Prediction of Sunspot Time Series Data: Applications of QKLMS and QKRLS Trained with MSE & MCC

Kaifa Lu

***Abstract*—Kernel-based algorithms have been widely applied to adaptive filters as a powerful tool to learn nonlinearity from time series data. However, the growth of the radial basis function (RBF) structure and the intervene of non-Gaussian noises will destroy the performance of kernel adaptive filters (KAFs) just based on mean square error (MSE). In this project, we apply vector quantization (VQ) technique to compare the performance of quantized kernel least mean square (QKLMS) and kernel recursive least square (QKRLS) trained with MSE and maximum correntropy criterion (MCC) in predicting the sunspot time series. Simulation indicates that the algorithms trained with MCC perform better than those with MSE, showing more concentrated error peaks and smaller errors. The QKLMS and QKRLS outperform the baseline model (least mean square, LMS) when predicting the next 1st sample, in training and testing MSE, and convergence speed. However, when predicting the next 10th and 20th samples, LMS trained with MCC achieves the best performances, whereas the QKRLS and QKLMS algorithms perform better in learning and generating the sunspot trajectory.**

I. Introduction

Kernel methods have been demonstrated as a powerful tool in combination with different learning algorithms for modelling nonlinear systems in the fields of pattern recognition, adaptive signal processing, and tracking and prediction of motions [1, 2]. The basic idea of the kernel methods is to use a Mercer kernel function that maps pairs of input vectors into an inner product in the Reproducing Kernel Hilbert Space (RKHS) [3]. Typical applications of kernel methods have covered a broad range of learning algorithms, i.e., neural networks (NNs), support vector machine (SVM), regularization networks, and kernel adaptive filters (KAFs) in the RKHS [2]. Nowadays, increasing focuses have been put on developing different online kernel methods that can approximate the desired nonlinear functions underlying the time series data that data arrive sequentially [4], particularly in KAFs.

Typical families of the online KAF algorithms include kernel least mean square (KLMS) [5], kernel recursive least square (KRLS) [3], and kernel affine projection algorithm (KAPA) [6]. These algorithms are gaining increasing interests for their stable and efficient performances in handling and modelling nonlinear systems with Gaussian noises [2]. However, the online KAF algorithms cannot perform well and have exhibited a decaying performance in coping with a variety of realistic cases where the time series signals are intervened by non-Gaussian noises, for example, radar signal detection, underwater acoustic signals [7], and tracking of motion trajectory. The main reason why the KAFs cannot work well is that the online KAF algorithms only use the statistics of error signals in second order whereas ignore statistical information in a higher order that may help improve the filter performances [2].

To solve the problem, new criteria of cost functions based on Information Theoretic Learning (ITL) theory [8] are developed, for example, minimum error entropy (MEE) [9] and maximum correntropy criterion (MCC) [10]. The MCC has been widely used as a more generalized measure than MEE to maximize the information potentials behind signals due to less computational costs. Substituting the mean square error (MSE) with the MCC as the cost function of KLMS and KRLS is common practice to improve the performances of the adaptive filters in modelling signals, particularly with non-Gaussian noises. However, there arises another question that as the number of samples increases, the computational costs of KLMS and KRLS trained with MCC will increase incrementally and even become impractical.

To curb the rapid network growth and reduce computational costs, a variety of online sparsification techniques are proposed accordingly, including novelty criterion (NC) [11], surprise criterion (SC) [8], and approximate linear dependency (ALD) criterion [3], etc. These sparsification methods are effective in reducing the network size (the number of centers) of the KAFs because the redundant data samples are purely discarded [4]. Although the redundant data are not so important for the update of network structures, they are very useful and can be utilized to update the coefficients of the current network. Motivated by this idea, Chen et al [12] proposed an online vector quantization (VQ) approach to partitioning the input space and constraining the network size, which can be directly applied to online kernel methods.

