorithm enhancements boost winner-take-all performance

Adding a persistence correction $J > 0$ significantly improves segmentation scores (see first two violins in Figure 5D), at the expense of some runs failing to converge. The latter happens because the simulation can get stuck in a single state and fail to learn both generating processes. This issue can be avoided by using soft clustering instead of hard clustering (third violin, in dark red, in Figure 5D). Interestingly, using soft-clustering on its own hurts rather than improve performance (see Appendix F). Also, adding error-averaging ($\eta_\Delta < 1$) to the soft, persistent BioWTA model can slightly hinder performance (fourth violin in Figure 5D); and in fact, this "fully-enhanced" BioWTA model is not much better than using error-averaging on its own (last violin in 5D). For each variation of the algorithm, the relevant hyperparameters were optimized according to the procedure described in section 5.1.

5.5 Autocorrelation-based algorithm learns faster but less accurately

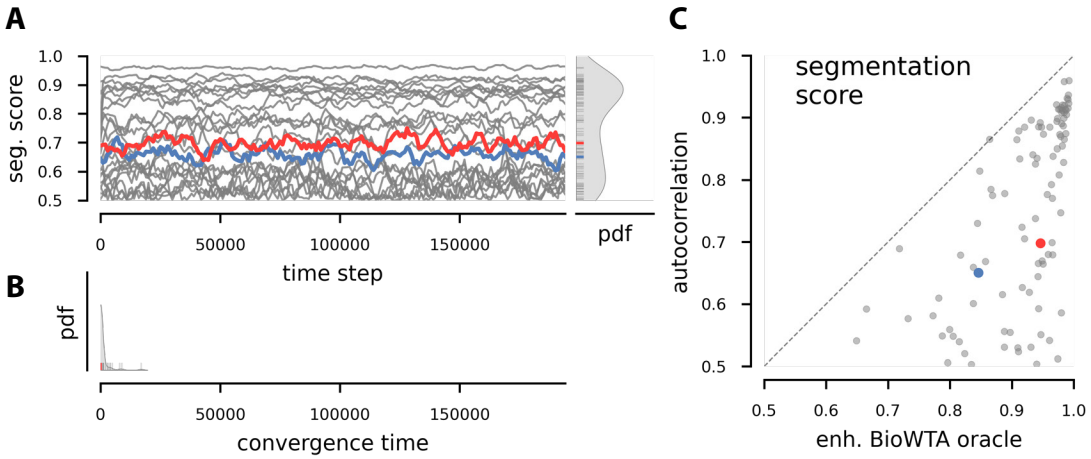

Figure 6: Segmentation accuracy for our autocorrelation-based algorithm. (A) Rolling segmentation score for a subset of 10 of the 100 simulated runs. The blue and red traces single out runs used in the other figures. A kernel-density estimate of the distribution of segmentation scores for all 100 runs is shown on the right. (B) Kernel-density estimate of the convergence times for all 100 runs. Convergence is defined as reaching a segmentation score that is at least 90% of the final score. (C) Comparison of segmentation scores from our autocorrelation learning algorithm (on the y-axis) with the scores obtained from an oracle-like BioWTA algorithm where the weights are kept fixed and equal to the ground-truth values.

Figure 6 shows the performance of the autocorrelation-based algorithm on the same 100 simulated time series used to test BioWTA above (Figure 3). The segmentation is less accurate than we obtained using BioWTA, but it still reaches a median score after learning of 78%, with 40% of runs scoring above 85%. Learning, however, is much faster than with BioWTA: 99% of runs reach 90% of their final segmentation score in less than 10,000 time steps; 84% converge in less than 1,000 steps (Figure 6B). As for the winner-take-all algorithm, we see that generally, signals generated by pairs of less similar AR processes are easier to segment using the autocorrelation method than ones where the generating processes are very similar.

5.6 BioWTA is competitive with oracle-like cepstral method

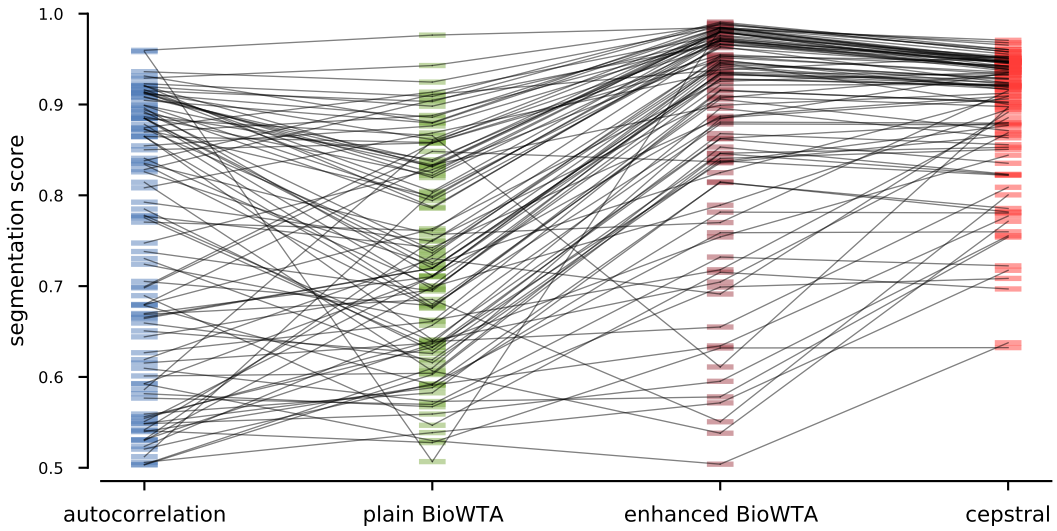

Figure 7: Comparison of segmentation accuracy across our algorithms, and with an oracle-like cepstral method. Each thick, colored horizontal mark corresponds to the segmentation accuracy estimated in the last fifth of a 200,000-sample signal. Each algorithm was tested on the same set of 100 signals drawn from alternating AR(3) processes. The corresponding marks between every pair of consecutive algorithms are connected by a thin gray line. This allows us to see, for instance, that although the plain BioWTA and autocorrelation method perform approximately the same in aggregate, the kinds of signals that are better segmented are very different between the two algorithms, so that many signals that are poorly segmented by the former work better with the latter and *vice versa*. In contrast, the ordering is roughly the same for the enhanced BioWTA results as for the plain BioWTA, with an almost uniform increase in quality for the latter. Meanwhile, the cepstral method is as accurate as enhanced BioWTA for the runs where the latter is relatively accurate, but it is not vulnerable to the cases where the coefficient learning fails, since it assumes knowledge of the ground-truth weights.

