KERNEL RECURRENT SYSTEM TRAINED BY REAL-TIME RECURRENT LEARNING ALGORITHM

Pingping Zhu, José C. Príncipe

University of Florida
Electrical and Computer Engineering
Gainesville, FL, 32611 USA

ABSTRACT

This paper presents a kernelized version of recurrent systems (KRS) and develops a kernel real-time recurrent learning (KRTRL) algorithm to train KRS. To avoid instabilities during training, the teacher forcing technique is adopted to modify the KRTRL learning. The proposed algorithm is compared with the KLMS in Lorenz time series prediction. The prediction performances of the proposed algorithm outperform the KLMS significantly.

Index Terms— recurrent networks, real-time recurrent learning (RTRL), reproducing kernel Hilbert space (RKHS), kernel adaptive filter, hidden state model

1. INTRODUCTION

In this paper, we propose a novel kernel adaptive filter algorithm for time series prediction with recurrent hidden state model, which is learned from the processing data and is able to describe the dynamics of the data.

Since the success of the support vector machine (SVM) [1], the kernel methodology has been applied to many algorithms of machine learning and adaptive filters. Utilizing the famed kernel trick, these linear methods have been recast in high dimensional reproducing kernel Hilbert spaces (RKHS) [2, 3, 4, 5] to yield more powerful nonlinear extensions, including kernel principal component analysis (KPCA) [6, 7] and kernel independent component analysis (KICA) [8, 9]. Recently, some on-line kernel adaptive filter algorithms are also developed to solve many nonlinear regression problems, such as kernel recursive least squares (KRLS) [10], kernel least mean squares (KLMS) [11] and kernel recurrent gamma network (KRGN) [12] algorithms, etc.

The KRLS and KLMS algorithms are able to discover the underlying input-output mapping very well for stationary cases. However, because of absence of hidden states, these algorithms cannot describe the underlying dynamics of the processing data for non-stationary cases. Therefore, they are not able to achieve good performance in such cases. In the KRGN algorithm, although the recursive Gamma filter is implemented into the RKHS, the specific topology only allows local recursion on the past of the input, which is easy to control stability but dose not allow a full recurrent state. To solve global recursiveness, the extended kernel recursive least squared (Ex-KRLS) algorithm [13] was proposed.But, it can only implement a random walk model in the hidden state. Then, another extended kernel recursive least squares algorithm (Ex-KRLS-KF) was proposed based on the Kalman filter (KF) [14], which can be applied to any known linear or nonlinear hidden state model cases. For both of these algorithms (Ex-KRLS and Ex-KRLS-KF), however, the hidden state model has to be known in advance, which may not be available in many signal processing problems.

To construct and learn the underlying hidden state model, the idea of recurrent neural networks (RNNs) [15] is adopted. In kernel recurrent systems, the KLMS algorithm is applied instead of the original perceptron algorithm, while making the topology recurrent, as in Jordan and Elman's networks [16, 17]. Furthermore, a kernel version of real-time recurrent learning (RTRL) algorithm [18] is derived to learn this recurrent network. To obtain a stable system, the teacher forcing technique is applied to the KRTRL which substitutes the dynamics of the hidden state using the desired response to avoid system instability.

The rest of the paper is organized as follows. In Section 2, the kernel recurrent system (KRS) are described. Next, the kernel RTRL (KRTRL) algorithm is derived in Section 3. Then, the experiment of Lorenz time series prediction is presented to evaluate the proposed algorithm in Section 4. Finally, discussions and conclusion are given in Section 5.

2. KERNEL RECURRENT NETWORKS

In this section, we present one general kernel recurrent network architectures, namely the state-space model and point out that any state-space model can be subsumed by a specific state-space model with a kernel state model and a linear measure model. Then, in the next section, we will develop a ker-

This work was supported by NSF grant ECCS 0856441.
ppzhu@cnel.ufl.edu
principe@cnel.ufl.edu

nel RTRL algorithm to train this specific state-space model.

