

## MIT Open Access Articles

*Kernel density compression for real-time Bayesian encoding/decoding of unsorted hippocampal spikes*

The MIT Faculty has made this article openly available. **Please share** how this access benefits you. Your story matters.

**Citation:** Sodkomkham, Danaipat, Davide Cilibertib, Matthew A. Wilson, Ken-ichi Fukui, Koichi Moriyama, Masayuki Numao, and Fabian Kloosterman. "Kernel density compression for real-time Bayesian encoding/decoding of unsorted hippocampal spikes." Knowledge-Based Systems 94 (15 February 2016), pp.1-12.

**As Published:** <http://dx.doi.org/10.1016/j.knosys.2015.09.013>

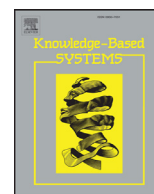
**Publisher:** Elsevier

**Persistent URL:** <http://hdl.handle.net/1721.1/103531>

**Version:** Final published version: final published article, as it appeared in a journal, conference proceedings, or other formally published context

**Terms of use:** Creative Commons Attribution-NonCommercial-NoDerivs License





# Kernel density compression for real-time Bayesian encoding/decoding of unsorted hippocampal spikes



Danaipat Sodkomkham<sup>a,\*</sup>, Davide Ciliberti<sup>b,c</sup>, Matthew A. Wilson<sup>d,e</sup>, Ken-ichi Fukui<sup>a,2</sup>, Koichi Moriyama<sup>a,2</sup>, Masayuki Numao<sup>a,2</sup>, Fabian Kloosterman<sup>b,f,g,1,\*</sup>

<sup>a</sup> Institute of Scientific and Industrial Research, Osaka University, 8-1 Mihogaoka, Ibaraki, Osaka 567-0047, Japan

<sup>b</sup> Neuro-Electronics Research Flanders, Kapeldreef 75, B-3001 Heverlee, Belgium

<sup>c</sup> Research Group of Neurophysiology, Department of Neurosciences, KU Leuven, Leuven, Belgium

<sup>d</sup> Department of Brain and Cognitive Sciences, MIT, Cambridge, USA

<sup>e</sup> Picower Institute for Learning and Memory, MIT, Cambridge, USA

<sup>f</sup> imec, Kapeldreef 75, B-3001 Heverlee, Belgium

<sup>g</sup> Brain & Cognition Research Unit, Faculty of Psychology and Educational Sciences, KU Leuven, Tiensestraat 102 – box 3711, 3000 Leuven, Belgium

## ARTICLE INFO

### Article history:

Received 10 March 2015

Revised 9 September 2015

Accepted 11 September 2015

Available online 14 September 2015

### Keywords:

Kernel density estimation

Data condensation

Online algorithm

Neural decoding

Neural decoding of unsorted spikes

## ABSTRACT

To gain a better understanding of how neural ensembles communicate and process information, neural decoding algorithms are used to extract information encoded in their spiking activity. Bayesian decoding is one of the most used neural population decoding approaches to extract information from the ensemble spiking activity of rat hippocampal neurons. Recently it has been shown how Bayesian decoding can be implemented without the intermediate step of sorting spike waveforms into groups of single units. Here we extend the approach in order to make it suitable for online encoding/decoding scenarios that require real-time decoding such as brain-machine interfaces. We propose an online algorithm for the Bayesian decoding that reduces the time required for decoding neural populations, resulting in a real-time capable decoding framework. More specifically, we improve the speed of the probability density estimation step, which is the most essential and the most expensive computation of the spike-sorting-less decoding process, by developing a kernel density compression algorithm. In contrary to existing online kernel compression techniques, rather than optimizing for the minimum estimation error caused by kernels compression, the proposed method compresses kernels on the basis of the distance between the merging component and its most similar neighbor. Thus, without costly optimization, the proposed method has very low compression latency with a small and manageable estimation error. In addition, the proposed bandwidth matching method for Gaussian kernels merging has an interesting mathematical property whereby optimization in the estimation of the probability density function can be performed efficiently, resulting in a faster decoding speed. We successfully applied the proposed kernel compression algorithm to the Bayesian decoding framework to reconstruct positions of a freely moving rat from hippocampal unsorted spikes, with significant improvements in the decoding speed and acceptable decoding error.

© 2015 The Authors. Published by Elsevier B.V.

This is an open access article under the CC BY-NC-ND license (<http://creativecommons.org/licenses/by-nc-nd/4.0/>).

## 1. Introduction

Neural encoders and decoders are commonly used to study the relation between behavioral or sensory covariates and neural re-

sponses. Statistical inferences have played an important role in many encoding/decoding frameworks, e.g. [1–6]. Generally, the encoding model captures necessary properties from the recorded neural activities and constructs a model that maps to the observed behaviors or stimuli. The decoding model then employs the constructed relation to infer behaviors or stimuli based on the observed neural activity. For example, neural signals recorded from the action potentials (spikes) of pyramidal neurons in the CA1 region of the rodent hippocampus contain information that is correlated to spatial behaviors of the animal [7]. These cells are also known as place cells because spiking activities of certain place cells become more active when an animal is in

\* Corresponding authors at: NERF, Kapeldreef 75, 3001 Leuven, Belgium. Tel.: +32 (0)16 283514 (F. Kloosterman), Tel.: +81 6 6879 8426 (D. Sodkomkham).

E-mail addresses: [danaipat@ai.sanken.osaka-u.ac.jp](mailto:danaipat@ai.sanken.osaka-u.ac.jp) (D. Sodkomkham), [fabian.kloosterman@nerf.be](mailto:fabian.kloosterman@nerf.be) (F. Kloosterman).

<sup>1</sup> Tel.: +32 1628 1211.

<sup>2</sup> Tel.: +81 6 6879 8426.

a certain location [8]. In other words, the temporal patterns of spikes from different place cells are spatially tuned to different locations.

Most of existing neural encoders/decoders require sorted spikes to operate [9,10,11–18,2] (see [19] for a review). That is spiking activity of each single neuron has to be isolated from others and separated from background electrical noise before being handed over to the encoding/decoding model. This prerequisite step is called “spike sorting”. Many works have been contributed to the developing of fast and reliable spike sorting algorithms [20]. However, a study has shown that classification errors of assigning spikes to incorrect unit have various impact to information capacity of the resulting sorted spikes [21]. In addition, the objective of spike sorting to isolate and identify the cell that originated each spike is rather different from the goal of neural decoding which is to minimize the decoding error. Unclassified spikes during the sorting in attempt to minimize sorting errors could still convey information that can be extracted by the encoder/decoder.

To avoid the possibility of information loss and accumulation of errors from spike sorting, Bayesian encoding/decoding framework proposed in [22] has introduced a method to create a direct mapping between spike waveform features and the covariates of interest without a prerequisite step of spike sorting. The name “Bayesian” comes from the adoption of a statistical inference that utilizes Bayes’ theorem. More specifically, the decoding is obtained by the maximum posterior probability in Eq. (1), where the covariates are spatial behaviors of the animal, e.g. positions or head directions.

$$p(\text{covariates}|\text{spikes}) = p(x|s) = \frac{p(s|x)p(x)}{p(s)} \propto \frac{p(s|x)p(x)}{p(s|x)p(x)} \quad (1)$$

Outline of the Bayesian encoding/decoding framework [22] is illustrated in Fig. 1. The first stage (A) detects and extracts spike waveforms from extracellularly recorded multiunit activity from CA1 region of a freely moving rat in an open field. Next (B), waveform features, such as amplitudes, are extracted. At the same time (C), position of the animal is tracked using a video camera and forwarded together with the waveform features to the next stage (D), where the probability models  $p(s,x)$  and  $\pi(x)$  are modeled.

During the decoding phase (e), a sequence of spikes is partitioned into bins. For each decoding bin, the posterior probability is computed and the behavior is decoded. The likelihood  $p(s|x)$  of the stimulus  $x$  given a set of spike features  $s$  models the relation between spiking patterns (modulation of spike amplitudes and firing rates) and behaviors by assuming spatiotemporal Poisson statistics as follows:

$$p(s|x) = \Delta t^n \left[ \prod_{i=1}^n \lambda(s_i, x) \right] e^{-\Delta t \lambda(x)}, \quad (2)$$

where the decoding bin containing  $n$  spikes has a size of  $\Delta t$  time interval. Rate parameter  $\lambda(s_i, x)$  which is the fraction of the occurrences of certain spike features  $s_i$  coinciding with certain stimulus  $x$  divided by the total time stimulus  $x$  (*occupancy*( $x$ )) is presented as follows:

$$\lambda(s_i, x) = \frac{\text{spikecount}(s_i, x)}{\text{occupancy}(x)} = \frac{N}{T} \frac{p(s_i, x)}{\pi(x)} = \mu \frac{p(s_i, x)}{\pi(x)}, \quad (3)$$

where  $N$  is the total number of spikes and  $T$  is the total time from all the decoding bins.  $p(s_i, x)$  is the joint probability distribution of

