A SLIDING-WINDOW ONLINE FAST VARIATIONAL SPARSE BAYESIAN LEARNING ALGORITHM

Thomas Buchgraber[⋆] *Dmitriy Shutin*[†] *H. Vincent Poor*[†]

[⋆] Signal Processing and Speech Comm. Lab., Graz University of Technology, Austria
[†] Department of Electrical Engineering, Princeton University, USA

ABSTRACT

In this work a new online learning algorithm that uses automatic relevance determination (ARD) is proposed for fast adaptive nonlinear filtering. A sequential decision rule for inclusion or deletion of basis functions is obtained by applying a recently proposed fast variational sparse Bayesian learning (SBL) method. The proposed scheme uses a sliding window estimator to process the data in an online fashion. The noise variance can be implicitly estimated by the algorithm. It is shown that the described method has better mean square error (MSE) performance than a state of the art kernel recursive least squares (Kernel-RLS) algorithm when using the same number of basis functions.

Index Terms— Variational inference, sparse Bayesian learning, online learning

1. INTRODUCTION

Online learning is used in diverse areas of signal processing [1]. In applications like system identification, channel equalization and time series prediction, in which data is time varying and arrives sequentially, online learning can be the only method of choice.

We define the available training set at time instant n as a set $\{\boldsymbol{x}_i, t_i\}_{i=1}^n$ consisting of d-dimensional real input vectors $\boldsymbol{x}_i \in \mathbb{R}^d$ and real valued scalar targets $t_i \in \mathbb{R}$. The targets included in the sliding-window block at time n can be modeled as

$$\boldsymbol{t}_n = \boldsymbol{\Phi}_n \boldsymbol{w}_n + \boldsymbol{\epsilon}_n, \tag{1}$$

where $\boldsymbol{\Phi}_n = [\boldsymbol{\varphi}_{n,1}, \ldots, \boldsymbol{\varphi}_{n,L}]$ is a design matrix consisting of L basis vectors $\boldsymbol{\varphi}_{n,l} = [\psi_l(\boldsymbol{x}_{n-k+1}), \ldots, \psi_l(\boldsymbol{x}_n)]^T$ with basis functions $\psi_l(\cdot)$, $\boldsymbol{t}_n = [t_{n-k+1}, \ldots, t_n]^T$ is a vector containing all targets in the window, $\boldsymbol{w}_n \in \mathbb{R}^L$ is the weight vector, $\boldsymbol{\epsilon}_n$ is a zero mean additive white Gaussian perturbation vector with covariance matrix $\tau^{-1}\mathbf{I}$ and k is the sliding-window length. In online sparse signal representation, the aim is to find a small number of non-zero weights in $\boldsymbol{w}_n$ to represent the targets $\boldsymbol{t}_n$ for each time n.

Recently, a new approach called kernel adaptive filtering [2] has emerged, which has its origin in kernel methods [3], a powerful method in machine learning. One famous example of a kernel adaptive filter is the kernel recursive least squares (Kernel-RLS) algorithm [4]. In kernel methods, the kernel basis functions $\psi_l(\cdot)$ are placed at each input $\boldsymbol{x}_i$, for $i = 1, \ldots, n$. Thus, kernel models are build directly from the data itself. Naturally, online implementation of Kernel-RLS requires some sparsification rule, since data arrives sequentially and the model complexity would otherwise explode. The Kernel-RLS therefore uses a method called approximate linear dependency (ALD) that requires a threshold parameter to be set in advance. The performance of the Kernel-RLS strongly depends on the adjustment of the threshold. Another drawback of the Kernel-RLS is its limitation to only kernel basis functions, which is due to the incorporation of the kernel-trick [3].

In sparse Bayesian learning (SBL) [5, 6], where there is no need for a sparsification parameter, selecting components is done automatically using automatic relevance determination (ARD). Also, because SBL uses no kernel trick, it does not rely on only kernel basis functions like other kernel methods. Since SBL methods are designed for batch learning, i.e. access to all data is needed, it is not suitable for online processing. Another drawback, which limits the use of SBL in many online applications, is its slow convergence.