Finally, we compare our algorithms with a method from control theory that uses the ground-truth weights and a cepstral measure to perform segmentation. Specifically, the inverse (moving-average) process is calculated for each ground-truth AR generating process, and the time series $y(t)$ is filtered using each inverse. When the matching inverse filter is used, the filtered output should be uncorrelated white noise. We use a cepstral norm (De Cock, 2002; Boets et al., 2005) to determine how close to uncorrelated each filtered output is, and assign each time step to the model that yields the lowest cepstral norm. This method effectively relies on a rolling-window estimate for the cepstral norm, so like the autocorrelation method, it naturally takes advantage of the persistence of the latent states in our simulations. See Appendix H for details.

We find that our enhanced BioWTA method works basically as well as the oracle-like cepstral method, with the exception of a small fraction of runs that were likely unable to converge on a set of useful weights (see Figure 7). Meanwhile, the plain BioWTA and the autocorrelation-based methods work less well but can still achieve good segmentation performance on many runs.

6 Conclusion

In this work we developed two biologically plausible algorithms for segmenting a one-dimensional time series based on the autoregressive processes that generated it. One algorithm is model-based and takes a normative approach, following from an optimization objective that combines clustering with model learning. This method relies on an estimate of the prediction error for both making inferences about latent states and

learning the model parameters. An alternative algorithm is model-free and relies on an *ad-hoc* mechanism for computing a running estimate of the autocorrelation structure of the signal. This estimate is then plugged into a clustering algorithm to achieve segmentation.

Importantly, both algorithms act online, and can be implemented in small neural networks comprised of biologically plausible units and connections with local learning rules. These provide two different architectures to look for in animal brains, depending on whether prediction error is present or not in the particular circuit under study.

Our circuits perform their task very well: the model-based, winner-take-all method achieves segmentation accuracies on-par with an oracle-like cepstral method that takes the ground-truth model parameters for granted. It also performs well when learning model parameters, although this can take many more samples than just learning to perform a good segmentation. The model-free method is less accurate than the model-based approach, but has the advantage of requiring very little training before becoming effective. This comes at the cost of not learning the model parameters in a form that can be easily used for prediction.

There are several extensions of our methods that can be readily implemented: multi-dimensional signals are a straightforward generalization; continuous signals or more complicated (but fixed) time dependencies can be handled directly by using arbitrary kernels relating the predictor vectors $\boldsymbol{x}$ to the signal values y; and signals with non-zero or even changing means can be accommodated.

Of course, nature is often not linear, so the ability to learn the parameters of non-linear dynamical systems and segment a signal based on their usage in a biologically plausible way is an important avenue for future work. Nature also does not always exhibit sharp transitions between different modalities but rather allows for gradual transitions. Adding support for such phenomena in our models would connect our work to non-negative independent component analysis (ICA), another interesting thought that we leave for future research.

References

Abeles, M., Bergman, H., Gat, I., Meilijson, I., Seidemann, E., Tishby, N., and Vaadia, E. (1995). Cortical activity flips among quasi-stationary states. *PNAS*, 92(19):8616–8620.

Beck, D. M., Rees, G., Frith, C. D., and Lavie, N. (2001). Neural correlates of change detection and change blindness. *Nature neuroscience*, 4(6):645–650.

Bengio, Y., Courville, A., and Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE*, 35(8):1798–1828.

Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305.

Boets, J., De Cock, K., Espinoza, M., and De Moor, B. (2005). Clustering time series, subspace identification and cepstral distances. *Communication in Information and Systems*, 5(1):69–96.

Brea, J., Senn, W., and Pfister, J. P. (2011). Sequence learning with hidden units in spiking neural networks. *Advances in Neural Information Processing Systems*, 24:1422–1430.

Brea, J., Senn, W., and Pfister, J. P. (2013). Matching recall and storage in sequence learning with spiking neural networks. *Journal of Neuroscience*, 33(23):9565–9575.

Bugmann, G. (1991). Summation and multiplication: two distinct operation domains of leaky integrate-and-fire neurons. *Network: Computation in Neural Systems*, 2(4):489–509.

Buhry, L., Azizi, A. H., and Cheng, S. (2011). Reactivation, replay, and preplay: How it might all fit together. *Neural Plasticity*, 2011:203462.

Camilleri, T. A., Camilleri, K. P., and Fabri, S. G. (2015). Semi-supervised segmentation of EEG data in BCI systems. In *IEEE EMBC*, pages 7845–7848.

Cohen, M. X. (2007). Individual differences and the neural representations of reward expectation and reward prediction error. *Social Cognitive and Affective Neuroscience*, 2(1):20–30.

De Cock, K. (2002). *Principal angles in system theory, information theory and signal processing*. PhD thesis, Katholieke Universiteit Leuven.

De Cock, K. and De Moor, B. (2002). Subspace angles between ARMA models. *Systems and Control Letters*, 46(4):265–270.

Deng, K., Moore, A. W., and Nechyba, M. C. (1997). Learning to recognize time series: combining ARMA models with memory-based learning. In *IEEE CIRA '97*, pages 246–251.

Egner, T., Monti, J. M., and Summerfield, C. (2010). Expectation and surprise determine neural population responses in the ventral visual stream. *Journal of Neuroscience*, 30(49):16601–16608.

Escola, S., Fontanini, A., Katz, D., and Paninski, L. (2011). Hidden Markov models for the stimulus-response relationships of multistate neural systems. *Neural Computation*, 23(5):1071–1132.

Goyal, A., Lamb, A., Hoffmann, J., Sodhani, S., Levine, S., Bengio, Y., and Schölkopf, B. (2019). Recurrent independent mechanisms. *arXiv preprint arXiv:1909.10893*.

Hebb, D. O. (2005). *The organization of behavior: A neuropsychological theory*. Psychology Press.

Hu, T., Pehlevan, C., and Chklovskii, D. B. (2014). A Hebbian/Anti-Hebbian network for online sparse dictionary learning derived from symmetric matrix factorization. *Asilomar Conference on Signals, Systems and Computers*, pages 613–619.

Jones, L. M., Fontanini, A., Sadacca, B. F., Miller, P., and Katz, D. B. (2007). Natural stimuli evoke dynamic sequences of states in sensory cortical ensembles. *PNAS*, 104(47):18772–18777.

Kandel, E. R., Schwartz, J. H., Jessell, T. M., Siegelbaum, S., Hudspeth, A. J., and Mack, S. (2000). *Principles of neural science*, volume 4. McGraw-Hill New York.