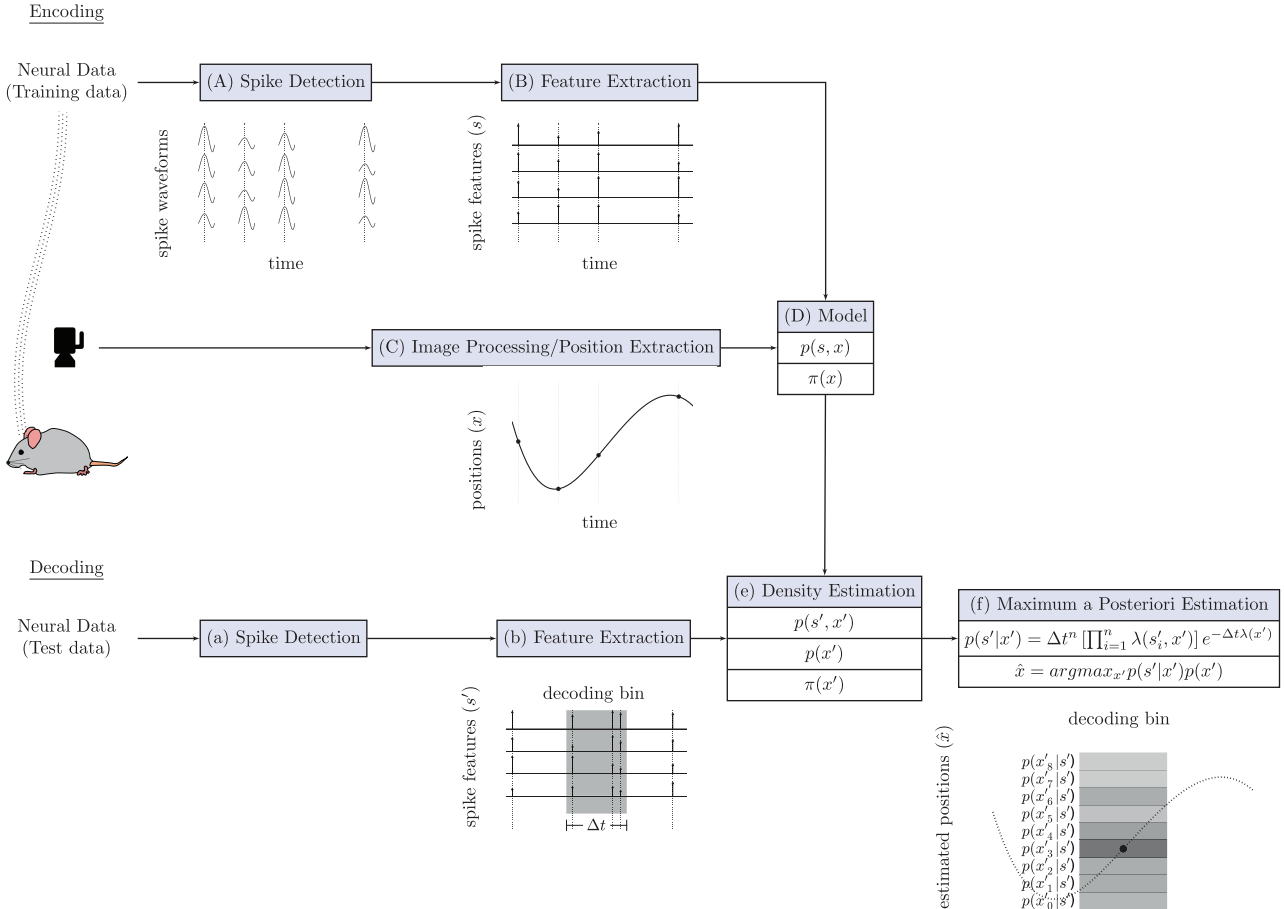


Fig. 1. Bayesian decoding using unsorted spikes in the rat hippocampus.

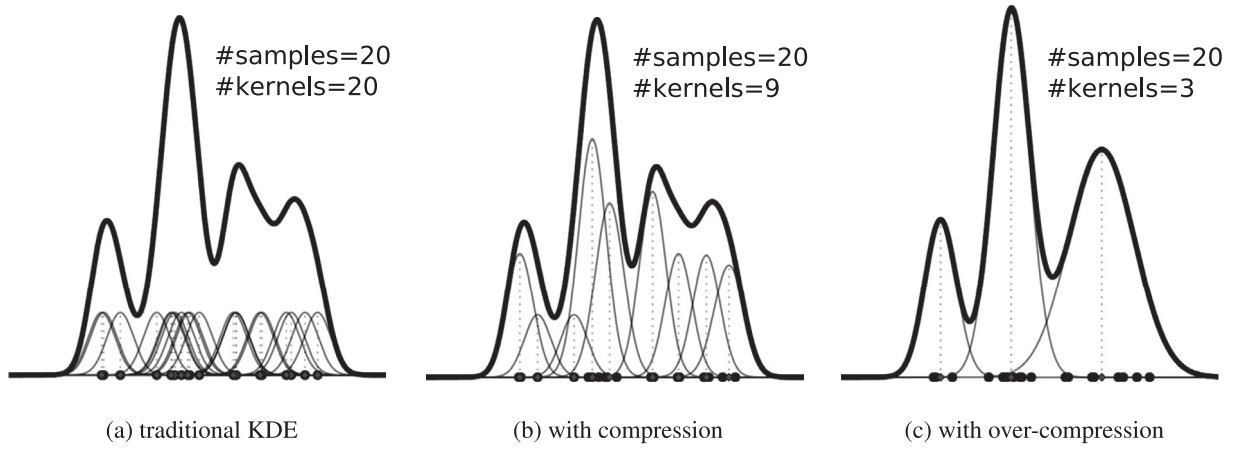


Fig. 2. Kernel compression.

finding spike features  $s_i$  that occur simultaneously when the animal experiences stimulus  $x$ . The probability distribution  $\pi(x)$  is the probability of finding the animal experiencing stimulus  $x$ . Similarly,

$$\lambda(x) = \frac{\text{spikecount}(x)}{\text{occupancy}(x)} = \frac{N}{T} \frac{p(x)}{\pi(x)} = \mu \frac{p(x)}{\pi(x)}, \quad (4)$$

with the exception that  $\text{spikecount}(x)$  counts all the spikes that occur during the experience of stimulus  $x$  regardless of the spike features.

In order to implement a real-time Bayesian decoding framework over streaming hippocampal spikes, it is crucial to find out the major bottlenecks in terms of computational time. Due to the high dimensionality of the joint probability density  $p(s_i, x)$ , it is often impractical to pre-compute it (see [22]), thus requiring it to be evaluated on-the-fly using kernel density estimation (KDE). The most time consuming step is therefore the computation of the probability densities  $p(s_i, x)$ ,  $p(x)$  and  $\pi(x)$ , which can be estimated using kernel density estimation (KDE). However, time complexity of KDE is approximately quadratic, which is extremely expensive, often requires excessive time and impractical for our application. Moreover, traditional KDE does not scale well to the unbounded streams of data, which is the nature of all real-time applications.

Thus we propose a kernel compression algorithm that aims to speedup the density estimation task in order to achieve real-time decoding. The proposed method achieves faster density estimation by replacing redundant kernel components with mixtures of merged components, resulting in a reduced number of kernel components; thus, the density evaluation time is reduced. Consider the example of traditional KDE in Fig. 2(a), where 20 samples were drawn from an unobservable density function. A Gaussian kernel is centered at each sample. The density estimate is then obtained by averaging the densities contributed by these 20 Gaussian kernels. With the compression, redundant (or closely placed) kernels are properly replaced by a merged kernel with larger weight. In Fig. 2(b), the number of kernels required for density estimation can be reduced by half while the density estimate remains nearly identical to the result from traditional KDE. Fewer kernels enable faster estimation; however, it is worth noting that excessive compression can lead to less accurate estimation, as shown in Fig. 2(c).

The remainder of this paper is organized as follows. A brief introduction to the density estimation problem and KDE are given in Section 2. In Section 3, the proposed kernel compression algorithm, kernel merging methods and density evaluation method are discussed. Performance evaluations and comparisons are discussed in Section 4. Next, advantages and disadvantages of the proposed method are

highlighted and discussed in Section 5. Finally, conclusions are provided in Section 6.

## 2. Kernel density estimation

KDE is a nonparametric method for approximating an unknown probability density function  $p(\mathbf{x})$  from observations. The term kernel refers to any probability density function that is placed over each observation to quantify the likelihood of a small region around each observation containing  $\mathbf{x}$ . Thus, the estimated density is the weighted summation of contributions that each observation makes to the distribution.

### 2.1. Traditional kernel density estimation

By definition, any kernel function  $K(u) : \mathbb{R} \mapsto \mathbb{R}_{\geq 0}$  must be symmetric, i.e.  $\forall u \in \mathbb{R} : K(-u) = K(u)$ , and satisfy the following condition:

$$\int_{-\infty}^{\infty} K(u) du = 1. \quad (5)$$

Given a set of observations  $\{\mathbf{X}_1, \dots, \mathbf{X}_N\}, \forall \mathbf{X}_i \in \mathbb{R}^d$ , the estimated probability density function  $p(\mathbf{x}), \mathbf{x} \in \mathbb{R}^d$  is as follows:

$$p(\mathbf{x}) = \frac{1}{Nh^d} \sum_{i=1}^N K\left(\frac{\mathbf{X}_i - \mathbf{x}}{h}\right), \quad (6)$$

where  $h$  is a parameter used to define the width of each kernel. This parameter is often referred to as *bandwidth*. For instance, a commonly used Gaussian kernel has the following density function:

$$p(\mathbf{x}) = \frac{1}{N} \sum_{i=1}^N \frac{1}{h^d \sqrt{(2\pi)^d}} \exp\left(-\frac{1}{2} \frac{(\mathbf{X}_i - \mathbf{x})^T (\mathbf{X}_i - \mathbf{x})}{h^2}\right). \quad (7)$$