To overcome the drawbacks of SBL, we first suggest not to use all the data, which we cannot even access in most situations. Instead we propose to use a block of the last k samples as in (1). This can be seen as a sliding-window masking the most recent k samples from the training data. Secondly, we use a recently proposed fast variational SBL method [7], which is a variational counterpart to the fast marginal likelihood maximization method [8]. This method provides a fast decision rule for selecting model components and allows for addition of new basis functions as well as deletion of components currently in the model. This is opposed to the constructive sparsity of the Kernel-RLS, which only controls the addition of new components by using the ALD test. Furthermore, since no kernel-trick is used, this method is not limited to the use of kernel basis functions only and thus arbitrary functions can be incorporated.

The paper is organized as follows. In Section 2 we introduce variational SBL [6] and describe the recently proposed method for fast variational SBL [7]. In Section 3 we show how basis functions can be added and pruned from the model using a fast variational SBL decision rule. Then, in Section 4 we describe our new approach by considering a window block of samples that allows for online inclusion and deletion of basis functions. The resulting sliding-window fast variational SBL (SW-FV-SBL) method is then compared to a Kernel-RLS algorithm in Section 5. We finally conclude in Section 6.

Throughout the paper we use the following notation. Vectors are represented as boldface lowercase letters, e.g. $\boldsymbol{x}$, and matrices as boldface uppercase letters, e.g. $\mathbf{X}$. For vectors and matrices $(\cdot)^T$ denotes the transpose. The expression $\mathrm{diag}(\boldsymbol{x})$ stands for a diagonal matrix with the elements of $\boldsymbol{x}$ on the main diagonal; $\mathrm{tr}(\mathbf{X})$ is the trace of $\mathbf{X}$; $[\mathbf{X}]_{\bar{k},\bar{l}}$ denotes a matrix obtained by deleting the kth row and the lth column from $\mathbf{X}$; $[\mathbf{X}]_{:,\bar{l}}$ is a matrix obtained by removing the lth column of $\mathbf{X}$; similarly, $[\boldsymbol{x}]_{\bar{l}}$ is a vector obtained by removing the lth element from $\boldsymbol{x}$.

This work was supported in part by the Austrian Science Fund (FWF) under Award S10604-N13 within the national research network SISE, in part by an Erwin Schrödinger Postdoctoral Fellowship, FWF J2909-N23 and in part by the U.S. office of Naval Research under Grant N00014-09-1-0342.

2. FAST VARIATIONAL SPARSE BAYESIAN LEARNING

In the following we omit the time index n in the subscript to simlify notation; we reintroduce it in Section 4.

In variational sparse Bayesian learning, exemplified by the variational relevance vector machine [6], a Bayesian network (BN) in the form of a directed acyclic graph (DAG) given in Figure 1 is considered. The joint probability density function (pdf) given by the graph is factored as

$$p(\boldsymbol{w},\boldsymbol{t},\boldsymbol{\alpha},\tau) = p(\boldsymbol{t}|\boldsymbol{w},\tau)p(\boldsymbol{w}|\boldsymbol{\alpha})p(\boldsymbol{\alpha})p(\tau), \tag{2}$$

where $\boldsymbol{\alpha} = [\alpha_1,\dots,\alpha_L]^T$. From (1) we define the likelihood as $p(\boldsymbol{t}|\boldsymbol{w},\tau) = \mathcal{N}(\boldsymbol{t}|\boldsymbol{\Phi w},\tau^{-1}\mathbf{I})$ and additionally a hierarchical weight prior $p(\boldsymbol{w}|\boldsymbol{\alpha}) = \prod_{l=1}^{L}\mathcal{N}(w_l|0,\alpha_l^{-1})$. Using a common definition of the gamma distribution $\mathrm{Ga}(x|a,b) = \frac{b^a}{\Gamma(a)}x^{a-1}\exp(-bx)$, the remaining prior pdfs are given by $p(\alpha_l) = \mathrm{Ga}(\alpha_l|a_l,b_l)$ and $p(\tau) = \mathrm{Ga}(\tau|c,d)$. Since the derivation of the posterior $p(\boldsymbol{w},\boldsymbol{\alpha},\tau|\boldsymbol{t})$ is generally intractable [5], variational inference aims to approximate it using a proxy pdf $q(\boldsymbol{w},\boldsymbol{\alpha},\tau)$ that maximizes the so called "variational lower bound" $\mathcal{L}(q(\boldsymbol{w},\boldsymbol{\alpha},\tau))$ [3]