Kuśmierz, Ł., Isomura, T., and Toyoizumi, T. (2017). Learning with three factors: modulating Hebbian plasticity with errors. *Current Opinion in Neurobiology*, 46:170–177.

Lloyd, S. P. (1982). Least Squares Quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137.

Locatello, F., Bauer, S., Lucic, M., Raetsch, G., Gelly, S., Schölkopf, B., and Bachem, O. (2019). Challenging common assumptions in the unsupervised learning of disentangled representations. In *ICML*, pages 4114–4124.

Locatello, F., Vincent, D., Tolstikhin, I., Rätsch, G., Gelly, S., and Schölkopf, B. (2018). Competitive training of mixtures of independent deep generative models. *arXiv preprint arXiv:1804.11130*.

Mazzucato, L., Fontanini, A., and La Camera, G. (2015). Dynamics of multistable states during ongoing and evoked cortical activity. *Journal of Neuroscience*, 35(21):8214–8231.

Memmesheimer, R.-M., Rubin, R., Ölveczky, B. P., and Sompolinsky, H. (2014). Learning Precisely Timed Spikes. *Neuron*, 82:1–14.

Minden, V., Pehlevan, C., and Chklovskii, D. B. (2018). Biologically Plausible Online Principal Component Analysis Without Recurrent Neural Dynamics. In *Asilomar Conference on Signals, Systems, and Computers*, pages 104–111.

Ni, H. and Yin, H. (2009). A self-organising mixture autoregressive network for FX time series modelling and prediction. *Neurocomputing*, 72(16-18):3529–3537.

Oppenheim, A. V., Buck, J. R., and Schafer, R. W. (2001). *Discrete-time signal processing*, volume 2. Upper Saddle River, NJ: Prentice Hall.

Ouyang, Y. and Yin, H. (2014). A neural gas mixture autoregressive network for modelling and forecasting FX time series. *Neurocomputing*, 135:171–179.

Parascandolo, G., Kilbertus, N., Rojas-Carulla, M., and Schölkopf, B. (2018). Learning independent causal mechanisms. In *ICML*, pages 4036–4044.

Pavlides, C. and Winson, J. (1989). Influences of hippocampal place cell firing in the awake state on the activity of these cells during subsequent sleep episodes. *The Jounral of Neuroscience*, 9(8):2907–2918.

Pehlevan, C. and Chklovskii, D. B. (2014). A Hebbian/anti-Hebbian network derived from online non-negative matrix factorization can cluster and discover sparse features. In *Asilomar Conference on Signals, Systems and Computers*, pages 769–775.

Pehlevan, C., Genkin, A., and Chklovskii, D. B. (2017). A clustering neural network model of insect olfaction. In *Asilomar Conference on Signals, Systems, and Computers*, pages 593–600.

Schölkopf, B. (2019). Causality for machine learning. *arXiv preprint arXiv:1911.10500*.

Schultz, W., Dayan, P., and Read Montague, P. (1997). A neural substrate of prediction and reward. *Science*, 275(5306):1593–1599.

Shumway, R. H., Stoffer, D. S., and Stoffer, D. S. (2000). *Time series analysis and its applications*, volume 3. Springer.

Tang, M. F., Smout, C. A., Arabzadeh, E., and Mattingley, J. B. (2018). Prediction error and repetition suppression have distinct effects on neural representations of visual information. *eLife*, 7:1–21.

Yu, A. J. (2006). Optimal change-detection and spiking neurons. *Advances in Neural Information Processing Systems*, 19:1545–1552.

A Derivation of the BioWTA algorithm

The BioWTA algorithm with all its enhancements (section 3.2) can be seen as an online approximation based on the following objective function:

$$\begin{aligned}\mathcal{L} &= \sum_t \sum_{k=1}^{M} z_k(t) \Bigg\{ \frac{\eta_\Delta}{2\sigma^2} \sum_{\tau=0}^{t} (1-\eta_\Delta)^\tau \left| y(t-\tau) - \boldsymbol{w}_k^\top \boldsymbol{x}(t-\tau) \right|^2 \\ &\qquad - J z_k(t-1) + T\big(\log z_k(t) - 1\big) \Bigg\} + \sum_t n(t) \Bigg[1 - \sum_{k=1}^{M} z_k(t) \Bigg] \\ &= \sum_t \Bigg\{ \sum_{k=1}^{M} z_k(t) \Bigg[\frac{1}{2\sigma^2} \Delta_k(t) - J z_k(t-1) + T\big(\log z_k(t) - 1\big) - n(t) \Bigg] + n(t) \Bigg\} ,\end{aligned} \tag{A.1}$$

where we added a sequence of Lagrange multipliers, $n(t)$, that help enforce the constraint $\sum_k z_k(t) = 1$ at every time step t. We are also assuming that the z_k variables are constrained to be non-negative, $z_k(t) \geq 0$. We use the convention $0 \log 0 = 0$ to make sense of the $z_k \log z_k$ terms when $z_k = 0$.

To obtain an online algorithm, we separate the objective function into a sequence of terms, one for each time step:

$$\mathcal{L} = \sum_t \ell(t) , \tag{A.2}$$

with

$$\ell(t) = \sum_{k=1}^{M} z_k(t) \Bigg[\frac{1}{2\sigma^2} \Delta_k(t) - J z_k(t-1) + T\big(\log z_k(t) - 1\big) - n(t) \Bigg] + n(t) . \tag{A.3}$$

We now make the online approximation by considering $\ell(t)$ alone to be the objective function that we use when processing the tth sample. Differentiating with respect to $z_k(t)$ and $n(t)$ and using gradient descent-ascent yields the fast dynamics:

$$\begin{aligned}\dot{z}_k(t) &= -\eta_z \frac{\partial \ell(t)}{\partial z_k(t)} = -\eta_z \left[\frac{1}{2\sigma^2} \Delta_k(t) - J z_k(t-1) + T \log z_k(t) - n(t) \right] , \\ \dot{n}(t) &= \eta_n \frac{\partial \ell(t)}{\partial z_k(t)} = \eta_n \Bigg[1 - \sum_{k=1}^{M} z_k(t) \Bigg] .\end{aligned} \tag{A.4}$$

In our simulations we do not explicitly model these fast variables, but instead directly set $z_k(t)$ to the fixed-point solution, eq. (15).