In general cases, the bandwidth parameter can be represented by a vector  $\mathbf{h} = [h_1, h_2, \dots, h_d]$  of  $d$  dimensions, in which each element individually represents the bandwidth parameter of each dimension. Thus, the density estimation function in Eq. (7) can be generalized as follows:

$$p(\mathbf{x}) = \frac{1}{N} \sum_{i=1}^N \frac{1}{\prod_{j=1}^d h_j \sqrt{(2\pi)^d}} \exp\left(-\frac{1}{2} \left(\frac{\mathbf{X}_i - \mathbf{x}}{\mathbf{h}}\right)^T \left(\frac{\mathbf{X}_i - \mathbf{x}}{\mathbf{h}}\right)\right). \quad (8)$$

### 2.2. Problem of KDE with online data

Kernel density estimation is a costly operation. The computation in Eq. (6) involves  $N$  iterations over all observations. Thus, a

simple looping density estimation of  $M$  query points has a time complexity of  $\mathcal{O}(MN)$  or  $\mathcal{O}(N^2)$ , where  $M \approx N$ . Although state-of-the-art dual-tree based KDE [23], which is known to be the fastest and most accurate algorithm, can scale the computation to  $\mathcal{O}(N)$ , it requires  $\mathcal{O}(N \log N)$  time for construction (building tree structures). However, the dual-tree approach was not designed as an incremental algorithm. Namely, it is inefficient in an online setting, where the tree-based data structure requires reconstruction every time a new observation becomes available. So are the recently proposed distributed and parallel KDE [24] that was designed specially for very large datasets. The distributed KDE employed sub-sampling techniques to reduce size of KDE model and speedup the density estimation. However, the algorithm proposed in [24] was not designed to be incremental, namely a new data point cannot be directly inserted to the built model without rebuilding the entire model. Because each update takes  $\mathcal{O}(N)$ , the sub-sampling KDE was obviously more suitable for batch processing than online processing, where  $N$  could grow unboundedly.

Reducing the number of computations required to compute a point estimate of the density has been a main interest in optimizing KDE. A number of improvements have been considered to reduce the number of kernel components. For example, a straightforward binning method has been introduced in [25,26], where the density is estimated from a sum of densities estimated from each bin weighted by the number of samples placed in each bin. In [27], a method that estimates the density with the convolution theorem using Fourier transformation, in which binning of data is also required, has been proposed. However, this requires the bin size for each dimension to be specified; therefore, such approaches quickly become inefficient for high-dimensional multivariate data streams.

As an alternative to binning, in [28–33], samples were clustered and reduced by replacing them with cluster centroids. The main problem with the cluster-based approaches is the computational burden of the optimization required for data partitioning and the solution's dependency on the initial condition. In addition, cluster-based methods, including a self organizing map-based KDE [34], often update their models with a small batch of buffered data (often a few hundreds). Hence, the models are not truly updated with each new sample, because they have to wait until the buffers get filled so that the models can call clustering procedure to update their density estimators with the new samples.

Some methods have been adapted to handle streaming data by reducing the data condensation overhead so that the model can keep pace with the input stream. The M-kernel merging technique [35] is an online density estimator that can handle a univariate data stream by limiting the number of kernel components and substituting redundant components with a representative component (kernel merging). The model invokes a merging routine whenever its buffer is full. Then, downhill simplex optimization is employed to find the best way to replace two components with a representative component that would minimize the merging cost, which is the absolute error ( $L_1$ -norm) between the estimation and the underlying density. Since its kernel merging strategy is based on numerical optimization, the time that the algorithm requires to update the model to a new sample can be high. As a solution, Heinz and Seeger [36] has introduced an M-kernel based online KDE with a constant time pairwise merging technique to solve the problem of high update cost of the M-kernel algorithm. However, the M-kernel and other M-kernel-based approaches [36–38] cannot be generalized to support multivariate data streams without losing speed because the algorithms rely on sortedness which is more complicated to achieved for multivariate data. The lack of total ordering in multidimensional data forces the algorithms to use nearest neighbors search, which would incur additional computational complexity [37].

### 2.3. Idea of the proposed method

In this paper, we addressed the problem of finding an optimal merging pair from a different perspective. Rather than optimizing for the optimal merging components pair that is expected to minimize the overall merging cost, we skip such optimization and search for the best merging pair by their distance to reduce the time required to process each new sample. In particular, a newly arriving component will be merged with the nearest kernel component only if the nearest kernel exists within a specified distance threshold  $\tau$ . Otherwise, the new component will be inserted into the model without merging. The parameter  $\tau$  is used to limit the scope for neighbor search and can be used to control the compression ratio. Additionally, validity of the M-kernel based approaches is limited to one-dimensional data only. Generalization of the M-kernel to higher dimensional data would impact the compression overhead. In other words, the time required for the query of the optimal merging pair would be quadratic in stead of linear to the buffer size. On the contrary, the time required by our approach to search for the nearest component is always linear.

A simulation wherein 1,000 pairs of 2D multivariate Gaussian components in which different weights, widths and centers were randomly assigned shows a result that supports our approach. The result shown in Fig. 3 indicates a clear relationship between the resulting merging cost and the distance between the two merging components; smaller distances, to some extent, tend to give smaller errors. Note that when the distances are greater than 10, this presumption is less likely to be true. Thus, we introduce a distance threshold parameter  $\tau$  that limits the greatest distance allowed for merging to avoid the high probability of merging non-cost-optimized pairs at greater distances. However, note that it is difficult to find nearby neighbor kernels to merge with when the search radius (distance threshold  $\tau$ ) is small, which results in a less compressed model. Thus, the trade-off between the additional error caused by merging non-optimal pairs and the compactness of the resulting model can be managed by the  $\tau$  parameter.

## 3. Proposed online kernel density estimation

### 3.1. Kernel compression algorithm

In an online setting, where observations are made sequentially and data samples arrive in order, the compression algorithm is required to process and compress new samples faster than the rate

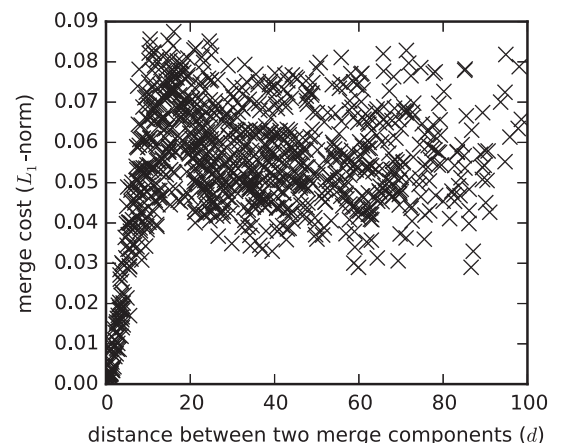


Fig. 3. Relationship between distance and merging cost.



of the input stream so that the model is always updated and ready for the density estimation at any time. The process for handling new observation (training) stream is illustrated by a flowchart in Fig. 4. Start off with an empty model  $G$ , as a new data sample  $\mathbf{x}_i$  arrives, a Gaussian kernel is assigned with  $\mathbf{x}_i$  as the center with a predefined diagonal covariance matrix  $\Sigma$ . Note that  $\Sigma = \text{diag}(\mathbf{h}^2)$ , where  $\mathbf{h}^2 = [h_1^2, h_2^2, \dots, h_d^2]$ . The choice of the covariance matrix  $\Sigma$  determines the final smoothness of the density. Moreover, larger initial kernels will result in more highly compressed density. Each new component is weighted uniformly with a weight coefficient  $w$  of 1. The algorithm then looks for an opportunity to merge this new component with an existing component in the mixture model in order to maintain the number of components. The algorithm performs a search for the most proper component to merge the new sample with in order to minimize compression error. Search for the proper component is done by a nearest neighbor search that considers the Mahalanobis distance between two components. In other words, it attempts to find a Gaussian component  $\hat{g}$  in a set of Gaussian mixture models  $G$  that minimizes the Mahalanobis distance  $D_{\hat{g}}(\mathbf{x}_i)$  as follows:

$$\hat{g} = \underset{g_j \in G}{\operatorname{argmin}} D_{g_j}(\mathbf{x}_i), \quad (9)$$

where

$$g_j = \mathcal{N}(w_j, \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j) \quad (10)$$

and

$$D_{g_j}(\mathbf{x}_i) = \sqrt{(\mathbf{x}_i - \boldsymbol{\mu}_j)^T \boldsymbol{\Sigma}_j^{-1} (\mathbf{x}_i - \boldsymbol{\mu}_j)}. \quad (11)$$

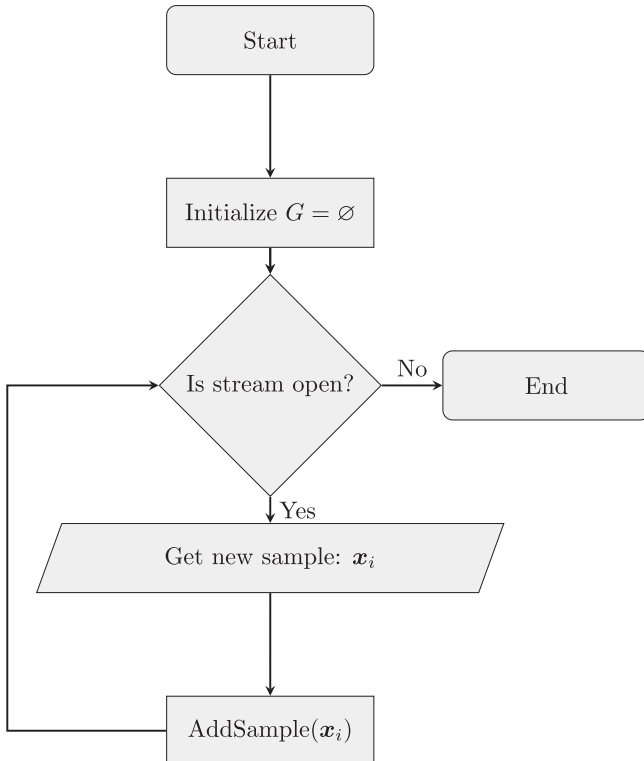


Fig. 4. Training data stream handler.