$$\begin{aligned}\ln p(\boldsymbol{t}) \;&\geq\; \int q(\boldsymbol{w},\boldsymbol{\alpha},\tau)\ln\frac{p(\boldsymbol{w},\boldsymbol{t},\boldsymbol{\alpha},\tau)}{q(\boldsymbol{w},\boldsymbol{\alpha},\tau)}d\boldsymbol{w}d\boldsymbol{\alpha}d\tau \\ &= \mathcal{L}(q(\boldsymbol{w},\boldsymbol{\alpha},\tau)),\end{aligned} \tag{3}$$

which is a lower bound for the log-evidence $\ln p(\boldsymbol{t})$. In [6] it is assumed that $q(\boldsymbol{w},\boldsymbol{\alpha},\tau)$ is factored as

$$q(\boldsymbol{w},\boldsymbol{\alpha},\tau) = q(\boldsymbol{w})q(\tau)\prod_{l=1}^{L}q(\alpha_l) \tag{4}$$

and the individual pdfs are defined as follows: $q(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w}|\hat{\boldsymbol{\mu}},\hat{\boldsymbol{\Sigma}})$, $q(\alpha_l) = \mathrm{Ga}(\alpha_l|\hat{a}_l,\hat{b}_l)$ and $q(\tau) = \mathrm{Ga}(\tau|\hat{c},\hat{d})$. Factorization (4) is also known as mean field approximation [3]. Maximization of the lower bound (3) with respect to the individual factors in (4) gives the following results [6]:

$$\hat{\boldsymbol{\Sigma}} = \left(\hat{\tau}\boldsymbol{\Phi}^T\boldsymbol{\Phi} + \mathrm{diag}(\hat{\boldsymbol{\alpha}})\right)^{-1} \tag{5}$$

$$\hat{\boldsymbol{\mu}} = \hat{\tau}\hat{\boldsymbol{\Sigma}}\boldsymbol{\Phi}^T\boldsymbol{t} \tag{6}$$

$$\hat{a}_l = a_l + 1/2, \quad \hat{b}_l = b_l + (\hat{\mu}_l{}^2 + \hat{\Sigma}_{ll})/2 \tag{7}$$

$$\hat{c} = c + \frac{k}{2}, \quad \hat{d} = d + \frac{||\boldsymbol{t}-\boldsymbol{\Phi}\hat{\boldsymbol{\mu}}||^2 + \mathrm{tr}(\hat{\boldsymbol{\Sigma}}\boldsymbol{\Phi}^T\boldsymbol{\Phi})}{2}, \tag{8}$$

where $\hat{\tau} = \mathbb{E}_{q(\tau)}\{\tau\} = \hat{c}/\hat{d}$, $\hat{\boldsymbol{\alpha}} = [\hat{\alpha}_1,\dots,\hat{\alpha}_L]^T$, $\hat{\alpha}_l = \mathbb{E}_{q(\alpha_l)}\{\alpha_l\} = \hat{a}_l/\hat{b}_l$, $\hat{\mu}_l$ is the lth element of the vector $\hat{\boldsymbol{\mu}}$ and $\hat{\Sigma}_{ll}$ is the lth main diagonal element of the matrix $\hat{\boldsymbol{\Sigma}}$. The notation $\mathbb{E}_{q(z)}\{\cdot\}$ denotes expectation with respect to the distribution $q(z)$. To find a sufficient statistic for the proxy pdf $q(\boldsymbol{w},\boldsymbol{\alpha},\tau)$, one has to iterate through the terms given in (5)-(8) until convergence, which corresponds to the maximum of the bound in (3). Because the bound in (3) is convex [3] with respect to each of the factors given in (4), we can update them in any arbitrary order.

In [7], this fact has been exploited in constructing a fast variational SBL method that assumes the ARD case, i.e. $a_l = b_l = 0, \forall l$. Instead of updating the terms (5)-(8) sequentially, the hyperparameters are updated in L subgroups each consisting of $q(\alpha_l)$ and $q(\boldsymbol{w})$. By updating only these two factors in the lth subgroup, a fixed point of the corresponding variational update expressions is found, i.e. an estimate of $q(\alpha_l)$ and $q(\boldsymbol{w})$ for which the variational lower bound reaches a maximum with respect to these two factors given that all others stay unchanged. The estimation of the noise precision pdf $q(\tau)$ using (8) can be done after all the subgroups have been updated. An efficient way to compute the fixed point $\hat{\alpha}_l^{[\infty]}$ of the lth subgroup with no need to iterate until convergence, is to substitute (5) and (6) into (7) and to solve for $\hat{\alpha}_l$. Note that for the ARD case we have $\hat{\alpha}_l = (\hat{\mu}_l{}^2 + \hat{\Sigma}_{ll})^{-1}$ and we can directly work with $\hat{\alpha}_l$ instead of $\hat{a}_l$ and $\hat{b}_l$. We summarize the results below; for more details the reader is referred to [7]:

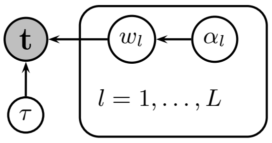

Fig. 1. A BN representing the SBL problem. Unshaded nodes are hidden random variables, whereas shaded nodes are observed.

$$\bar{\boldsymbol{\Sigma}}_l = \Big(\hat{\tau}\boldsymbol{\Phi}^T\boldsymbol{\Phi} + \sum_{j\neq l}\hat{\alpha}_j\boldsymbol{e}_j\boldsymbol{e}_j^T\Big)^{-1}, \tag{9}$$

$$\varsigma_l = \boldsymbol{e}_l^T\bar{\boldsymbol{\Sigma}}_l\boldsymbol{e}_l, \quad \rho_l^2 = \hat{\tau}^2\boldsymbol{e}_l^T\bar{\boldsymbol{\Sigma}}_l\boldsymbol{\Phi}^T\boldsymbol{t}\boldsymbol{t}^T\boldsymbol{\Phi}\bar{\boldsymbol{\Sigma}}_l\boldsymbol{e}_l, \tag{10}$$

$$\hat{\alpha}_l^{[\infty]} = \begin{cases}(\rho_l^2 - \varsigma_l)^{-1} & \rho_l^2 > \varsigma_l \\ \infty & \rho_l^2 \leq \varsigma_l\end{cases}, \tag{11}$$

where $\boldsymbol{e}_j = [0,\dots,0,1,0,\dots,0]^T$ is a vector of all zeros with 1 at the jth position. From (11) we see that evaluating $\hat{\alpha}_l^{[\infty]}$ reduces to just computing ς_l and ρ_l^2, where the case $\rho_l^2 \leq \varsigma_l$ renders the lth basis function as irrelevant and we can prune it from the model. This can be seen by substituting $\hat{\alpha}_l = \infty$ into (5) and (6) which leads to a zero variance and mean for the lth model weight and removes the influence of the corresponding basis function. The terms ς_l and ρ_l^2 in (10) can be efficiently computed without explicitly computing the matrix inversion in (9) for each l, which is given in [7].

3. ADAPTIVE DELETION AND INCLUSION OF BASIS FUNCTIONS

Equation (11) provides a fast rule for deciding whether to keep or delete a basis function from the model corresponding to an $\hat{\alpha}_l^{[\infty]}$ of finite or infinite value respectively. Furthermore we can ask if a new candidate basis function should be added to a given model or not. In the following we will discuss both situations in greater detail.

3.1. Testing basis functions currently in the model

For a given model with design matrix $\boldsymbol{\Phi}$ we can test the lth basis function - corresponding to the lth column of $\boldsymbol{\Phi}$ - and update the terms according to Algorithm 1. We use the notation $\hat{\alpha}_l^{\text{old}}$ to denote the previous value of $\hat{\alpha}_l$. When a basis function is kept, we use the matrix inversion lemma [9] to efficiently add the term $(\hat{\alpha}_l - \hat{\alpha}_l^{\text{old}})\boldsymbol{e}_l\boldsymbol{e}_l^T$ to the inverse of the weight covariance matrix $\hat{\boldsymbol{\Sigma}}$, which is a rank one update of the lth hyperparameter. When the lth basis function is pruned from the model, we delete the lth column of $\boldsymbol{\Phi}$ and the lth element of $\hat{\boldsymbol{\alpha}}$. Thus, according to (5), we remove the lth row and column from $\hat{\boldsymbol{\Sigma}}^{-1}$, which can be implemented using