Differentiating $\ell(t)$ with respect to the process coefficients $\boldsymbol{w}_k$ and using gradient descent, we get

$$\begin{aligned}\Delta \boldsymbol{w}_k &= -\frac{\eta_w \sigma^2}{\eta_\Delta} \frac{\partial \ell(t)}{\partial \boldsymbol{w}_k} = -\frac{\eta_w}{2\eta_\Delta} z_k(t) \frac{\partial \Delta_k(t)}{\partial \boldsymbol{w}_k} \\ &= -\frac{\eta_w}{2} z_k(t) \sum_{\tau=0}^{t} (1-\eta_\Delta)^\tau \frac{\partial}{\partial \boldsymbol{w}_k} \left| y(t-\tau) - \boldsymbol{w}_k^\top \boldsymbol{x}(t-\tau) \right|^2 \\ &= \eta_w z_k(t) \sum_{\tau=0}^{t} (1-\eta_\Delta)^\tau \boldsymbol{x}(t-\tau) \big(y(t-\tau) - \boldsymbol{w}_k^\top \boldsymbol{x}(t-\tau) \big) .\end{aligned} \tag{A.5}$$

This depends on the history of the input which we would want to avoid: in an online setting we do not want to keep many things in memory. The approach taken in the text simply ignores terms with $\tau > 0$, which makes sense if η_Δ is not much smaller than 1. Then eq. (A.5) reduces to

$$\Delta \boldsymbol{w}_k \approx \eta_w z_k(t) \boldsymbol{x}(t) \big(y(t) - \boldsymbol{w}_k^\top \boldsymbol{x}(t) \big) , \tag{A.6}$$

which matches the text.

A different approach would involve keeping track of the expression

$$\bar{\boldsymbol{x}}(t) = \sum_{\tau=0}^{t} (1-\eta_\Delta)^\tau \boldsymbol{x}(t-\tau)\big(y(t-\tau) - \boldsymbol{w}_k^\top \boldsymbol{x}(t-\tau)\big)\,, \tag{A.7}$$

which is akin to an eligibility trace. This obeys

$$\begin{aligned}\bar{\boldsymbol{x}}(t) &= \boldsymbol{x}(t)\big(y(t) - \boldsymbol{w}_k^\top \boldsymbol{x}(t)\big) + \sum_{\tau=1}^{t} (1-\eta_\Delta)^\tau \boldsymbol{x}(t-\tau)\big(y(t-\tau) - \boldsymbol{w}_k^\top \boldsymbol{x}(t-\tau)\big) \\ &= \boldsymbol{x}(t)\big(y(t) - \boldsymbol{w}_k^\top \boldsymbol{x}(t)\big) + \bar{\boldsymbol{x}}(t-1)\,.\end{aligned} \tag{A.8}$$

Note that there is a subtlety in the expression above: in principle, the value that we use at time t for $\bar{\boldsymbol{x}}(t')$ for $t' < t$ should depend on $\boldsymbol{w}_k(t)$, not on earlier values of $\boldsymbol{w}_k$. This would again pose problems in an online setting, so we can use the approximation

$$\bar{\boldsymbol{x}}(t) = \boldsymbol{x}(t)\big(y(t) - \boldsymbol{w}_k(t)^\top \boldsymbol{x}(t)\big) + \bar{\boldsymbol{x}}(t-1)\,. \tag{A.9}$$

We do not pursue this alternative approach here.

B Accuracy measures

Segmentation accuracy We define segmentation accuracy by measuring the fraction of time steps for which the inferred segmentation matches the ground-truth. We ignore the first p samples for models using p-dimensional coefficient vectors $\boldsymbol{w}_k$, since no prediction can be made for an autoregressive process without having sufficient historical data.

The inferred labels can be a permutation of the ground-truth labels, since learning is unsupervised. To account for this, we use a minimum-weight matching algorithm (`linear_sum_assignment` from `scipy.optimize`) to find the permutation that maximizes the segmentation score. A side effect of this is that the segmentation score cannot drop below $1/M$, where M is the number of models in the simulation.

For calculating the evolution of the accuracy score with time, we use a rolling window and apply the method described above in each window. In particular, this allows the inferred-to-ground-truth permutation to be different for different positions of the rolling window. The step by which we shift the rolling window is typically smaller than the size of the window itself. As described in the text, we use a window size of 5000 and a step of 1000 in this paper.

Weight-reconstruction accuracy The weight reconstruction error is calculated by taking the differences between inferred and ground-truth coefficients, and normalizing these by the magnitude of the difference between the ground-truth values. More specifically, the error is given by:

$$\text{normalized weight-reconstruction error} = \frac{\sqrt{2\sum_k \big[\boldsymbol{w}_k^{\text{inferred}} - \boldsymbol{w}_{\sigma(k)}^{\text{true}}\big]^2}}{\big|\boldsymbol{w}_2^{\text{true}} - \boldsymbol{w}_1^{\text{true}}\big|}\,, \tag{B.1}$$

where σ is the permutation that maps each inferred model with the ground-truth model that it matches best.

The measure defined above has the useful property that if both sets of model weights converge to the same value in-between the two ground-truth coefficients, $\boldsymbol{w}_k^{\text{inferred}} \to (\boldsymbol{w}_1^{\text{true}} + \boldsymbol{w}_2^{\text{true}})/2$, then the normalized weight-reconstruction error is 1. An even larger error, $\sqrt{2}$, is obtained if both inferred weights converge to a single one of the true models, $\boldsymbol{w}_k^{\text{inferred}} \to \boldsymbol{w}_1^{\text{true}}$.

The normalized weight reconstruction error is calculated using the instantaneous weight values at each time step. The error for an entire run is defined to be the error at the final time step. The time evolution of the weight reconstruction (Figure 4A) employs a rolling average of the normalized reconstruction scores, with rolling-window size and step size equal to those used for calculating the rolling segmentation score (5000 steps and 1000 steps, respectively).

C Signal generation

We generate the signals for testing our segmentation algorithms in two steps: (1) generate the latent-state sequence, and (2) generate AR samples. The parameters of the autoregressive processes themselves are chosen randomly, as discussed below.

Latent-state sequence generation We sample the latent states from a discrete-time semi-Markov model with dwell times distributed according to a truncated geometric distribution. In other words, each latent state persists for a minimum number of steps, beyond which the system switches to a different latent state with a fixed probability at each step.

Autoregressive sample generation Samples are generated directly according to the model definition in eq. (2). For the first p samples, some components of the lag vector are not defined; we define them by setting $y(t) = 0$ for $t \leq 0$. Thus we can expect the first few samples of each signal to behave like a transient before stationarity is reached. After each latent-state transition, the output from the new model will depend on past samples that are generated by the old model for the first p time steps.