Fig. 1 shows the block diagram of a state-space model. $\mathbf{u}_i \in \mathbb{R}^{n_\mathbf{u}}$, $\mathbf{x}_i \in \mathbb{R}^{n_\mathbf{x}}$ and $\mathbf{y}_i \in \mathbb{R}^{n_\mathbf{y}}$ are inputs, hidden states

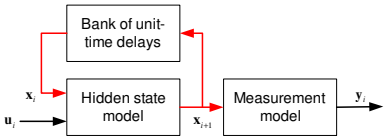

Fig. 1. State-space model, the feedback part is shown in red.

and output at time i, respectively. The state-space model is

$$\mathbf{x}_{i+1} = \mathbf{f}(\mathbf{x}_i, \mathbf{u}_i) \quad (1)$$

$$\mathbf{y}_i = \mathbf{h}(\mathbf{x}_{i+1}), \quad (2)$$

where $\mathbf{f}_i(\mathbf{x}_i, \mathbf{u}_i) = \left[f^{(1)}(\mathbf{x}_i, \mathbf{u}_i), \ldots, f^{(n_\mathbf{x})}(\mathbf{x}_i, \mathbf{u}_i)\right]^T$ and $\mathbf{h}_i(\mathbf{x}_i) = \left[h^{(1)}(\mathbf{x}_{i+1}), \ldots, h^{(n_\mathbf{y})}(\mathbf{x}_{i+1})\right]^T$. In this paper the superscript (k) denotes the kth component of a vector or the kth vector of a matrix.

In order to simplify the computation and reduce learning time, we modify this state-space model as

$$\begin{bmatrix} \mathbf{x}_{i+1} \\ \mathbf{y}_i \end{bmatrix} = \begin{bmatrix} \mathbf{f}(\mathbf{x}_i, \mathbf{u}_i) \\ \mathbf{h} \circ \mathbf{f}(\mathbf{x}_i, \mathbf{u}_i) \end{bmatrix} \quad (3)$$

$$\mathbf{y}_i = \begin{bmatrix} \mathbf{0} & \mathbf{I}_{n_\mathbf{y}} \end{bmatrix} \begin{bmatrix} \mathbf{x}_{i+1} \\ \mathbf{y}_i \end{bmatrix} \quad (4)$$

We denote $\begin{bmatrix} \mathbf{x}_{i+1} \\ \mathbf{y}_i \end{bmatrix}$ by $\mathbf{s}_{i+1}$ as a new hidden state, and $\begin{bmatrix} \mathbf{0} & \mathbf{I}_{n_\mathbf{y}} \end{bmatrix}$ by $\mathbf{W}_m$ as a known measurement matrix, where $\mathbf{I}_{n_\mathbf{y}}$ is an $n_\mathbf{y} \times n_\mathbf{y}$ identity matrix, and $\circ$ is the composition operator. Furthermore, let $\mathbf{g}(\mathbf{s}_i, \mathbf{u}_i) = \mathbf{f}(\mathbf{x}_i, \mathbf{u}_i)$.

We map $\mathbf{s}_i$ and $\mathbf{u}_i$ into the RKHS $\mathcal{H}_\mathbf{s}$ and $\mathcal{H}_\mathbf{u}$ as $\varphi(\mathbf{s}_i) \in \mathcal{H}_\mathbf{s}$ and $\phi(\mathbf{u}_i) \in \mathcal{H}_\mathbf{u}$, respectively. Then the new non-linear state transition weights $\mathbf{W}_\mathcal{H} = \begin{bmatrix} \mathbf{g}(\cdot,\cdot) \\ \mathbf{h} \circ \mathbf{g}(\cdot,\cdot) \end{bmatrix}$ are in the same RKHS $\mathcal{H}_\mathbf{su} = \mathcal{H}_\mathbf{s} \otimes \mathcal{H}_\mathbf{u}$, where $\otimes$ is the tensor operator. We denote $\varphi(\mathbf{s}_i) \otimes \phi(\mathbf{u}_i) \in \mathcal{H}_\mathbf{su}$ by $\psi(\mathbf{s}_i, \mathbf{u}_i)$.