Algorithm 1 Testing the lth basis φ_l

```
Compute $\varsigma_l$ and $\rho_l^2$ from (10)
if $\rho_l^2 > \varsigma_l$ then
    % keep basis
    $\hat{\alpha}_l = (\rho_l^2 - \varsigma_l)^{-1}, \; \hat{\mathbf{\Sigma}} = \hat{\mathbf{\Sigma}} - \frac{\hat{\mathbf{\Sigma}} \boldsymbol{e}_l \boldsymbol{e}_l^T \hat{\mathbf{\Sigma}}}{(\hat{\alpha}_l - \hat{\alpha}_l^{\text{old}})^{-1} + \boldsymbol{e}_l^T \hat{\mathbf{\Sigma}} \boldsymbol{e}_l}$
else
    % prune basis
    $\hat{\mathbf{\Sigma}} = \hat{\mathbf{\Sigma}}_{\bar{l}}$ from (12), $\hat{\boldsymbol{\alpha}} = [\hat{\boldsymbol{\alpha}}]_{\bar{l}}, \; \mathbf{\Phi} = [\mathbf{\Phi}]_{:,\bar{l}}, \; L = L - 1$
end if
```

another rank one update:

$$\hat{\mathbf{\Sigma}}_{\bar{l}} = \left[\hat{\mathbf{\Sigma}} - \frac{\hat{\mathbf{\Sigma}} \boldsymbol{e}_l \boldsymbol{e}_l^T \hat{\mathbf{\Sigma}}}{\boldsymbol{e}_l^T \hat{\mathbf{\Sigma}} \boldsymbol{e}_l} \right]_{\bar{l},\bar{l}} . \tag{12}$$

3.2. Testing basis functions not included in the model

To test a new candidate basis function for an existing model containing L basis vectors in the columns of the design matrix $\mathbf{\Phi}$ we compute

$$\bar{\mathbf{\Sigma}}_{L+1} = \begin{pmatrix} \hat{\tau} \mathbf{\Phi}^T \mathbf{\Phi} + \text{diag}(\hat{\boldsymbol{\alpha}}) & \hat{\tau} \mathbf{\Phi}^T \boldsymbol{\varphi}_{L+1} \\ \hat{\tau} \boldsymbol{\varphi}_{L+1}^T \mathbf{\Phi} & \hat{\tau} \boldsymbol{\varphi}_{L+1}^T \boldsymbol{\varphi}_{L+1} \end{pmatrix}^{-1} , \tag{13}$$

which is equivalent to (9) if $\mathbf{\Phi}$ would have been extended by the basis vector $\boldsymbol{\varphi}_{L+1}$ in an additional column at the end. By using the inversion rule for structured matrices [10], (13) is equivalent to

$$\bar{\mathbf{\Sigma}}_{L+1} = \begin{pmatrix} \mathbf{V} & -\gamma \hat{\tau} \hat{\mathbf{\Sigma}} \mathbf{\Phi}^T \boldsymbol{\varphi}_{L+1} \\ -\gamma \hat{\tau} \boldsymbol{\varphi}_{L+1}^T \mathbf{\Phi} \hat{\mathbf{\Sigma}} & \gamma \end{pmatrix} \tag{14}$$

where $\mathbf{V} = \hat{\mathbf{\Sigma}} + \gamma \hat{\tau}^2 \hat{\mathbf{\Sigma}} \mathbf{\Phi}^T \boldsymbol{\varphi}_{L+1} \boldsymbol{\varphi}_{L+1}^T \mathbf{\Phi} \hat{\mathbf{\Sigma}}$ and

$$\gamma = (\hat{\tau} \boldsymbol{\varphi}_{L+1}^T \boldsymbol{\varphi}_{L+1} - \hat{\tau}^2 \boldsymbol{\varphi}_{L+1}^T \mathbf{\Phi} \hat{\mathbf{\Sigma}} \mathbf{\Phi}^T \boldsymbol{\varphi}_{L+1})^{-1}. \tag{15}$$