#### Algorithm 1. Adding new sample

---

```

1: function ADDSAMPLE ( $\mathbf{x}_i$ )
2:    $g' \leftarrow \mathcal{N}(1, \mathbf{x}_i, \Sigma)$ 
3:   if  $G$  is empty then
4:      $G.\text{insert}(g')$ 
5:   else
6:      $\hat{g} \leftarrow \underset{g_j \in G}{\operatorname{argmin}} D_{g_j}(\mathbf{x}_i)$ 
7:     if  $D_{\hat{g}}(\mathbf{x}_i) \leq \text{distance\_threshold}$  then
8:        $G.\text{remove}(\hat{g})$ 
9:        $g' \leftarrow \text{Merge}(g', \hat{g})$ 
10:       $G.\text{insert}(g')$ 
11:     else
12:        $G.\text{insert}(g')$ 
13:     end if
14:   end if
15: end function

```

---

The Mahalanobis distance  $D_{\hat{g}}$  is then used to determine whether the newly arrived data sample should be merged with the nearest component. Only the nearest component with a specified *distance threshold* is merged. Otherwise, a new Gaussian kernel is centered on the new data sample with a predefined covariance matrix  $\Sigma$ , thereby resulting in a new component. The outline of the algorithm for adding a new sample is given in Algorithm 1. We further discuss how the kernel merging routine (line 9) is handled in the next subsection.

#### 3.2. Gaussian kernel merging

Goal of kernel merging is to substitute a mixture of two Gaussian components with a single Gaussian component that approximates the mixture best. The M-kernel merging [35], an alternative solution for Gaussian kernel merging that we have introduced in Section 2, approximated mean and bandwidth of the merged component using downhill simplex optimization method on the function of accuracy lost in the kernel merging. However, the optimization becomes less efficient when applying the M-kernel technique to higher dimensional data. Therefore, rather than optimizing for the mean vector and the covariance matrix that would minimize the accuracy lost from kernel merging, we opt for the moment-preserving merge, which can be approximated in almost constant time ( $\mathcal{O}(1)$ ) given that the number of dimensions is small.

##### 3.2.1. Full covariance match

A mixture of two Gaussian components  $\mathcal{N}(w_1, \boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)$  and  $\mathcal{N}(w_2, \boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2)$  can be approximated by a single representative Gaussian component by matching the zeroth, first, and second moments to the mixture. Namely, weight  $w$ , mean vector  $\boldsymbol{\mu}$  and covariance matrix  $\boldsymbol{\Sigma}$  of the resulting component  $\mathcal{N}(w, \boldsymbol{\mu}, \boldsymbol{\Sigma})$  can be approximated by the moment matching method as follow.

$$w = w_1 + w_2. \quad (12)$$

$$\boldsymbol{\mu} = w^{-1} (w_1 \boldsymbol{\mu}_1 + w_2 \boldsymbol{\mu}_2). \quad (13)$$

$$\begin{aligned}
 \boldsymbol{\Sigma} &= w^{-1} (w_1 (\boldsymbol{\Sigma}_1 + (\boldsymbol{\mu}_1 - \boldsymbol{\mu})(\boldsymbol{\mu}_1 - \boldsymbol{\mu})^T) \\
 &\quad + w_2 (\boldsymbol{\Sigma}_2 + (\boldsymbol{\mu}_2 - \boldsymbol{\mu})(\boldsymbol{\mu}_2 - \boldsymbol{\mu})^T)) \\
 &= w^{-1} (w_1 \boldsymbol{\Sigma}_1 + w_2 \boldsymbol{\Sigma}_2 + w_1 \boldsymbol{\mu}_1 \boldsymbol{\mu}_1^T + w_2 \boldsymbol{\mu}_2 \boldsymbol{\mu}_2^T) - \boldsymbol{\mu} \boldsymbol{\mu}^T \\
 &= w^{-1} \sum_{i=1}^2 w_i (\boldsymbol{\Sigma}_i + \boldsymbol{\mu}_i \boldsymbol{\mu}_i^T) - \boldsymbol{\mu} \boldsymbol{\mu}^T.
 \end{aligned} \quad (14)$$

Although the moment-preserving merge described above was proven to give a merged component that has a minimal Kullback-Leiber discrimination from the mixture (see Theorem 3.2 in [39]), the resulting covariance matrix is not always diagonal in which many computational optimizations can take advantage of its mathematical properties (see Section 3.3 for further details). Therefore, we propose another kernel merging method for Gaussian components called the *Bandwidth match*, wherein the merge happens on the bandwidth vector of a Gaussian kernel and the merged covariance matrix is reconstructed from the merged bandwidth. To put it differently, the merge only considers diagonal elements of the covariance matrices. Hence, the resulting covariance matrix is guaranteed to be a diagonal matrix.

### 3.2.2. Bandwidth match

Suppose we are given a  $d$ -dimensional Gaussian kernel with the bandwidth  $\mathbf{h} = [h_1, h_2, \dots, h_d]$ . By definition, the corresponding  $d \times d$  covariance matrix  $\Sigma$  of the Gaussian kernel can be constructed as follows.

$$\Sigma \equiv \begin{bmatrix} h_1^2 & 0 & \dots & 0 \\ 0 & h_2^2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & h_d^2 \end{bmatrix}. \quad (15)$$

In other words,

$$\begin{aligned} \Sigma &\equiv \text{diag}([h_1^2, h_2^2, \dots, h_d^2]) \\ &= \text{diag}(\mathbf{h}^2). \end{aligned} \quad (16)$$

Referring to the moment-preserving merge, we modified Eq. (14) to only match diagonal elements of the covariance matrix, ignoring all other elements as follow.

$$\begin{aligned} \mathbf{h}^2 &= w^{-1} (w_1 (\mathbf{h}_1^2 + (\boldsymbol{\mu}_1 - \boldsymbol{\mu})^2) + w_2 (\mathbf{h}_2^2 + (\boldsymbol{\mu}_2 - \boldsymbol{\mu})^2)) \\ &= w^{-1} (w_1 \mathbf{h}_1^2 + w_2 \mathbf{h}_2^2 + w_1 \boldsymbol{\mu}_1^2 + w_2 \boldsymbol{\mu}_2^2) - \boldsymbol{\mu}^2 \\ &= w^{-1} \sum_{i=1}^2 w_i (\mathbf{h}_i^2 + \boldsymbol{\mu}_i^2) - \boldsymbol{\mu}^2. \end{aligned} \quad (17)$$

If needed, the resulting bandwidth  $\mathbf{h}^2$  can also always be used to reconstruct a diagonal covariance matrix  $\Sigma$  using Eq. (16).

It is worth noting that, in contrast to the full covariance match, the proposed bandwidth match method does not require matrix multiplication, which takes  $\Theta(d^2)$  to compute for each merge. Instead,  $\mathbf{h}^2$  only requires  $\Theta(d)$  time complexity to compute. Hence, kernel merging of high-dimensional data could perform significantly faster with the bandwidth match method.

Differences among the two merging methods for one-dimensional (1D) and two-dimensional (2D) kernels are visualized in Fig. 5.

### 3.3. Efficient density evaluation

Suppose we are given a mixture of Gaussian components, probability density function of the mixture containing  $C$  components can be expressed as follows:

$$p(\mathbf{x}) = \sum_{i=1}^C w_i \phi_i(\mathbf{x}), \quad (18)$$

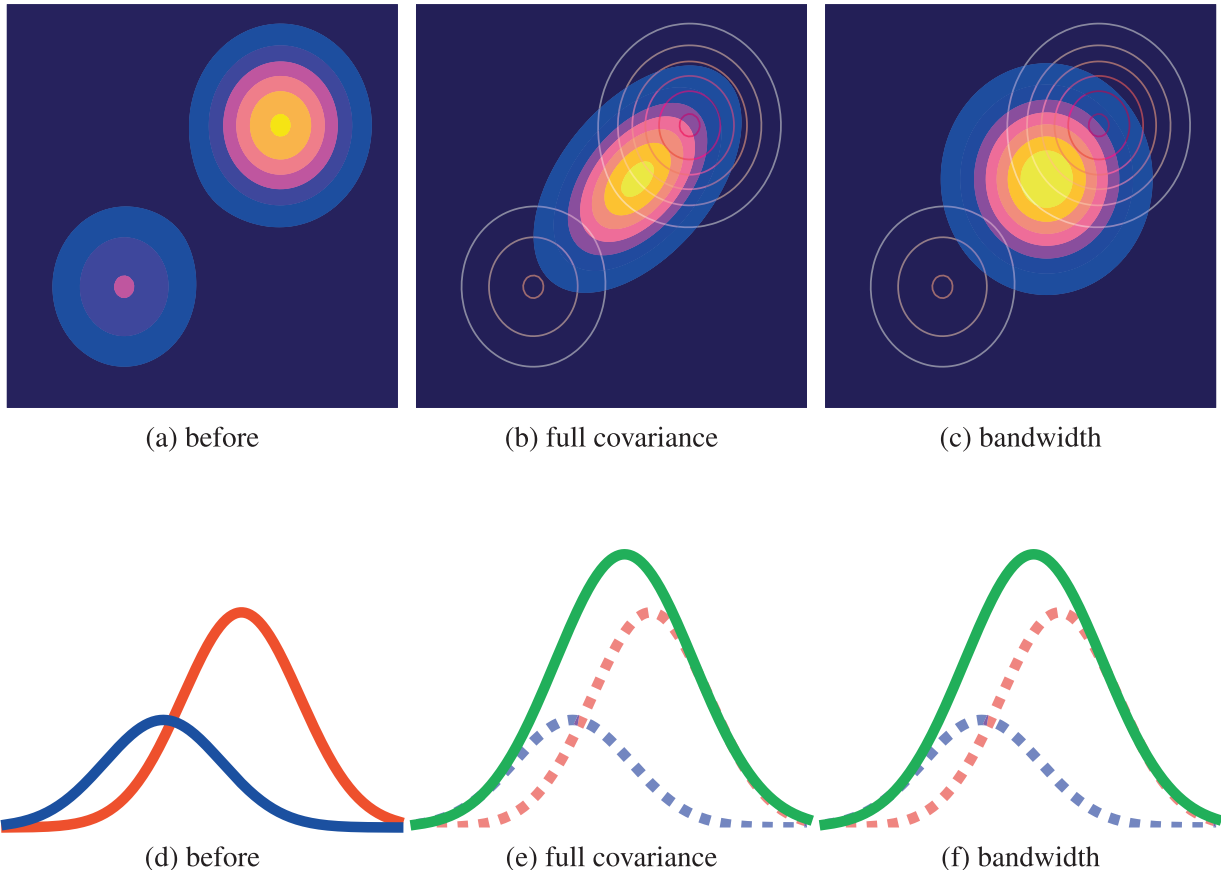


Fig. 5. Kernel merging methods.

where

$$\phi_i(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^d |\Sigma_i|}} \exp\left(-\frac{1}{2}(\mathbf{x} - \mu_i)^T \Sigma_i^{-1}(\mathbf{x} - \mu_i)\right). \quad (19)$$

An interesting characteristic of the bandwidth match merging method is that the covariance matrices of all components remain diagonal after compression. Therefore, we can rewrite Eq. (19) to

$$\phi_i(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^d \prod_{j=1}^d h_{i,j}^2}} \exp\left(-\frac{1}{2} \sum_{j=1}^d \frac{(x_j - \mu_{i,j})^2}{h_{i,j}^2}\right), \quad (20)$$

where we can set a *cut-off threshold* to skip the density evaluation of certain components  $\mu_i$  when they are too distant, which assumes that the evaluating point  $\mathbf{x}$  is less likely to be drawn from such distributions. Thus, the very low density contributions from distant components are omitted to obtain a computational speedup. The cut-off optimization is implemented while looping the sum through  $d$  dimensions in the exponent part of Eq. (20). That is, if the normalized sum of the squared distance at any point reaches the threshold, then the evaluation of  $\phi_i(\mathbf{x})$  is stopped, the result of  $\phi_i(\mathbf{x})$  is dismissed and the procedure moves on to the next component  $\phi_{i+1}(\mathbf{x})$ . Note that the cut-off can occur earlier if the dimensions are sorted such that dimensions with higher variance are considered first. This optimization significantly reduces evaluation time, particularly for high-dimensional data.

#### 4. Simulations and applications to experimental data

We have separated this section into three parts to cover (1) the trade-off between speed and accuracy, (2) performance evaluation on high-dimensional data streams and (3) performance evaluation on the real-time decoding of the rat hippocampal spikes.

For the application to experimental data, we used a dataset in which hippocampal spiking data was recorded from a freely behaving rat. The rat was surgically implanted with a multi-tetrode array under isoflurane anesthesia [43,44]. After recovery, the rat explored a 1.8 m diameter open field with 50 cm high black walls and a single orienting white cue. Spike waveforms on 9 implanted tetrodes were detected, digitized and stored for offline analysis. The position of the rat was monitored using an overhead video tracking system. The experiment was conducted under protocols reviewed and approved by the Massachusetts Institute of Technology Committee on Animal Care and followed the guidelines of the National Institutes of Health.

##### 4.1. Trade-off between speed and accuracy

We tested the proposed kernel compression algorithm on a data stream of neural and behavioral recordings as the rat was freely exploring the open field. Each sample of the spike features stream contains 4 values (spike peak amplitude from 4 separated electrode in a tetrode), and the data stream of the animal positions contains 2 values (xy-coordinates). Bandwidth parameters used in all the tests were chosen from grid-search with cross-validation such that the median of the decoding error from the encoder/decoder that implemented uncompressed KDE is minimized. To give an overview image of how the distance threshold  $\tau$  has an effect to the compression, we streamed the first 10,000 samples from the 4-dimensional data stream of spike features to the proposed compression algorithm. As displayed in Fig. 6(a), the larger the distance threshold  $\tau$ , the more the number of KDE components were reduced. As a result, the time required to handle each new sample was also reduced greatly, as shown in Fig. 6(b). After the compression, we compared the estimation accuracy and the estimation speed of the compressed model with a

traditional KDE model which was built from the same set of dataset. The densities estimated from the traditional KDE (without compression) were used as a ground truth. The accuracy of the compressed model was evaluated by the average of relative errors which is defined as follows:

$$\text{relative error}_i = \frac{|d_i^{\text{compressed}} - d_i^{\text{KDE}}|}{d_i^{\text{KDE}}}, \quad (21)$$

where  $d_i^{\text{compressed}}$  is the density at the evaluation point  $i$  estimated from the compressed KDE and  $d_i^{\text{KDE}}$  is the true density at  $i$  estimated from the uncompressed KDE. The averaged errors were computed from the density estimations of other 10,000 data points drawn from the spike features stream. Accuracy and speed of the density estimation of the proposed method with two different kernel merging methods are shown in Fig. 6(c). From the result, as we increased the distance threshold parameter  $\tau$ , the times required to estimate all 10,000 evaluation points were reduced quickly. At the same time, the accuracy loss from kernels merging started to increase as we raised  $\tau$  higher until the errors were almost stable when  $\tau$  were high enough to merge every new sample, resulting in a compressed KDE with only one component. When applying the compression to experimental data traditional KDE took 2.5 s to finish density estimations of 10,000 evaluation points. At  $\tau = 2$ , the proposed methods only took 0.15 s to finish the same task. That is we sped up the density estimation by almost 17 folds, whereas the average estimation errors were raised only by 15% from the full covariance match and bandwidth match methods.

##### 4.2. Performance evaluation on high-dimensional data streams

To emphasize the advantages of the proposed bandwidth match method over other kernel compression techniques on high dimensional data streams, we compared compression speeds of the proposed methods and the cluster-based *Online Discriminative Kernel Density Estimator with Gaussian kernels* [31] on a synthesized dataset of high dimensional data streams of 1,000 uniformly random numbers. In this simulation, all compression algorithms were set to compress 1,000 original samples to about 500 samples (50% compression ratio). For the cluster-based online KDE, an existing cluster is updated every time 100 new samples are added (buffer size = 100). Visualization in Fig. 7(a) and Fig. 7(b) clearly show that our kernel merging approaches outperformed the cluster-based online KDE in both speed and accuracy tests. From the result, it is apparent that the number of dimensions has significant impact on the performance of the cluster-based approach, while our choices of kernel merging techniques suffered less from high dimensionality, especially for the proposed bandwidth matching method.

##### 4.3. Performance evaluation on the real-time decoding of hippocampal spikes

In this final test, we integrated the proposed kernel compression algorithm into the Bayesian decoding framework [22] and applied it to the experimental dataset. To simulate the real-time encoding/decoding contingencies, we set a non-overlapping sliding window to read in small batches from data streams of both spike features and the animal's positions at a time. Size of the sliding window was set to 250 ms, which is an acceptable bin size for decoding hippocampal unsorted spikes during the run periods of the rat [22]. The real-time decoding framework implemented in this test was designed to alternate between the decoding and encoding steps. Referring to the block diagram presented in Fig. 8, when the sliding window moves on to the next batch of