At this point, we can express general state-space recurrent networks using a special kernel state-space model as

$$\mathbf{s}_{i+1} = \mathbf{W}_\mathcal{H}^T \psi(\mathbf{s}_i, \mathbf{u}_i) \quad (5)$$

$$\mathbf{y}_i = \mathbf{W}_m \mathbf{s}_{i+1} \quad (6)$$

where $\mathbf{W}_\mathcal{H}$ are weights in RKHS and $\mathbf{W}_m$ is a known linear matrix.

3. ON-LINE RECURRENT SYSTEM LEARNING

In the previous section, nonlinear recurrent systems are reformulated in (5) and (6). To perform on-line learning in this network, we develop the kernel real-time recurrent learning (KRTRL) algorithm in this section and teacher forcing technique is also introduced to avoid instability during training.

3.1. Kernel real-time recurrent learning algorithm

The kernel real-time recurrent learning (KRTRL) is developed based on the RTRL algorithm [18], which derives its name from the fact that adjustments are made to the weights of a fully connected recurrent network in real time. The KRTRL algorithm can also update the weights in RKHS, which are functions actually, while the network continues to perform its signal-processing function. The KRTRL is derived with respect to (5) and (6). Without loss of generality, all kernels involved in the algorithm are Gaussian kernels defined as

$$k(\mathbf{x}, \mathbf{y}) = exp(-\sigma \|\mathbf{x} - \mathbf{y}\|^2), \quad (7)$$

where σ is the kernel parameter. For the RKHS $\mathcal{H}_\mathbf{u}$ and $\mathcal{H}_\mathbf{s}$, the kernels are $k_{\sigma_\mathbf{u}}(\mathbf{x}, \mathbf{y})$ and $k_{\sigma_\mathbf{s}}(\mathbf{x}, \mathbf{y})$ with kernel parameters $\sigma_\mathbf{u}$ and $\sigma_\mathbf{s}$, respectively.

To complete the description of this learning process, we need to calculate the gradient of the error surface with respect to $\boldsymbol{\omega}_k \in \mathcal{H}_\mathbf{su}$, which is the kth component of $\mathbf{W}_\mathcal{H}$. To do this, we first use (6) to define the $n_\mathbf{y} \times 1$ error vector:

$$\mathbf{e}_i = \mathbf{d}_i - \mathbf{y}_i, \quad (8)$$

and the instantaneous sum of squared errors at time-step i is defined in term of $\mathbf{e}_i$ by

$$\mathcal{E}_i = \frac{1}{2} \mathbf{e}_i^T \mathbf{e}^i. \quad (9)$$

To implement this on-line learning algorithm, we use the method of steepest descent, which requires knowledge of the gradient matrix $\frac{\partial \mathcal{E}_i}{\partial \boldsymbol{\omega}_k}$, written as

$$\frac{\partial \mathcal{E}_i}{\partial \boldsymbol{\omega}_k} = \frac{\partial \mathbf{e}_i^T \mathbf{e}_i}{2 \partial \boldsymbol{\omega}_k} = -\mathbf{e}_i^T \frac{\partial \mathbf{y}_i}{\partial \boldsymbol{\omega}_k} = -\mathbf{e}_i^T \frac{\partial \mathbf{y}_i}{\partial \mathbf{s}_{i+1}} \frac{\partial \mathbf{s}_{i+1}}{\partial \boldsymbol{\omega}_k}. \quad (10)$$

According to (5) and (6), we have

$$\frac{\partial \mathbf{y}_i}{\partial \mathbf{s}_{i+1}} = \mathbf{W}_m, \quad (11)$$

and

$$\frac{\partial \mathbf{s}_{i+1}}{\partial \boldsymbol{\omega}_k} = \frac{\partial \mathbf{W}_\mathcal{H}^T \psi(\mathbf{s}_i, \mathbf{u}_i)}{\partial \boldsymbol{\omega}_k} = \mathbf{W}_\mathcal{H}^T \frac{\partial \psi(\mathbf{s}_i, \mathbf{u}_i)}{\partial \boldsymbol{\omega}_k} + \mathbf{I}_{n_\mathbf{s}}^{(k)} \psi(\mathbf{s}_i, \mathbf{u}_i)^T \quad (12)$$

where $\mathbf{I}_{n_\mathbf{s}}$ is the $n_\mathbf{s} \times n_\mathbf{s}$ identity matrix and $\mathbf{I}_{n_\mathbf{s}}^{(k)}$ is the kth column of the identity matrix.