Similarly we can efficiently compute $\hat{\mathbf{\Sigma}}_{L+1}$ by using $\lambda = (\gamma^{-1} + \hat{\alpha}_{L+1})^{-1}$, and $\mathbf{W} = \hat{\mathbf{\Sigma}} + \lambda \hat{\tau}^2 \hat{\mathbf{\Sigma}} \mathbf{\Phi}^T \boldsymbol{\varphi}_{L+1} \boldsymbol{\varphi}_{L+1}^T \mathbf{\Phi} \hat{\mathbf{\Sigma}}$ or by updating $\bar{\mathbf{\Sigma}}_{L+1}$ with

$$\begin{aligned} \hat{\mathbf{\Sigma}}_{L+1} &= \begin{pmatrix} \mathbf{W} & -\lambda \hat{\tau} \hat{\mathbf{\Sigma}} \mathbf{\Phi}^T \boldsymbol{\varphi}_{L+1} \\ -\lambda \hat{\tau} \boldsymbol{\varphi}_{L+1}^T \mathbf{\Phi} \hat{\mathbf{\Sigma}} & \lambda \end{pmatrix} \\ &= \bar{\mathbf{\Sigma}}_{L+1} - \frac{\bar{\mathbf{\Sigma}}_{L+1} \boldsymbol{e}_{L+1} \boldsymbol{e}_{L+1}^T \bar{\mathbf{\Sigma}}_{L+1}}{\hat{\alpha}_{L+1}^{-1} + \boldsymbol{e}_{L+1}^T \bar{\mathbf{\Sigma}}_{L+1} \boldsymbol{e}_{L+1}} \end{aligned} . \tag{16}$$

Algorithm 2 summarizes the main steps for adding or rejecting a new candidate basis function.

Algorithm 2 Testing a new basis φ_{L+1}

```
Compute $\varsigma_{L+1}$ and $\rho_{L+1}^2$ from (10) using $\bar{\mathbf{\Sigma}}_{L+1}$ from (14)
if $\rho_{L+1}^2 > \varsigma_{L+1}$ then
    % add new basis
    $\hat{\alpha}_{L+1} = (\rho_{L+1}^2 - \varsigma_{L+1})^{-1}$
    Compute $\hat{\mathbf{\Sigma}} = \hat{\mathbf{\Sigma}}_{L+1}$ using (16)
    $\mathbf{\Phi} = [\mathbf{\Phi}, \boldsymbol{\varphi}_{L+1}]$, $\hat{\boldsymbol{\alpha}} = [\hat{\boldsymbol{\alpha}}^T, \hat{\alpha}_{L+1}]^T$, $L = L + 1$
else
    % reject new basis - no action needed
end if
```

4. SLIDING-WINDOW ONLINE LEARNING

Now we are ready to bring the two key elements of fast variational SBL, namely pruning existing components and adding new arriving basis functions, into a single framework that we term the SW-FV-SBL algorithm. For that, we reintroduce the time index n in the subscript indices of the variables to emphasize their time dependency. Algorithm 3 presents the implementation details for the proposed online SW-FV-SBL method. Note that references to equations and subroutines given in Algorithm 3 assume the use of the windowed versions of the variables, i.e. $\mathbf{\Phi}_n$ and $\boldsymbol{t}_n$ instead of $\mathbf{\Phi}$ and $\boldsymbol{t}$, when referring back to previous sections. In the initial phase of Algorithm 3, the sliding-window length k is assumed to grow until $k = K$, where K is the final sliding-window length. Computing $\mathbf{\Phi}_n$ and $\boldsymbol{t}_n$ during the algorithm depends on the current window length k and thus the number of rows in both terms also grows until $k = K$. The method starts with an arbitrary basis function ψ^* evaluated at the first input sample $\boldsymbol{x}_1$ in the design matrix. The other variables are then initialized according to Algorithm 3. In some cases it is better not to start the re-estimation of the noise precision $\hat{\tau}_n$ at the very beginning. We have observed that especially at the initial phase where basis functions are rare, the noise precision estimate performs poorly and a delayed re-estimation start could give better results. At each iteration, all basis functions in the model are tested, then updated or pruned according to Algorithm 1. A new candidate basis function is tested, then kept or rejected according to Algorithm 2. As given in [5], the model prediction at time n for a test sample $\boldsymbol{x}^*$ can be computed by

$$y_n^* = \boldsymbol{\phi}(\boldsymbol{x}^*)^T \hat{\boldsymbol{\mu}}_n, \tag{17}$$

where we define $\boldsymbol{\phi}(\boldsymbol{x}) = [\psi_1(\boldsymbol{x}), \ldots, \psi_L(\boldsymbol{x})]^T$, the evaluation vector of all basis functions currently in the model.