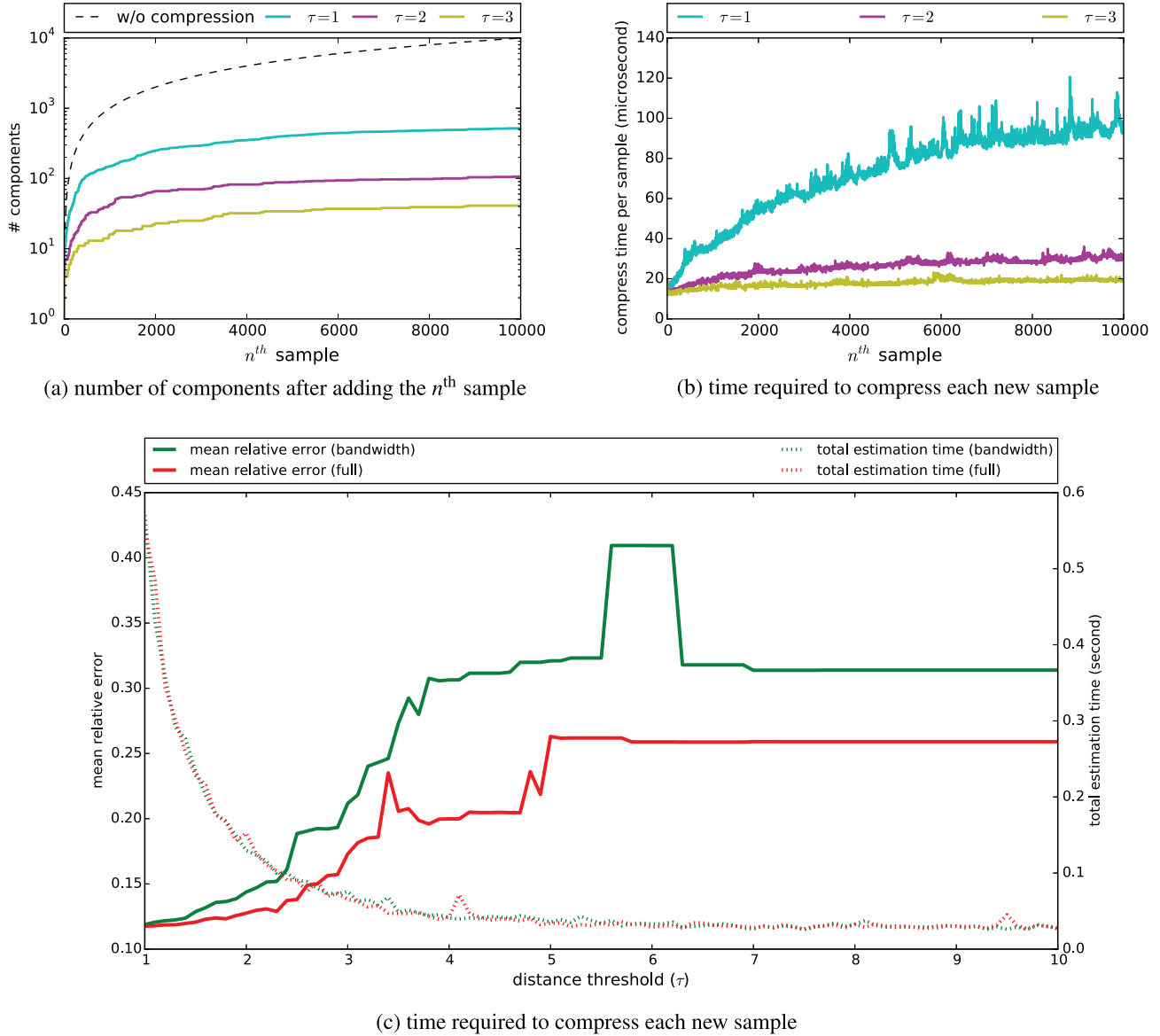


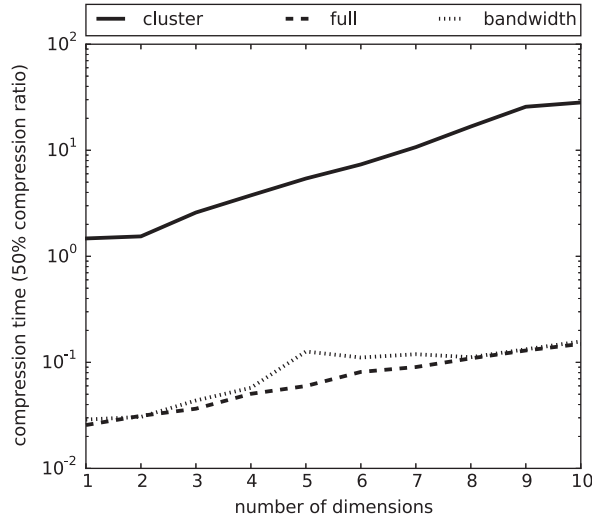
Fig. 6. Performance of the kernel compression algorithm on the data stream of spike features from rat hippocampus.

the streams (a), the decoding step (b) is invoked first to decode information from the newly observed neural signals then the encoding step (d) joins to update the encoding neural model with the newly observed data. The sliding window then proceeds (e) to process next batch of data from the streams.

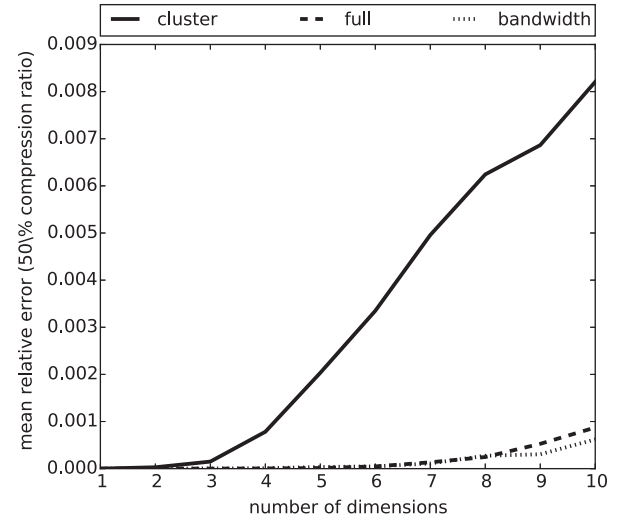
Accuracy of the decoding is evaluated by the euclidean distance between the observed position and the position estimated from neural data. To find the right amount of compression that would speed up the encoding/decoding to real-time and would not incur much accuracy loss, we varied the merging distance threshold ( $\tau$ ) to find the right parameters for each kernel merging methods. Because the number of spikes in the sliding window may vary from time to time, the amount of time required to process each window can also vary. We measured the max amount of time required to decode and encode each window and visualized the longest time needed by each algorithm as a function of  $\tau$ . The results are shown in Fig. 9(a). For the encoding/decoding framework to be able to process data streams in real-time, the time required to process each batch of neural and behavioral data in a sliding window has to be shorter than the streaming rate, which is 250 ms per batch. According to the results displayed in Fig. 9(a), the encoding/decoding

model that implements full covariance matrix match would need to set the distance threshold above 2.8, whereas the proposed bandwidth match can afford to have lower distance threshold  $\tau$ , which resulted in lower decoding errors as shown in Fig. 9(b). Given that the bandwidth matching method only matches diagonal elements of the covariance matrix and discard the rest, we can expect the bandwidth matching method to produce higher error. Interestingly, it was as accurate as its competition until  $\tau = 2.75$ , yet always faster.

Next, we set the distance threshold  $\tau = 3.0$  for both of the kernel merging methods. Results in Fig. 10(a) show progressions of the time required to process each window as the encoding models processed more data points over time. It can be seen that the performance in terms of speed was steady. Namely, the model can scale to large data stream efficiently. Decoding accuracy from both kernel compression methods visualized by the moving median of the decoding errors (cm) are shown in Fig. 10(b). From the result, it can be seen that the decoding errors decreased quickly and became stable when both decoders had sufficient training data for the decoding at about the 500th bin. The decoding model that was equipped with the full covariance matrix matching methods



(a) the average time required to compress each high-dimensional sample



(b) Mean relative errors from kernel density compression on high dimensional data streams

Fig. 7. Performance comparison on high dimensional data streams.

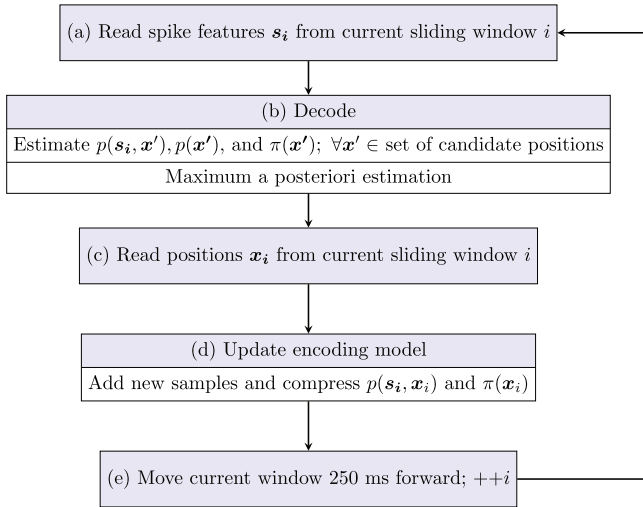


Fig. 8. Block diagram of the experimental real-time encoding/decoding framework.

obtained the median decoding error of 11.03 cm, compared to a slightly larger median error of 11.21 cm from the bandwidth matching method. However, the bandwidth matching method was much faster and truly capable of real-time decoding. In contrast to the full covariance match, the decoding model could not process all bins within the time limit of 250 ms per encoding/decoding bin.

In summary, we were able to speed up the Bayesian encoding/decoding framework by at least 300 times, which is fast enough to run the encoder/decoder on real-time data. The decoding accuracy loss from kernel compressions was relatively small and still manageable.

#### 4.4. Notes on the developing and testing environment

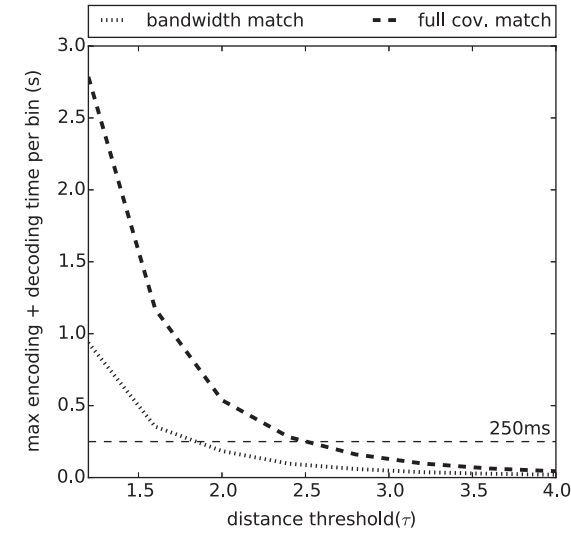
All the KDE algorithms that were used for the simulations in this paper were implemented in C programming language. The code for performance evaluations was written in Python and Cython. The

computing server used in all simulations runs on Intel Xeon 12-Core 2.7 GHz  $\times$  2 CPUs.