According to representation theory, we can express $\boldsymbol{\omega}_{k,i}$ the weight at time i as a linear combination of $\{\psi(\mathbf{s}_j, \mathbf{u}_j)\}_{j=0}^{i-1}$ such as

$$\boldsymbol{\omega}_{(k),i} = \Psi_i \mathbf{c}_{k,i} \quad (13)$$

where $\Psi_i = [\psi(\mathbf{s}_0, \mathbf{u}_0), \ldots, \psi(\mathbf{s}_{i-1}, \mathbf{u}_{i-1})]$ and $\mathbf{c}_{k,i} \in \mathbb{R}^i$, and $\mathbf{W}_\mathcal{H}$ can be expressed as

$$\mathbf{W}_\mathcal{H} = \Psi_i \mathbf{C}_i \quad (14)$$

where $\mathbf{C}_i = [\mathbf{c}_{1,i}, \ldots, \mathbf{c}_{n_\mathbf{s},i}] \in \mathbb{R}^{i \times n_\mathbf{s}}$.

Then the first term of the last line in (12) is calculated by

$$\begin{aligned} \mathbf{W}_\mathcal{H}^T \frac{\partial \psi(\mathbf{s}_i, \mathbf{u}_i)}{\partial \boldsymbol{\omega}_k} &= \mathbf{C}_i^T \frac{\partial \Psi_i^T \psi(\mathbf{s}_i, \mathbf{u}_i)}{\partial \mathbf{s}_i} \frac{\partial \mathbf{s}_i}{\partial \boldsymbol{\omega}_k} \\ &= 2\sigma_\mathbf{x} \mathbf{C}_i^T \mathbf{D}_i \mathbf{S}_i^T \frac{\partial \mathbf{s}_i}{\partial \boldsymbol{\omega}_k} \\ &= \Gamma_i \frac{\partial \mathbf{s}_i}{\partial \boldsymbol{\omega}_k} \end{aligned} \tag{15}$$

where $\mathbf{D}_i = \text{diag}(\Psi^T \psi(\mathbf{s}_i, \mathbf{u}_i))$, $\mathbf{S}_i = [(\mathbf{s}_0 - \mathbf{s}_i), \ldots, (\mathbf{s}_{i-1} - \mathbf{s}_i)]$, and $\Gamma_i = \frac{\partial \mathbf{s}_{i+1}}{\partial \mathbf{s}_i} = 2\sigma_\mathbf{s} \mathbf{C}_i^T \mathbf{D}_i \mathbf{S}_i^T$. Substituting (15) into (12), we have the following recursive equation:

$$\frac{\partial \mathbf{s}_{i+1}}{\partial \boldsymbol{\omega}_k} = \Gamma_i \frac{\partial \mathbf{s}_i}{\partial \boldsymbol{\omega}_k} + \mathbf{I}_{n_\mathbf{s}}^{(k)} \psi(\mathbf{s}_i, \mathbf{u}_i)^T. \tag{16}$$

If we assume that $\frac{\partial \mathbf{s}_1}{\partial \boldsymbol{\omega}_k} = \mathbf{0}$, then we can express $\frac{\partial \mathbf{s}_i}{\partial \boldsymbol{\omega}_k}$ as

$$\frac{\partial \mathbf{s}_i}{\partial \boldsymbol{\omega}_k} = \mathbf{M}_{k,i} \Psi_i^T, \tag{17}$$

where $\mathbf{M}_{k,i}$ is a $n_\mathbf{s} \times i$ matrix and $\mathbf{M}_{k,1} = [0, 0, \ldots, \ldots, 0]^T$. Furthermore, (16) can be rewritten as

$$\begin{aligned} \frac{\partial \mathbf{s}_{i+1}}{\partial \boldsymbol{\omega}_k} &= \Gamma_i \mathbf{M}_i \Psi_i^T + \mathbf{I}_{n_\mathbf{s}}^{(k)} \psi(\mathbf{s}_i, \mathbf{u}_i)^T \\ &= \left[\Gamma_i \mathbf{M}_i, \mathbf{I}_{n_\mathbf{s}}^{(k)}\right] \Psi_{i+1}^T \\ &= \mathbf{M}_{k,i+1} \Psi_{i+1}^T \end{aligned} \tag{18}$$

where

$$\mathbf{M}_{k,i+1} = \left[\Gamma_i \mathbf{M}_{k,i}, \mathbf{I}_{n_\mathbf{s}}^{(k)}\right]. \tag{19}$$