Algorithm 3 Sliding-window online learning algorithm

```
Initialize $\Phi_1 = \psi^*(\boldsymbol{x}_1)$ with basis function $\psi^*$
Set $\hat{\tau}_1$ to an initial value, $n = 1$, $k = 1$, $L = 1$, $\hat{\alpha}_{n,1} = 0$
Compute $\hat{\Sigma}_1 = (\hat{\tau}_1 \Phi_1^2)^{-1}$ and $\hat{\mu}_1 = \hat{\tau}_1 \hat{\Sigma}_1 \Phi_1 t_1 = \Phi_1^{-1} t_1$
for n = 2, 3, ... do
    Update $\hat{\tau}_n = \hat{c}_{n-1} / \hat{d}_{n-1}$ from (8)
    if $k < K$ then
        $k = k + 1$  % grow window length until $k = K$
    end if
    Update $\mathbf{\Phi}_n$ and $\boldsymbol{t}_n$ with current window length $k$
    Compute $\hat{\mathbf{\Sigma}}_n$ from (5)
    % Test all current basis functions
    Run Algorithm 1, $\forall l = 1, \ldots, L$
    % Test a new candidate basis function
    Run Algorithm 2 with candidate basis $\boldsymbol{\varphi}_{n,L+1}$
    Compute $\hat{\boldsymbol{\mu}}_n$ from (6)
end for
```

5. SIMULATION

In this section, we evaluate the performance of the proposed algorithm on one-step-ahead prediction of Mackey-Glass chaotic time series, where we have generated the data as described in [2, Section 2.11.1]. Mackey-Glass has been extensively used for benchmarking different time-series prediction algorithms [11, 12]. In one-step-ahead time-series prediction, the training (and test) data

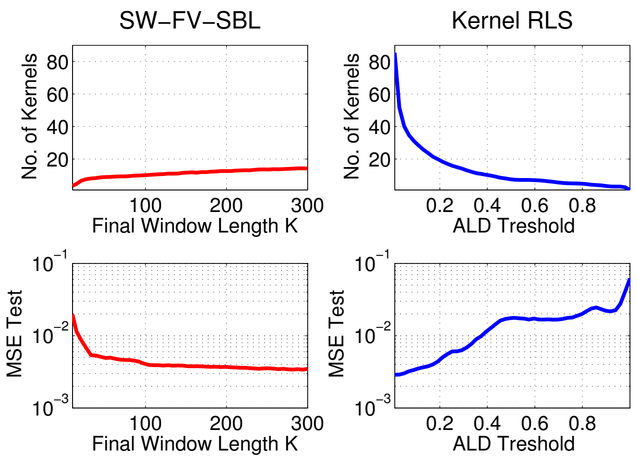

Fig. 2. Comparison of the SW-FV-SBL algorithm with an ALD Kernel-RLS for one-step ahead prediction of Mackey-Glass data. Results are averaged over 200 independent realizations for 500 samples. For the MSE, a test set of 200 samples was used.

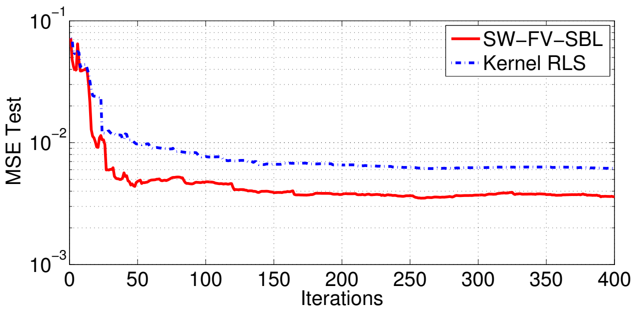

Fig. 3. Example learning curves of the SW-FV-SBL algorithm using a window length of 300 samples and the Kernel-RLS using an ALD threshold of 0.3. Both methods result in the same number of 14 kernels at the last iteration. For the MSE, a test set of 200 samples was used.