Choice of autoregressive processes We generate random autoregressive processes by starting with randomly generated poles. For this purpose, we choose a maximum pole radius $r_{\max}$ and generate $\lfloor p/2 \rfloor$ complex numbers uniformly distributed inside the disk of radius $r_{\max}$. These and their complex conjugates will by chosen as poles. If n is odd, we additionally generate one single real pole, drawn uniformly from $[-r_{\max}, r_{\max}]$. We then build the monic polynomial that has these poles as roots and set it equal to $z^p - w_1 z^{p-1} - \cdots - w_p$ to read off the coefficients w_k.

We typically set the maximum pole radius $r_{\max}$ to 0.95 to ensure that the generated processes are stable.

D Hyperparameter optimization

The algorithms that we use depend on a number of parameters, such as the learning rate η_w, the temperature T, and the persistence parameter J in the case of enhanced BioWTA. We optimize these parameters once for every choice of algorithm and type of signal. Different choices of enhancements of BioWTA count as different algorithms. The type of signal is set by the order of the autoregressive processes and the parameters of the semi-Markov model that dictates the latent-state sequence. Throughout this paper we use only one choice for the type of signal—AR order $p = 2$, minimum dwell time 50, and average dwell time 100—so all hyperparameter optimizations are done for that case.

We use uniform random sampling of hyperparameter values to perform hyperparameter optimization. Random search has been shown to be one of the best hyperparameter optimization methods when the number of hyperparameters is small (Bergstra and Bengio, 2012).

The performance of our algorithms varies due to several factors. The hyperparameter choices change the way the algorithms behave. The initial conditions affect initial transients and unlucky choices can lead to getting caught in local optima. And the signal-generation process itself is stochastic. We thus summarize the performance of the inference procedure for a batch of signals at any fixed value of the hyperparameters.

Specifically, for a number N_h of randomly generated hyperparameter tuples, we choose N_s random signals and measure the algorithm's segmentation accuracy on each signal at each value of the hyperparameters. We summarize the score for each hyperparameter tuple by using the fraction of "successful" runs, where successful is defined as having a segmentation score above some threshold θ_{good}. This method balances the desire for runs that reach very good segmentation scores with the requirement that only a few of the runs diverge (thus receiving close to the minimum segmentation score, $1/M$).

Throughout this paper, we used $N_h = 2000$ hyperparameter choices, $N_s = 200$ signals per batch, and $\theta_{\text{good}} = 0.85$ to define successful runs.

E Theoretical estimate of oracle BioWTA segmentation score

To estimate how the theoretically maximum segmentation score of plain BioWTA depends on the difference between the two ground-truth AR processes, we assume an "oracle" setting, where the two inferred models are kept equal to their ground-truth counterparts, $\boldsymbol{w}_k^{\text{inferred}} = \boldsymbol{w}_k^{\text{true}}$, and use the argument sketched in Figure 5A to estimate the fraction of misclassified samples.

More specifically, assume that a particular signal is generated by ground-truth model 1. The algorithm, however, will assign this sample to whichever process yields the lowest squared prediction error,

$$z_k = \arg\min_k |y - \boldsymbol{w}_k^\top \boldsymbol{x}|^2 \,. \tag{E.1}$$

Note that we are omitting the time index in this section, since it is always equal to t, and we are also omitting the "true" superscript on the ground-truth coefficients $\boldsymbol{w}_k$.

For model 1, the squared prediction error is simply given by the noise sample $\epsilon(t) \equiv \epsilon$ (see eq. (2)). For model 2, we have

$$\begin{aligned} |y - \boldsymbol{w}_2^\top \boldsymbol{x}|^2 &= |y - \boldsymbol{w}_1^\top \boldsymbol{x} - \Delta\boldsymbol{w}^\top \boldsymbol{x}|^2 \\ &= |\epsilon - \Delta\boldsymbol{w}^\top \boldsymbol{x}|^2 \,, \end{aligned} \tag{E.2}$$

where

$$\Delta\boldsymbol{w} = \boldsymbol{w}_2 - \boldsymbol{w}_1 \tag{E.3}$$

is the difference between the two ground-truth models.

Now, the sample will be assigned to the correct model (model 1) provided we have

$$\epsilon^2 < (\epsilon - \Delta\boldsymbol{w}^\top \boldsymbol{x})^2 \,. \tag{E.4}$$

Expanding the square, this yields

$$s^2 > 2\epsilon s \,, \tag{E.5}$$

with s denoting the separation between the predictions from the two models,

$$s = \Delta\boldsymbol{w}^\top \boldsymbol{x} \,. \tag{E.6}$$

Dividing through by $2s$, we get

$$\text{correct prediction: } \begin{cases} \epsilon < \frac{1}{2} s & \text{if } s \geq 0, \\ \epsilon > -\frac{1}{2}|s| & \text{if } s < 0. \end{cases} \tag{E.7}$$

This corresponds to the area shaded in blue in Figure 5A.

Since ϵ is drawn from a normal distribution with standard deviation σ, we can calculate the probability of a correct prediction:

$$P_{\text{correct}} = \text{NormalCDF}\left(\frac{1}{2\sigma}|s|\right) = \frac{1}{2} + \frac{1}{2}\,\text{erf}\left(\frac{|s|}{\sigma\sqrt{8}}\right) \,. \tag{E.8}$$

Note that due to the symmetry of the Gaussian distribution, this works for either sign of s.

The expression above tells us how accurately we can expect the cluster assignment for a given sample to be. Suppose we are now looking at an entire signal where the components of $\boldsymbol{x}$ are drawn from a normal distribution with standard deviation Σ. The expected accuracy is

$$\mathbb{E}\big[P_{\text{correct}}\big] = \frac{1}{2} + \frac{1}{2}\mathbb{E}\left[\text{erf}\left(\frac{|\Delta\boldsymbol{w}^\top \boldsymbol{x}|}{\sigma\sqrt{8}}\right)\right] \,. \tag{E.9}$$

The value $\Delta\boldsymbol{w}^\top \boldsymbol{x}$ is itself normally distributed, with mean and variance given by

$$\begin{aligned} \mathbb{E}\big[\Delta\boldsymbol{w}^\top \boldsymbol{x}\big] &= 0 \,, \\ \mathbb{E}\big[(\Delta\boldsymbol{w}^\top \boldsymbol{x})^2\big] &= \mathbb{E}\left[\sum_{i,j} \Delta w_i \Delta w_j x_i x_j\right] = \sum_{i,j} \Delta w_i \Delta w_j \text{cov}_{ij} \\ &\sim |\Delta\boldsymbol{w}|^2 \Sigma^2 \,, \end{aligned} \tag{E.10}$$

where in the last line we neglected cross-correlations between different components of $\boldsymbol{x}$. Note that these cross-correlations are not necessarily small—the fact that our signals are generated by autoregressive processes implies that these correlations are there and potentially large. Our approximation is simply meant to give a rough estimate for the expected segmentation score of plain BioWTA. Simulations suggest that our estimate is indeed quite good, Figure 5B.