Substituting (11) and (17) into (10), we have

$$\frac{\partial \mathcal{E}_i}{\partial \boldsymbol{\omega}_k} = -\mathbf{e}_i^T \mathbf{W}_m \mathbf{M}_{k,i+1} \Psi_{i+1}^T. \tag{20}$$

and can update $\boldsymbol{\omega}_k$ by

$$\boldsymbol{\omega}_{k,(i+1)} = \boldsymbol{\omega}_{k,i} + \eta \Psi_{i+1} \left(\mathbf{W}_m \mathbf{M}_{k,i+1}\right)^T \mathbf{e}_i \tag{21}$$

where η is the learning rate parameter.

Therefore, the feature matrix Ψ_i and coefficient matrix $\mathbf{C}_i \in \mathbb{R}^{i \times n_\mathbf{s}}$ should be updated by

$$\Psi_{i+1} = [\Psi_i, \psi(\mathbf{x}_i, \mathbf{u}_i)] \tag{22}$$

$$\mathbf{C}_{i+1}^{(k)} = \mathbf{c}_{k,i+1} = \left[\mathbf{c}_{k,i}^T, 0\right]^T + \eta_1 \mathbf{M}_{k,i+1}^T \mathbf{W}_m^T \mathbf{e}_i \quad (k = 1, 2, \ldots, n_\mathbf{x}) \tag{23}$$

If we substitute (19) into (23), we have

$$\begin{aligned} \mathbf{M}_{k,i+1}^T \mathbf{W}_m^T \mathbf{e}_i &= \left[\Gamma_i \mathbf{M}_{k,i}, \mathbf{I}_{n_\mathbf{s}}^{(k)}\right]^T \mathbf{W}_m^T \mathbf{e}_i \\ &= \left[\left(\mathbf{M}_{k,i}^T \Gamma_i^T \mathbf{W}_m^T \mathbf{e}_i\right)^T, \left(\mathbf{W}_m^T \mathbf{e}_i\right)^{(k)}\right]^T \end{aligned} \tag{24}$$

Algorithm: Kernel Real-Time Recurrent Learning

Initialization: For $i = 0$,
Input Dim.: $n_\mathbf{u}$, State Dim.: $n_\mathbf{s}$ and Output Dim.: $n_\mathbf{y}$
randomly set $\mathbf{u}_0$, $\mathbf{s}_0$ and $\mathbf{s}_1$,
Gaussian kernel size: $\sigma_\mathbf{s}$ and $\sigma_\mathbf{u}$
feature matrices: $\Psi_{i+1} = [\psi(\mathbf{s}_i, \mathbf{u}_i)]$
Coeff. matrix: randomly set $\mathbf{C}_{i+1} \in \mathbb{R}^{1 \times n_\mathbf{s}}$
Meas. matrix: $\mathbf{W}_m \in \mathbb{R}^{1 \times n_\mathbf{y}}$
Gradient matrix: $\mathbf{M}_{k,i+1} = \mathbf{0}_{n_\mathbf{s} \times 1}$ for $k = 1, \ldots, n_\mathbf{s}$
Learning rate parameters: η