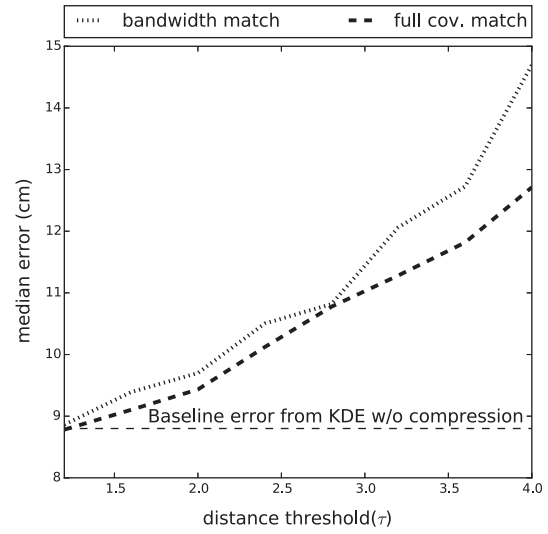
## 5. Discussion

Even though the proposed kernel compression algorithm has an advantage over some existing online KDE algorithms, such as the M-kernel merging algorithm [35], and most of cluster-based approaches [28–32] when dealing with multi-dimensional data streams, the proposed methods have not taken into account the situation where data stream is not stationary. Evolving density is a common problem that can impact performance of our kernel compression algorithms in many ways. Firstly, kernel merging would be less likely to occur and size of the model could expand uncontrollably if the distribution keeps drifting away its over time without repeating the same spot. Secondly, the fixed distance threshold parameter ( $\tau$ ) might get outdated when the underlying distribution expanded or condensed as the distribution evolved. The first problem can be handled easily using decaying weights similarly to [36,37], in which old components would be weighted less and can be removed after certain period of time. The more challenging problem is the second problem. Detecting changes [40–42] and adapting KDE model to changes in streaming data are interesting challenges that, if solved, could significantly improve accuracy of the density estimation of non-stationary data streams.

Our results show that the proposed kernel compression algorithm can be used for estimating the animal's position in an open field in a real-time fashion given a set of (unsorted) spikes recorded from a population of hippocampal neurons. The specific experimental application determines the constraints on the processing latency. In Fig. 10a we show that our algorithm satisfies the processing latency limit of 250 ms (equal to the bin size) per encoding/decoding bin. This limit is acceptable in the scenario of a brain-computer interface aimed at inferring the position of a freely running rodent in real-time [45]). For other applications more stringent time constraints are required. For example, in sleep neural activity patterns are reactivated at a 100 ms time scale [46], which imposes a real-time constraint of 5–10 ms on the neural decoding process. For these more demanding applications, there

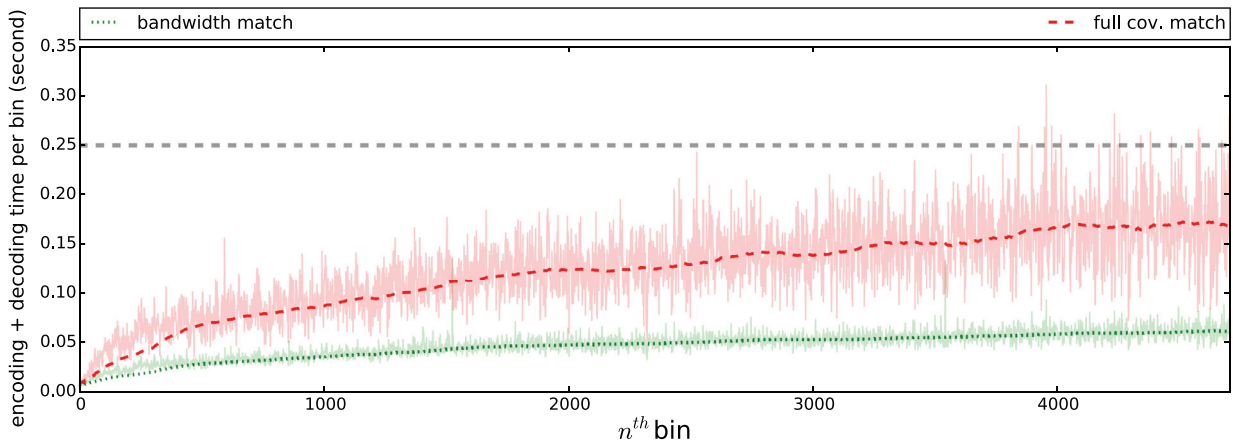


(a) Relation between the amount of time required to decode and encode each window and the merging distance threshold ( $\tau$ )

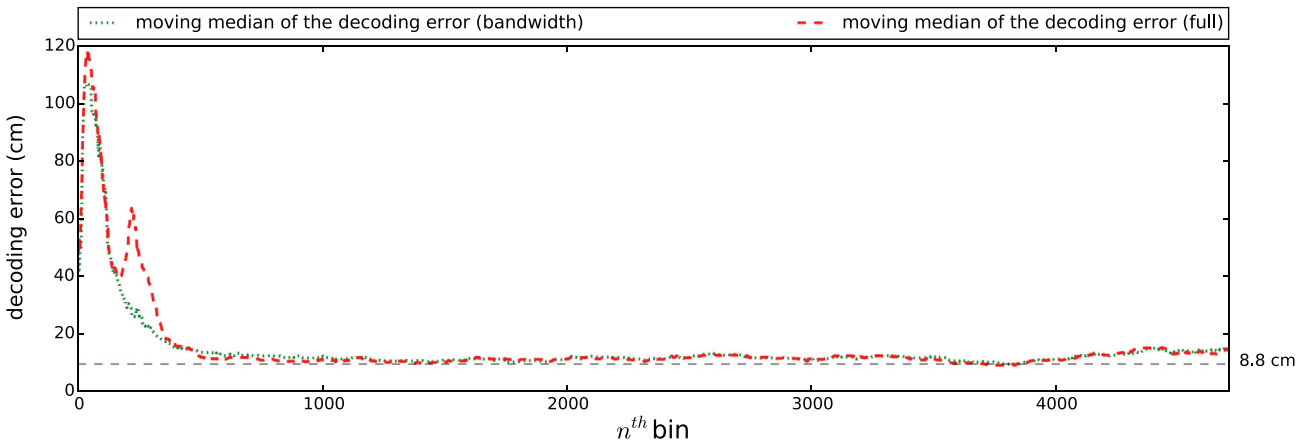


(b) Relation between the decoding error (cm) and the merging distance threshold ( $\tau$ )

**Fig. 9.** Performance comparison between the full covariance match and the proposed bandwidth match kernel merging approaches.



(a) Progression of the encoding/decoding speed over time



(b) Decoding accuracy of the decoder that implements full covariance matching method in details

**Fig. 10.** Performance comparison between the full covariance match and the proposed bandwidth match kernel merging approaches on the decoding of rat positions from unsorted spikes.

are several considerations that can further speed up the encoding/decoding time.

On modern machines a speedup in executing Bayesian decoding of hippocampal spikes can be simply provided with parallel computing using multi-threading: the  $p(s, x)$  of each tetrode can be processed independently on different threads, providing a speedup factor close to the number of tetrodes. Moreover, encoding and decoding on a single tetrode can run on separate threads. In order to have a measurement of computational time that is independent of the number of tetrodes and of the bin size, we measured the decoding time per single spike, which is the most critical bottleneck for a brain-machine interface. Our tests reveal that using the bandwidth match method and a threshold of 2.4 (which gives an acceptable decoding error of 10.42 cm) the average decoding time per spike is 0.95 ms, a value that is comparable to the typical duration of a neuronal spike (approximately 1 ms). Given this result, we are confident that our algorithm can be successfully applied to more stringent scenario as well, such as those requiring time bins of 10–20 ms.

Lastly, both the number of grid points and the dimensionality of the environment are two important parameters of the Bayesian decoding framework that impact decoding time. The KDE computational time directly depends on the number of grid points at which the densities need to be evaluated. By reducing the number of grid points, one can trade-off resolution of the decoded variable for increased processing speed. Furthermore, a reduction of the dimensionality of the decoded variables has the effect of reducing the number of components of the compressed model, which significantly reduces the time needed for density evaluation.

## 6. Conclusions

In this work, we have improved the decoding speed of the Bayesian framework for encoding and decoding of unsorted spikes from the rat hippocampus. Real challenges of this work were to design a kernel compression algorithm that could enable KDE to handle high-dimensional data streamed at high speed efficiently. Thus we proposed a fast kernel compression technique that not only can reduce size of a density estimator with very low accuracy loss, but also works well with high-dimensional data. More specifically, the proposed bandwidth match for kernel compression has been shown by the simulations presented in this paper to be very efficient especially when compressing high-dimensional data. Results from the real-time neural decoding simulation also confirmed the potentials of the bandwidth matching method.

## Acknowledgements

This work was partly supported by JSPS Strategic Young Researcher Overseas Visits Program for Accelerating Brain Circulation.

This work was supported by National Institute of Mental Health Grant MH-061976 and Office of Naval Research MURI N00014-10-1-0936 Grant to M. A. Wilson.