We thus have

$$\mathbb{E}\big[P_{\text{correct}}\big] = \frac{1}{2} + \frac{1}{2}\mathbb{E}\left[\operatorname{erf}\left(\frac{\Sigma}{\sigma}\frac{|\Delta \boldsymbol{w}|}{\sqrt{8}}\, z\right)\right], \quad \text{for } z \sim \mathcal{HN}(0,1), \tag{E.11}$$

where $\mathcal{HN}(0,1)$ is the half-normal distribution. The half-normal appears here as a direct result of eq. (E.9).

This expectation value can actually be calculated analytically (we used `Mathematica`), and the result is

$$\mathbb{E}\big[P_{\text{correct}}\big] = \frac{1}{2} + \frac{1}{\pi}\arctan\left(\frac{\Sigma}{\sigma}\sqrt{\frac{\pi}{8}}|\Delta \boldsymbol{w}|\right). \tag{E.12}$$

Now, the expression above gives the probability of assigning a sample to, *e.g.,* the first cluster *provided* the sample was, in fact, generated from that cluster. In a typical simulation run used in this paper, the ground truth will alternate between the two clusters, spending about half the time in each one. This implies that the overall segmentation accuracy score should be given by

$$\begin{aligned}\text{predicted accuracy} &= \frac{1}{2} + \frac{1}{2\pi}\left[\arctan\left(\frac{\Sigma_1}{\sigma}\sqrt{\frac{\pi}{8}}|\Delta \boldsymbol{w}|\right) + \arctan\left(\frac{\Sigma_2}{\sigma}\sqrt{\frac{\pi}{8}}|\Delta \boldsymbol{w}|\right)\right] \\ &= \frac{1}{2} + \frac{1}{2\pi}\left[\arctan \alpha\Sigma_1 + \arctan \alpha\Sigma_2\right],\end{aligned} \tag{E.13}$$

where Σ_i is the standard deviation of the samples generated from process i, and we introduced the notation

$$\alpha \equiv \frac{1}{\sigma}\sqrt{\frac{\pi}{8}}|\Delta \boldsymbol{w}|. \tag{E.14}$$

In our simulations, we choose the noise standard deviation σ such that the overall variance of the output is 1. Since the generating process is split 50-50 between the two possible latent states, this implies

$$\frac{1}{2}\left(\Sigma_1^2 + \Sigma_2^2\right) = 1. \tag{E.15}$$

The specific values for Σ_i will depend on the run, but in order to get a rough answer (that will turn out to work well in practice), we choose the case in which the two are approximately equal,

$$\Sigma_1 \approx \Sigma_2 = 1. \tag{E.16}$$

This means that we can estimate

$$\text{predicted accuracy} \approx \frac{1}{2} + \frac{1}{\pi}\arctan \alpha \equiv \frac{1}{2} + \frac{1}{\pi}\arctan\frac{1}{\sigma}\sqrt{\frac{\pi}{8}}|\Delta \boldsymbol{w}|. \tag{E.17}$$

This is the formula we used in the paper.

General case Employing a standard trigonometric identity and using the fact that α, Σ_1, and Σ_2 are all positive, we can rewrite eq. (E.13) as[5]

$$\text{predicted accuracy} = \frac{1}{2} + \frac{1}{2\pi}\operatorname{arccot}\frac{1 - \alpha^2\Sigma_1\Sigma_2}{\Sigma_1 + \Sigma_2}. \tag{E.18}$$

From eq. (E.15), we find

$$(\Sigma_1 + \Sigma_2)^2 = 2(1 + \Sigma_1\Sigma_2), \tag{E.19}$$

[5]We use the arccot here instead of arctan because the range of arctan is $[-\pi/2, \pi/2]$, while the sum $\arctan \alpha\Sigma_1 + \arctan \alpha\Sigma_2$ can range from 0 to π.

which, using the notation

$$\theta \equiv \frac{\Sigma_1 + \Sigma_2}{2}\,, \tag{E.20}$$

yields

$$\text{predicted accuracy} = \frac{1}{2} + \frac{1}{2\pi}\,\text{arccot}\,\frac{1-\alpha^2(2\theta^2-1)}{2\alpha\theta}\,. \tag{E.21}$$

Now, eq. (E.19) shows that the sum of the two standard deviations, $\Sigma_1 + \Sigma_2$, is at least $\sqrt{2}$. Put differently, $\theta \geq \frac{\sqrt{2}}{2} \approx 0.71$. Conversely, the product of two numbers with a fixed sum is maximum when the two numbers are equal to each other, which, in combination with eq. (E.15), implies that $\Sigma_1^2\Sigma_2^2 \leq 1$. This in turn means that $\theta \leq 1$. All in all, the mean standard deviation is confined to a rather small range of values:

$$\frac{\sqrt{2}}{2} \leq \theta \leq 1\,. \tag{E.22}$$

F Performance effects of BioWTA enhancements

We saw in the text that the enhancements we described for the BioWTA method—the persistence correction controlled by the J parameter; the soft clustering controlled by the temperature T; and the averaging of the squared error controlled by the η_Δ parameter—affect performance in non-trivial way. Figure 8 shows this in more detail.

In particular, note that soft clustering on its own has a consistent detrimental effect on segmentation performance, but the highest-scoring method includes soft clustering in addition to the persistence correction. This latter method performs significantly better than if the persistence correction is used on its own (see first, sixth, and last row and column in Figure 8).

The error-averaging correction has the opposite behavior: on its own it yields results almost as good as the top-performing method, although it is more vulnerable to convergence failure. On the other hand, when combined with the other enhancements, it fails to improve the high-performing runs and instead hurts performance by hindering convergence in some runs (see first through fourth row and column in Figure 8). The only exception is when used in conjunction with soft clustering, which works better than either having only one of the enhancements, or none at all (see second, fifth, and last row and column in Figure 8).

Persistence on its own improves performance significantly for many runs, but leads to convergence problems in other runs (see sixth and seventh row and column in Figure 8). It behaves almost identically to the error-averaging correction alone when used in conjunction with it (second and fourth row and column in Figure 8), but it has the best performance of all the methods we've tried when used with soft clustering (first and last row and column in Figure 8).