Learning: For $i = 1, \ldots$
Forward pass:
$\mathbf{s}_{i+1} = \mathbf{C}_i^T \Psi_i^T \psi(\mathbf{s}_i, \mathbf{u}_i)$, $\mathbf{y}_i = \mathbf{W}_m \mathbf{s}_{i+1}$
$\mathbf{e}_i = \mathbf{d}_i - \mathbf{y}_i$
Backward pass:
$\mathbf{S}_i = [(\mathbf{s}_0 - \mathbf{s}_i), \ldots, (\mathbf{s}_{i-1} - \mathbf{s}_i)]$
$\mathbf{D}_i = \text{diag}(\Psi_i^T \varphi(\mathbf{s}_i, \mathbf{u}_i))$
$\Gamma_i = 2\sigma_\mathbf{s} \mathbf{C}_i^T \mathbf{D}_i \mathbf{S}_i^T$
$\mathbf{M}_{k,i+1} = \left[\Gamma_i \mathbf{M}_{k,i}, \mathbf{I}_{n_\mathbf{s}}^{(k)}\right]$ for $k = 1, \ldots, n_\mathbf{s}$
Update weights in RKHS $\mathbf{W}_\mathcal{H}$:
$\Psi_{i+1} = [\Psi_i, \psi(\mathbf{s}_i, \mathbf{u}_i)]$
$\mathbf{C}_{i+1}^{(k)} = \mathbf{C}_i^{(k)} + \eta \mathbf{M}_{k,i}^T \Gamma_i^T \mathbf{W}_m^T \mathbf{e}_i$ for $k = 1, \ldots, n_\mathbf{x}$
$\mathbf{C}_{i+1} = \left[\mathbf{C}_{i+1}^T, \eta \left(\mathbf{W}_m^T \mathbf{e}_i\right)^T\right]^T$

At this point, the learning procedure is complete and is summarized in the inset KRTRL algorithm.

The computational complexity of KRTRL algorithm is $O(n_\mathbf{s} n_\mathbf{y} i + n_\mathbf{s}^2 i)$, considering that D_i is a diagonal matrix. The computational complexity increases linearly with the sample number i, like the KLMS algorithm.

3.2. Teacher forcing

The KRTRL algorithm is a gradient based approach applied to a recurrent system as can be seen in Fig. 1. However, unlike the RTRL algorithm in which the activation functions are fixed in advance and only the weights are updated to adjust the network, the KRTRL constructs the expected KRS by adjusting the functions themselves. Therefore, the stability of the learning system is a big issue for this learning algorithm, and unfortunately, the stability analysis of recurrent functions is so complicated that there is no general method to easily specify stability conditions. Specifically, the updated functions and calculated gradients both depend on the inputs $\mathbf{u}_i$ and hidden states $\mathbf{s}_i$, which are functions in this case. Although, in the KRTRL algorithm the gradient of hidden states with respect to the functions $\frac{\partial \mathbf{s}_{i+1}}{\partial \omega_k}$ is propagated forward by (16) to (19), the propagated gradient cannot always reflect the real contribution of ω_k to the current cost function, because if the dynamics are unstable the weight update will be wrong. The update made based on the gradient does not guarantee that the function is modified in a proper way, whatever initial

condition or how small learning rate.

Fortunately, we can apply the teacher forcing technique to avoid this problem and train the KRS in a stable manner. The idea of the teacher forcing technique is to replace, in the training procedure, the actual hidden state $\mathbf{s}_{i+1}^{(j)}$ by the corresponding desired response $\mathbf{d}_i^{(j)}$ in subsequent computation of the behavior of the network. This can be done at every iteration or periodically to avoid states to diverge. The teacher forcing technique uses the desired response, which is assumed bounded and stable, to constrain and guide the learning dynamics of the hidden states by following the sequential desired responses and to force the KRS to converge in a proper way, avoiding system instability.

In detail, to derive a learning algorithm for this situation, we once again differentiate the cost function with respect to ω_k. We find only the term $\mathbf{M}_{k,i}$ need to be modified. In (17), $\frac{\partial \mathbf{s}_i}{\partial \omega_k} = \mathbf{M}_{k,i}\Psi_i^T$. If $\mathbf{s}_{i+1}^{(j)}$ is replaced by $\mathbf{d}_i^{(j)}$, then $\frac{\partial \mathbf{s}_{i+1}^{(j)}}{\partial \omega_k}$ should be equal to zero. That is to say the jth row of the matrix $\mathbf{M}_{k,i}$ is set as zeros. If all the hidden states $\mathbf{s}_{i+1}^{(j)}$ $(j = 1, \ldots, n_{\mathbf{s}})$ are all substituted by $\mathbf{d}_i$, the system training procedure degrades to a MIMO KLMS algorithm. In addition, the teacher forcing is applied once every t steps.