$\{\boldsymbol{x}_n, t_n\}_{n=1}^{\infty}$ can be generated with $\boldsymbol{x}_n = [t_{n-d}, \ldots, t_{n-2}, t_{n-1}]^T$, where each t_i is a sample from the time-series with additive white Gaussian noise having variance σ^2. For the simulations we have used an input dimension of $d = 7$ and a noise variance $\sigma^2 = 10^{-3}$. We compare the simulation results with a Kernel-RLS [4], which uses ALD as a constructive sparsification criterion. As basis functions $\psi_i(\cdot)$, we use the same Gaussian kernels defined as $\kappa(\boldsymbol{x}, \boldsymbol{x}_i) = \exp\{-||\boldsymbol{x} - \boldsymbol{x}_i||^2\}$ for both algorithms. Similarly to the Kernel-RLS, we define the candidate basis vector in Algorithm 3 at time n as $\boldsymbol{\varphi}_{n,L+1} = [\kappa(\boldsymbol{x}_{n-k+1}, \boldsymbol{x}_n), \ldots, \kappa(\boldsymbol{x}_n, \boldsymbol{x}_n)]^T$ and the initial basis function as $\psi^*(\cdot) = \kappa(\cdot, \boldsymbol{x}_1)$. For the noise precision estimate $\hat{\tau} = \hat{c}/\hat{d}$ from (8), a non-informative prior $c = d = 0$ was used and as the initial value we set $\hat{\tau}_1 = 10^5$.

The ALD criterion in the Kernel-RLS has the need to specify a threshold parameter [4] to adjust the amount of sparsity. In comparison, our method has no need for such an inconvenient parameter. However, in our case a final window length K has to be chosen. Another difference from the Kernel-RLS is that the SW-FV-SBL method provides an estimator for the noise precision $\hat{\tau}$, where in the Kernel RLS no such estimator exists. In Fig. 2 we see a comparison of both methods for different parameter settings. One can see that for the same estimated sparsity, the Kernel-RLS has a higher test mean square error (MSE) than our proposed method. A sample learning curve resulting in the same sparsity of 14 kernels is presented in Fig. 3 with a window length $W = 300$ for the SW-FV-SBL method and an ALD threshold of 0.3 for the Kernel-RLS.

6. CONCLUSIONS

In this work, a sliding-window fast variational sparse Bayesian learning algorithm for online learning and nonlinear filtering has been considered. The method exploits fast variational SBL with automatic relevance determination to determine the important basis functions by computing the fixed points of the update expressions of a variational posterior approximation. By processing data using a sliding window mechanism, the proposed algorithm can be used in online learning scenarios. We have shown that this allows adding and removing basis functions in an efficient online manner. The presented simulation results for prediction of Mackey-Glass chaotic time-series data demonstrate that the proposed algorithm has a better mean square error performance than a state of the art Kernel-RLS when using the same number of basis functions.

7. REFERENCES

[1] Simon Haykin, *Adaptive Filter Theory (4th Edition)*, Prentice Hall, Upper Saddle River, NJ, September 2001.

[2] W. Liu, J. C. Principe, and S. Haykin, *Kernel Adaptive Filtering*, Wiley Publishing, Hoboken, NJ, 2010.

[3] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*, Springer, New York, 1st ed. 2006. corr. 2nd printing edition, October 2007.

[4] Y. Engel, S. Mannor, and R. Meir, “The kernel recursive least-squares algorithm,” *IEEE Transactions on Signal Processing*, vol. 52, no. 8, pp. 2275 – 2285, Aug. 2004.

[5] M. E. Tipping, “Sparse Bayesian learning and the relevance vector machine,” *Journal of Machine Learning Research*, vol. 1, pp. 211–244, June 2001.

[6] C. M. Bishop and M. E. Tipping, “Variational relevance vector machines,” in *UAI ’00: Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*, San Francisco, CA, USA, 2000, pp. 46–53, Morgan Kaufmann Publishers Inc.

[7] D. Shutin, T. Buchgraber, S. R. Kulkarni, and H. V. Poor, “Fast variational sparse Bayesian learning with automatic relevance determination,” *IEEE Transactions on Signal Processing*, 2010, submitted.

[8] M. E. Tipping and A. C. Faul, “Fast marginal likelihood maximisation for sparse Bayesian models,” in *Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics*, Key West, FL,, January 2003.

[9] G. H. Golub and C. F. Van Loan, *Matrix Computations (Johns Hopkins Studies in Mathematical Sciences - 3rd Edition)*, The Johns Hopkins University Press, Baltimore, MD, 3rd edition, October 1996.

[10] K. B. Petersen and M. S. Pedersen, “The matrix cookbook,” October 2008, Version 20081110.

[11] J. D. Farmer and J. J. Sidorowich, “Predicting chaotic time series,” *Physical Review Letters*, vol. 8, no. 59, pp. 845–848, 1987.

[12] H. Jaeger and H. Haas, “Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication,” *Science*, pp. 78–80, April 2004.