G Some details about ARMA processes

ARMA processes and inverses It can be useful to think of an extension of AR models, the autoregressive moving-average (ARMA) process. These involve a weighted moving average (MA) of the noise signal in addition to the autoregressive part,

$$y(t) = w_1 y(t-1) + \cdots + w_p y(t-p) + \epsilon(t) + b_1\epsilon(t-1) + \cdots + b_q\epsilon(t-q)\,. \tag{G.1}$$

The output of an AR process can be inverted using an MA process to get back at the noise sequence $\epsilon(t)$:

$$\begin{aligned} \text{if } y(t) &= w_1 y(t-1) + \cdots + w_p y(t-p) + \epsilon(t)\,, \\ \text{then } \epsilon(t) &= y(t) - w_1 y(t-1) - \cdots - w_p y(t-p)\,. \end{aligned} \tag{G.2}$$

More generally, any ARMA process admits an inverse (though the inverse process might be unstable). This is relevant for the cepstral oracle method described in the next section.

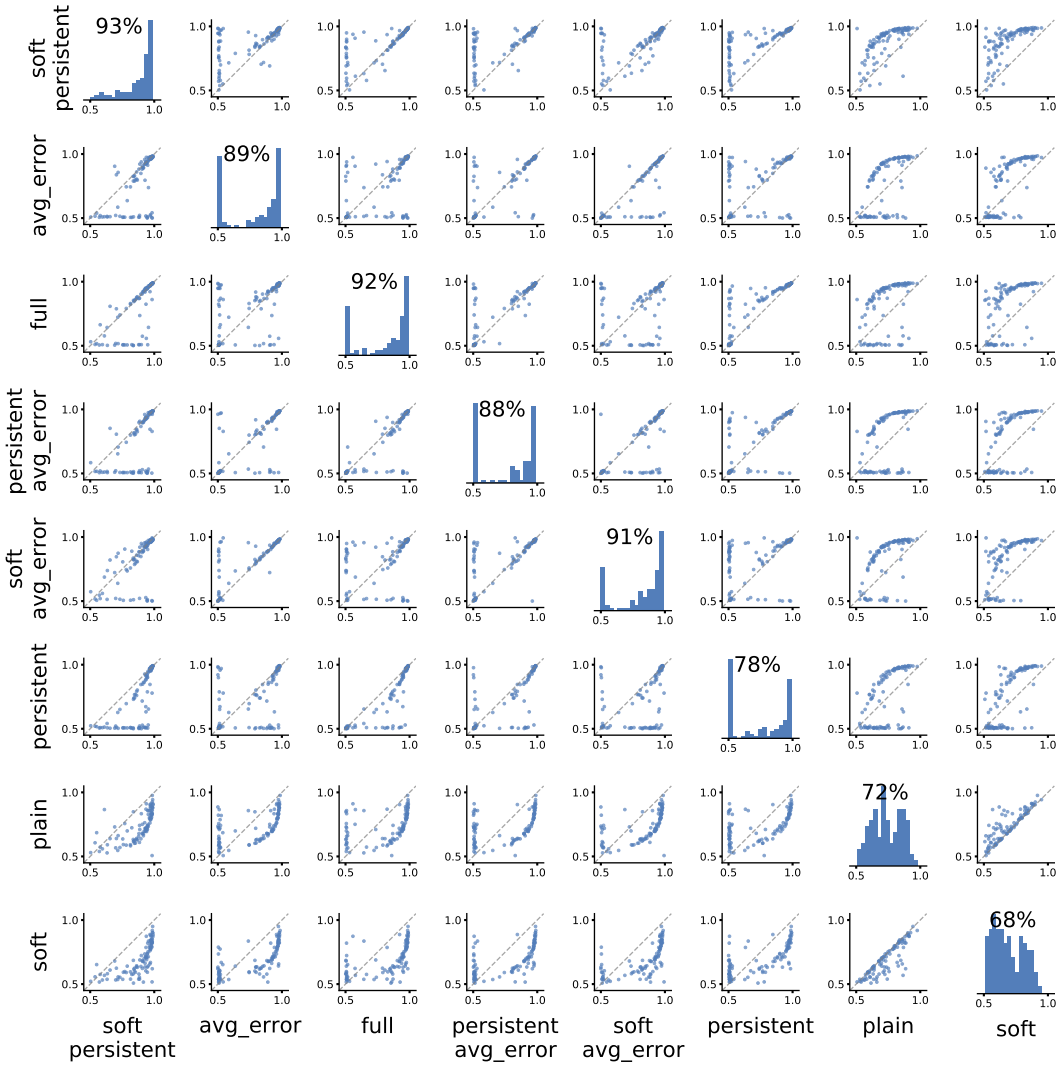

Figure 8: Comparison of segmentation accuracy for all combinations of BioWTA enhancements. The plot at position (i, j), for $i \neq j$ in the figure compares the segmentation accuracy from method i (on the y-axis) to that from method j (on the x-axis). The plots on the diagonal (*i.e.,* when $i = j$) are histograms showing the distribution of the accuracy scores for each method. The number above each histogram is the median segmentation score obtained for that method.

Spectral properties: poles and zeros The signal $y(t)$ and noise $\epsilon(t)$ enter linearly in the definition of an ARMA process, eq. (G.1), with various delays. Because of this, a z-transform is useful for analyzing ARMA processes:

$$Y(z) = \sum_t y(t) z^{-t} \,, \tag{G.3}$$

where z is a complex number. This can be related to the Fourier series (or frequency-space representation) of the signal by focusing on the unit circle, $z = e^{-2\pi i f}$.

The transformation induced by an ARMA process has a simple form after a z-transform:

$$Y(z) = H(z) E(z) \,, \tag{G.4}$$

where the *transfer function* $H(z)$ is given by

$$H(z) = \frac{1 + b_1 z^{-1} + \cdots b_q z^{-q}}{1 - w_1 z^{-1} - \cdots w_p z^{-p}} \equiv \frac{B(z)}{W(z)} \,. \tag{G.5}$$

The transfer function blows up at roots of the denominator $W(z)$, which is why they are also called the "poles" of the system. The magnitude of a pole is related to the temporal extent of the response to a particular excitation; poles outside the unit circle give rise to instabilities in the ARMA process.

The transfer function vanishes at the roots of the numerator $B(z)$, so these are called the "zeros" of the system. Inverting an ARMA process swaps the poles with the zeros, so a system with a stable inverse must have all the zeros contained within the unit circle.