## References

- [1] E.N. Brown, L.M. Frank, D. Tang, M.C. Quirk, M.A. Wilson, A statistical paradigm for neural spike train decoding applied to position prediction from ensemble firing patterns of rat hippocampal place cells, *J. Neurosci.* 18 (18) (1998) 7411–7425.
- [2] K. Zhang, I. Ginzburg, B.L. McNaughton, T.J. Sejnowski, Interpreting neuronal population activity by reconstruction: unified framework with application to hippocampal place cells, *J. Neurophysiol.* 79 (2) (1998) 1017–1044.
- [3] L. Paninski, J. Pillow, J. Lewi, Statistical models for neural encoding, decoding, and optimal stimulus design, *Progr. Brain Res.* 165 (2007) 493–507.
- [4] E. Stark, M. Abeles, Predicting movement from multiunit activity, *J. Neurosci.* 27 (31) (2007) 8387–8394.
- [5] V. Ventura, Spike train decoding without spike sorting, *Neural Comput.* 20 (4) (2008) 923–963.
- [6] R.Q. Quiroga, S. Panzeri, Extracting information from neuronal populations: information theory and decoding approaches, *Nat. Rev. Neurosci.* 10 (3) (2009) 173–185.
- [7] J. O'Keefe, J. Dostrovsky, The hippocampus as a spatial map. preliminary evidence from unit activity in the freely-moving rat, *Brain Res.* 34 (1) (1971) 171–175.
- [8] J. Bures, A. Fenton, Y. Kaminsky, L. Zinyuk, Place cells and place navigation, *Proc. Nat. Acad. Sci.* 94 (1) (1997) 343–350.
- [9] R. Barbieri, L.M. Frank, D.P. Nguyen, M.C. Quirk, V. Solo, M.A. Wilson, E.N. Brown, Dynamic analyses of information encoding in neural ensembles, *Neural Comput.* 16 (2) (2004) 277–307.
- [10] A.E. Brockwell, A.L. Rojas, R. Kass, Recursive Bayesian decoding of motor cortical signals by particle filtering, *J. Neurophysiol.* 91 (4) (2004) 1899–1907.
- [11] A.P. Georgopoulos, A.B. Schwartz, R.E. Kettner, Neuronal population coding of movement direction, *Science* 233 (4771) (1986) 1416–1419.
- [12] A.P. Georgopoulos, R.E. Kettner, A.B. Schwartz, Primate motor cortex and free arm movements to visual targets in three-dimensional space. ii. Coding of the direction of movement by a neuronal population, *J. Neurosci.* 8 (8) (1988) 2928–2937.
- [13] D.W. Moran, A.B. Schwartz, Motor cortical representation of speed and direction during reaching, *J. Neurophysiol.* 82 (5) (1999) 2676–2692.
- [14] E. Salinas, L. Abbott, Vector reconstruction from firing rates, *J. Comput. Neurosci.* 1 (1) (1994) 89–107.
- [15] T.D. Sanger, Probability density estimation for the interpretation of neural population codes, *J. Neurophysiol.* 76 (4) (1996) 2790–2793.
- [16] S. Shoham, L.M. Paninski, M.R. Fellows, N.G. Hatsopoulos, J.P. Donoghue, R.A. Normann, Statistical encoding model for a primary motor cortical brain-machine interface, *IEEE Trans. Biomed. Eng.* 52 (7) (2005) 1312–1322.
- [17] W. Truccolo, U.T. Eden, M.R. Fellows, J.P. Donoghue, E.N. Brown, A point process framework for relating neural spiking activity to spiking history, neural ensemble, and extrinsic covariate effects, *J. Neurophysiol.* 93 (2) (2005) 1074–1089.
- [18] W. Wu, A. Shaikhouni, J. Donoghue, M. Black, Closed-loop neural control of cursor motion using a Kalman filter, *Proc. of Engineering in Medicine and Biology Society, IEMBS'04, 26th Annual International Conference of the IEEE*, vol. 2, IEEE, 2004, pp. 4126–4129.
- [19] A. Brockwell, R.E. Kass, A. Schwartz, Statistical signal processing and the motor cortex, *Proc. IEEE* 95 (5) (2007) 881–898.
- [20] M.S. Lewicki, A review of methods for spike sorting: the detection and classification of neural action potentials, *Network: Comput. Neural Syst.* 9 (4) (1998) R53–R78.
- [21] I.N. Goodman, D.H. Johnson, Information theoretic bounds on neural prosthesis effectiveness: the importance of spike sorting, in: *IEEE International Conference on Proc. of Acoustics, Speech and Signal Processing, IEEE*, 2008, pp. 5204–5207.
- [22] F. Kloosterman, S.P. Layton, Z. Chen, M.A. Wilson, Bayesian decoding using unsorted spikes in the rat hippocampus, *J. Neurophysiol.* 111 (1) (2014) 217–227.
- [23] A.G. Gray, A.W. Moore, Nonparametric density estimation: toward computational tractability, *Proc. of SIAM International Conference on Data Mining, SIAM*, 2003, pp. 203–211.
- [24] Y. Zheng, J. Jests, J.M. Phillips, F. Li, Quality and efficiency for kernel density estimates in large data, *Proc. of the 2013 ACM SIGMOD International Conference on Management of Data, ACM*, 2013, pp. 433–444.
- [25] D.W. Scott, S.J. Sheather, Kernel density estimation with binned data, *Commun. Statist.-Theory Methods* 14 (6) (1985) 1353–1359.
- [26] L. Holmström, The accuracy and the computational complexity of a multivariate binned kernel density estimator, *J. Multivar. Anal.* 72 (2) (2000) 264–309.
- [27] B. Silverman, Algorithm as 176: Kernel density estimation using the fast fourier transform, *Appl. Stat.* (1982) 93–99.
- [28] G.A. Babich, O.I. Camps, Weighted Parzen windows for pattern classification, *IEEE Trans. Pattern Anal. Mach. Intell.* 18 (5) (1996) 567–570.
- [29] B. Jeon, D.A. Landgrebe, Fast Parzen density estimation using clustering-based branch and bound, *IEEE Trans. Pattern Anal. Mach. Intell.* 16 (9) (1994) 950–954.
- [30] M. Kristan, A. Leonardis, Multivariate online kernel density estimation, in: *Proc. of Computer Vision Winter Workshop*, 2010, pp. 77–86.
- [31] M. Kristan, A. Leonardis, Online discriminative kernel density estimator with gaussian kernels, *IEEE Trans. Cybern.* 44 (3) (2014) 355–365.
- [32] K. Zhang, J.T. Kwok, Simplifying mixture models through function approximation, in: *Proc of Advances in Neural Information Processing Systems*, 2006, pp. 1577–1584.
- [33] M. Xu, H. Ishibuchi, X. Gu, S. Wang, Dm-KDE: dynamical kernel density estimation by sequences of KDE estimators with fixed number of components over data streams, *Front. Comput. Sci.* 8 (4) (2014) 563–580.
- [34] Y. Cao, H. He, H. Man, Somke: Kernel density estimation over data streams by sequences of self-organizing maps, *IEEE Trans. Neural Networks Learn. Syst.* 23 (8) (2012) 1254–1268.
- [35] A. Zhou, Z. Cai, L. Wei, W. Qian, M-kernel merging: towards density estimation over data streams, in: *Proc. of Database Systems for Advanced Applications, DASFAA 2003, Eighth International Conference on, IEEE*, 2003, pp. 285–292.

- [36] C. Heinz, B. Seeger, Towards kernel density estimation over streaming data, in: Proc. of International Conference on Management of Data, 2006, pp. 80–91.
- [37] C. Heinz, B. Seeger, Cluster kernels: resource-aware kernel density estimators over streaming data, *IEEE Trans. Knowl. Data Eng.* 20 (7) (2008) 880–893.
- [38] A.P. Boedihardjo, C.-T. Lu, F. Chen, Fast adaptive kernel density estimator for data streams, *Knowl. Inf. Syst.* 42 (2) (2015) 285–317.
- [39] A.R. Runnalls, Kullback–Leibler approach to gaussian mixture reduction, *IEEE Trans. Aerosp. Electron. Syst.* 43 (3) (2007) 989–999.
- [40] D. Kifer, S. Ben-David, J. Gehrke, Detecting change in data streams, in: Proc. of the Thirtieth international conference on Very large data bases, vol. 30, VLDB Endowment, 2004, pp. 180–191.
- [41] L.I. Kuncheva, Change detection in streaming multivariate data using likelihood detectors, *IEEE Trans. Knowl. Data Eng.* 25 (5) (2013) 1175–1180.
- [42] Y. Sakamoto, K. ichi Fukui, J. Gama, D. Nicklas, K. Moriyama, M. Numao, Concept drift detection with clustering via statistical change detection methods, in: Proc. of The Seventh International Conference on Knowledge and Systems Engineering, KSE2015, 2015.
- [43] F. Kloosterman, T.J. Davidson, S.N. Gomperts, S.P. Layton, G. Hale, D.P. Nguyen, M.A. Wilson, Micro-drive array for chronic in vivo recording: drive fabrication, *J. Vis. Exp.* 26 (2009).
- [44] T.J. Davidson, F. Kloosterman, M.A. Wilson, Hippocampal replay of extended experience, *Neuron* 63 (4) (2009) 497–507.
- [45] C. Guger, T. Gener, C.M. Pennartz, J.R. Brotons-Mas, G. Edlinger, S.B. i Badia, P. Verschure, S. Schaffelhofer, M.V. Sanchez-Vives, Real-time position reconstruction with hippocampal place cells, *Front. Neurosci.* 5 (2011).
- [46] J. O'Neill, B. Pleydell-Bouverie, D. Dupret, J. Csicsvari, Play it again: reactivation of waking experience and memory, *Trends Neurosci.* 33 (5) (2010) 220–229.