4. EXPERIMENTS AND RESULTS

To evaluate the proposed KRTRL algorithm, we chose to predict the Lorenz time series and the performance is compared with the KLMS algorithm. The system is nonlinear, three-dimensional and deterministic, defined by the set of differential equations given in [13]. Here, the experimental data are generated by the coefficients $\beta = 8/3$, $\sigma = 10$ and $\rho = 28$.

The first order difference approximation is used with a step size 0.01 to obtain the signal $\mathbf{x}_i = [x_i(1)x_i(2)x_i(3)]^T$. We pick the first component $x(1)$ as the input signal to implement a short prediction of the other components. The short term prediction task can be formulated as follows: using 10 past data $\mathbf{u}_i = [x(1)_{i-10}, \ldots, x(1)_{i-1}]^T$ as the input to predict $x(k)_i$ $k = 1, 2, 3$. For each prediction only the input signal $x(1)$ and desired signal $x(k)$ are available. We do not have the whole three signals at the same time. In the first 3000 time series, we randomly obtain 2000 time-steps to learn the filters, and use the next 200 time-steps to test these algorithms. To compare the prediction performances in a fair way, we run 50 independent simulations and the mean squared error (MSE) and signal noise ratio (SNR) are recorded.

All kernel parameters involved in this experiment are set as 1. For the KLMS algorithm the learning rates are set as 0.1 and 0.01. For our KRTRL algorithm, the learning rate is set as 0.1 and the dimension of the hidden states is 2. The teacher forcing technique is applied at every step $(t = 1)$. $[x_i(k), x_{i-1}(k)]$ is used as teacher signal to train the KRS. For testing, the hidden state corresponding to $x_i(k)$ is chosen as the system output. The prediction performances are tabulated as below:

Des.	Algorithms	SNR	MSE
$x(1)$	KLMS($\eta = 0.1$)	25.741+/-1.7716(dB)	0.00096218+/-0.00083813
	KLMS($\eta = 0.01$)	12.1933+/-2.1166(dB)	0.021771+/-0.013692
	KRS($\eta = 0.1$)	24.3963+/-1.6035(dB)	0.0012755+/-0.00097267
$x(2)$	KLMS($\eta = 0.1$)	-2.223+/-0.25915(dB)	1.6834+/-0.12373
	KLMS($\eta = 0.01$)	3.483+/-0.27802(dB)	0.44998+/-0.026276
	KRS($\eta = 0.1$)	23.4077+/-2.4422(dB)	0.0051275+/-0.002389
$x(3)$	KLMS($\eta = 0.1$)	-2.5039+/-0.22352(dB)	1.5296+/-0.11027
	KLMS($\eta = 0.01$)	3.0712+/-0.25711(dB)	0.42038+/-0.024646
	KRS($\eta = 0.1$)	17.9362+/-1.7771(dB)	0.014737+/-0.005885

Table 1. Using $x(1)$ as inputs to predict $x(1)$, $x(2)$ and $x(3)$

From these results, one can conclude that both algorithms can obtain good prediction performances while using $x(1)$ to predict $x(1)$. But the KLMS algorithm get poor performances when predicting $x(2)$ or $x(3)$ time series using $x(1)$. While, our algorithm can obtain good prediction performances in predicting the there dimensional system. This is because the mappings from $x(1)$ to $x(2)$ or $x(3)$ are more complex and the error increases with iterations proportionally to the largest Lyapunov exponent of the system that produced the time series, which is positive in this case (Chaotic system). The KRS learned by the KRTRL algorithm is designed to describe the system dynamics, which can quantify better the data structure to implement the prediction task. Therefore, there is no surprise that our proposed algorithm can still be successful.