H Cepstral oracle method

General overview A standard control-theory method for fault detection relies on using the inverse of a system with a known transfer function to detect anomalies in the functioning of the system (De Cock, 2002; Boets et al., 2005).

Specifically for our purposes, consider the signal from eq. (2),

$$y(t) = \sum_{k=1}^{M} z_k(t) \Big[\boldsymbol{w}_k^\top \boldsymbol{x}(t) + \epsilon(t) \Big] \,. \tag{H.1}$$

Each of the AR processes defined by the coefficients $\boldsymbol{w}_k$ has an inverse MA process with coefficients $\boldsymbol{b}_k = -\boldsymbol{w}_k$, as shown in eq. (G.2). We can filter the signal $y(t)$ using each of these inverse filters to obtain

$$\epsilon_k(t) = y(t) - \boldsymbol{w}_k^\top \boldsymbol{x}(t) \,. \tag{H.2}$$

Notice that this is nothing else but the prediction error in the "oracle" setting where the model weights $\boldsymbol{w}_k$ are set to their ground-truth values.

If the actual process that generated the sample at time t is $\hat{z}(t)$, then $\epsilon_{\hat{z}(t)} \equiv \epsilon(t)$, *i.e.,* uncorrelated Gaussian noise. In contrast, all other filterings, $\epsilon_k(t)$ for $k \neq \hat{z}(t)$, will still contain temporal correlations.

The cepstral oracle method relies on a measure of the strength of temporal correlations to find the index k that leads to the least temporally correlated filtering $\epsilon_k(t)$. This provides a best guess for the identity of the generating process.

Cepstral norm The specific measure of temporal correlation that we use here is a *cepstral norm* (De Cock and De Moor, 2002; De Cock, 2002; Boets et al., 2005).

The (power) cepstrum is the inverse Fourier transform of the logarithm of the power spectrum of a signal (De Cock, 2002),

$$c(k) = \Big| \mathcal{F}^{-1} \Big(\log |\mathcal{F}(y(t))|^2 \Big) \Big| \,, \tag{H.3}$$

where $\mathcal{F}$ denotes the Fourier transform operator. This has the convenient property that it turns convolutions into sums, thus allowing to separate different stages of filtering if these have different-enough spectral responses.

If we define the *cepstral norm*

$$g(y)^2 = \sum_{k=0}^{\infty} k |c(k)|^2 \,, \tag{H.4}$$

this provides a measure of the distance between the signal $y(t)$ and uncorrelated Gaussian noise (De Cock, 2002; Boets et al., 2005). In practice, only a finite number of cepstral coefficients are used in the calculation.

Calculating the cepstral norm A sequence of samples from the signal $y(t)$ are necessary for calculating the cepstral norm. Applying the definitions (H.3) and (H.4) directly does not lead to the most efficient estimate. Instead, we start by defining "past" and "future" *Hankel matrices*,

$$\begin{aligned} Y_p &= \frac{1}{\sqrt{\tau}} \begin{pmatrix} y(0) & y(1) & \dots & y(\tau-1) \\ y(1) & y(2) & \dots & y(\tau) \\ \vdots & \vdots & \ddots & \vdots \\ y(k-1) & y(k) & \dots & y(k+\tau-2) \end{pmatrix}, \\ Y_f &= \frac{1}{\sqrt{\tau}} \begin{pmatrix} y(k) & y(k+1) & \dots & y(k+\tau-1) \\ y(k+1) & y(k+2) & \dots & y(k+\tau) \\ \vdots & \vdots & \ddots & \vdots \\ y(2k-1) & y(2k) & \dots & y(2k+\tau-2) \end{pmatrix}, \end{aligned} \tag{H.5}$$

where k gives the maximal cepstral order used in the cepstral norm formula, eq. (H.4), and τ gives the number of samples over which we're averaging. We also define a "total" Hankel matrix

$$Y = \begin{pmatrix} Y_p \\ Y_f \end{pmatrix}. \tag{H.6}$$

In terms of the Hankel matrices, the cepstral norm can be approximated by

$$g(y)^2 \approx \log\det Y_p Y_p^T + \log\det Y_f Y_f^T - \log\det Y Y^T, \tag{H.7}$$

which becomes exact in the limit $k \to \infty$, $\tau \to \infty$. Furthermore, the calculation of the determinants can be simplified by using LQ decompositions,

$$Y_p = L_p Q_p, \qquad Y_f = L_f Q_f, \qquad Y = LQ, \tag{H.8}$$

where L are lower-triangular matrices and Q are orthogonal matrices. With these notations, we can write

$$\begin{aligned} g(y)^2 &\approx 2\sum_{j=1}^{k} \left(\log L^p_{jj} + \log L^f_{jj}\right) - 2\sum_{j=1}^{k} \log L_{jj} \\ &= 2\sum_{j=1}^{k} \log L^f_{jj} - 2\sum_{j=k+1}^{2k} \log L_{jj}, \end{aligned} \tag{H.9}$$

Rolling estimate of the cepstral norm In our setting, the generating process changes during the duration of the signal. To find a local estimate of how uncorrelated a filtering $\epsilon_k(t)$ from eq. (H.2) is, we could apply the cepstral norm calculation in sliding window, much like we do when we calculate the segmentation score. A more efficient approach uses a discounting mechanisms akin to an exponential moving average, and can be implemented online. We will not give all the details here, but it relies on a redefinition of the Hankel matrices:

$$\tilde{Y}_{ij}(t) = \gamma^{\tau-j} y(t+i+j) = \gamma^{\tau-j} Y_{ij}(t), \tag{H.10}$$

where the indices are assumed to be zero-based. This discounts older samples by a factor of γ raised to the number of time steps that have passed since those samples were observed.

Whenever a new sample is obtained, a column is appended to the Hankel matrix, and the rest of the elements are discounted by an additional factor of γ,

$$\tilde{Y}_{ij}(t+1) = \gamma^{\tau-j} y(t+i+j+1) = \frac{\gamma^{\tau-j}}{\gamma^{\tau-j-1}} \tilde{Y}_{i,j+1}(t) = \gamma \tilde{Y}_{i,j+1}(t). \tag{H.11}$$

Since for the cepstral norm calculation we are only interested in the L factor of the LQ decomposition of Y (as in eq. (H.9)), we can actually use an algorithm for updating the LQ decomposition based on Givens rotations (Oppenheim et al., 2001) to calculate the effect of appending a column. The effect of multiplying the Hankel matrix by γ is simply to multiply L by the same factor.

Details of these procedure can be found in the implementation available on GitHub, at `https://github.com/ttesileanu/bio-time-series`.