5. CONCLUSION

In this paper, we proposed a kernel recurrent system (KRS) which can describe general state-space model recurrent networks and implement the nonlinear computations in the RKHS. Furthermore, a kernel real-time recurrent learning (KRTRL) algorithm is developed to train the KRS. The teacher forcing technique is applied to modify the KRTRL algorithm to improve the learning tasks. The KRTRL algorithm is applied to Lorenz time series prediction, and the prediction performances outperforms the KLMS algorithm when the input to output mappings are non-stationary.

The computational complexity of the KRTRL algorithm is similar to the KLMS increasing linearly with the sample number i. However, because of the recurrent architecture, the whole coefficients are updated when an new sample arrives, unlike the KLMS algorithm, which only updates current coefficient. This property also enhance the learning capability of this algorithm. In the experiment, the teacher forcing is applied at every step. We can also increase the interval t, like $t = 2$ or 3, to learn the state dynamics better, which however requires the smaller learning rate and longer training time.

6. REFERENCES

[1] V. Vapnik, *Adaptive filter thoery. (4th ed.)*, The Nature of Statistical Learning Theory. New York: Springer Verlag, 1995.

[2] A. Berlinet and C. Thomas-Agnan, *Reproducing Kernel Hilbert Spaces in Probability and Statistics*, Kluwer Academic, 2004.

[3] N. Aronszajn, "Theory of reproducing kernels," *Trans. Amer. Math. Soc.*, vol. 68, pp. 337–404, 1950.

[4] Thomas Hofmann, Bernhard Schölkopf, and Alexander J. Smola, "Kernel methods in machine learning," *The Annals of Statistics*, vol. 36, no. 3, pp. 1171–1220, Dec 2008.

[5] Martin Sewell, "Kernel Methods," *Department of Computer Science, University College London, UK*.

[6] B. Schölkopf and K. & Müller A. Smola, "Nonlinear component analysis as a kernel eigenvalue problem," *Neural Computation*, vol. 10, no. 5, pp. 1299–1319, July 1998.

[7] S. Mika, B. Schölkopf, A. Smola, K. Müller, M. Scholz, and G. Rätsch, "Kernel PCA and De-Noising in feature Spaces," *Proceedings of the 1998 conference on Advances in neural information processing systems II*, pp. 536–542, 1999.

[8] Francis R. Bach and Michael I. Jordan, "Kernel Independent Component Analysis," *Journal of Machine Learning Research*, vol. 3, pp. 1–48, Mar. 2003.

[9] H. Shen, S. Jegelka, and A. Gretton, "Fast Kernel-Based Independent Component Analysis," *IEEE Transactions on Signal Processing*, vol. 57, no. 9, pp. 3498–3511, Sep. 2009.

[10] 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. 2009.

[11] W. Liu, P. Pokharel, and J. C. Príncipe, "The Kernel Least Mean Square Algorithm," *IEEE Transactions on Signal Processing*, vol. 56, no. 2, pp. 543–554, Feb. 2009.

[12] G. Camps-Valls D. Tuia and M. Martinez-Ramon, "Explicit recursivity into reproducing kernel Hilbert spaces," *IEEE-ICASSP*, pp. 4148–4151, 2011.

[13] W. Liu, I. Park, Y. Wang, and J. C. Príncipe, "Extended Kernel Recursive Least Squares Algorithm," *IEEE Transactions on Signal Processing*, vol. 57, no. 10, pp. 3801–3814, Oct. 2009.

[14] P. Zhu, B. Chen, and J. C. Príncipe, "A novel extended kernel recursive least squares algorithm," *Neural Networks*, vol. 32, pp. 349–357, Aug. 2012.

[15] S. Haykin, *Neural Networks and Learning Machines (3rd Edition)*, 2008.

[16] M.L. Jordan, " Attractor dynamics and parallelism in a connectionist sequential machine," *Proceedings of the eighth annual conference of the cognitive science society*, pp. 531–546, 1986.

[17] J.L. Elman, " Finding structure in time," *Cognitive Science, 14*, pp. 179–211, 1990.

[18] R. J. Williams and D. Zipser, " Experimental analysis of the Real-time Recurrent Learning Algorithm," *Connection Science, 1*, pp. 87–111, 1 1989.