In a brief, we apply the online VQ technique into KLMS and KRLS, separately called QKLMS and QKRLS. The aim of this project is to compare the implementations of the QKLMS and QKRLS algorithms trained with MSE and MCC to predict the next sample of the sunspot time series data. The remainder of this project is organized as follows: Section II clarifies the problem statement; Section III presents the implementation of the QKLMS and QKRLS algorithms trained with MSE and MCC. Section IV describes the simulation results and findings. Section V summarizes our main conclusions.

Project 2 report submitted on April 25, 2022.

K. Lu is with University of Florida, Gainesville, FL 32611, USA.

II. Problem Statement

The problem of sunspot time series prediction is formulated in the framework of using the KAF to map pairs of input vectors into an inner product in a high-dimensional Hilbert space and capture the linearity in RKHS to predict the next 1st, 10th, and 20th samples of the sunspot time series. Specifically, we use an input layer with 6 delays to map the sunspot time series data into an input embedding vector with the size of 6 in RKHS, and the next sample as the desired response to train the KAF with different adaptive algorithms (i.e., LMS, QKLMS and QKRLS). The goal of the KAF is to learn the filter weights that precisely predict the next sample of the sunspot time series data based on the last 6 samples.

III. Implementation

A. Preliminaries

1) Least Mean Square (LMS) Algorithm

As a baseline model, adaptive LMS algorithm is introduced to update the weights of the linear filter that can achieve precise prediction of the sunspot time series data.

The LMS algorithm minimizes the cost function [5]:

$$J(\boldsymbol{w}) = \sum_{i=1}^{N}(d_i - \boldsymbol{w}^T\boldsymbol{x}_i)^2 \quad (1)$$

Using the stochastic gradient, the rule of the weight updates is presented below [13]:

$$\boldsymbol{w}_i = \boldsymbol{w}_{i-1} + \eta e_i \boldsymbol{x}_i \quad (2)$$

Where: η is the step size; e_i is the error between the desired response d_i and filter output y_i, $e_i = d_i - \boldsymbol{w}_i^T\boldsymbol{x}_i$.

TABLE I
Pseudocodes of implementing the LMS algorithm

Algorithm 1: LMS Algorithm

Input: $(\boldsymbol{x}_n, d_n) = \big((x_n, x_{n-1}, \cdots, x_{n-L+1}), x_{n+1}\big)$, $L = 6$
Output: the filter weights $\boldsymbol{w} = [w_1, w_2, w_3, w_4, w_5, w_6]^T$
Initialization: step size η, $\boldsymbol{w}_0 = [w_1, w_2, w_3, w_4, w_5, w_6]^T \sim N(0, 0.1)$
While $\{(\boldsymbol{x}_i, d_i), i = 1, \cdots, N\}$ is available **do**
  $e_i = d_i - \boldsymbol{w}^T\boldsymbol{x}_i$
  $\boldsymbol{w}_i = \boldsymbol{w}_{i-1} + \eta e_i \boldsymbol{x}_i$
end while

2) Kernel Least Mean Square (KLMS) Algorithm

The main idea of the KLMS algorithm is to use the kernel methods to map the data into a high-dimensional feature space $\boldsymbol{\varphi}_i = \varphi(\boldsymbol{x}_i)$ and then construct a linear model in RKHS. Here, we select the Gaussian kernel function as follows:

$$\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) = \varphi^T(\boldsymbol{x}_i)\varphi(\boldsymbol{x}_i) = exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right) \quad (3)$$

The KLMS algorithm minimizes the cost function [5]:

$$J(\boldsymbol{\Omega}) = \sum_{i=1}^{N}\big(d_i - \boldsymbol{\Omega}^T\varphi(\boldsymbol{x}_i)\big)^2 \quad (4)$$

Using the stochastic gradient, the weights are updated below after n step training:

$$\boldsymbol{\Omega}_n = \eta\sum_{i=1}^{n} e_i\varphi(\boldsymbol{x}_i) \quad (5)$$

With an input of $\boldsymbol{x}$, the output of the filter with KLMS is calculated:

$$\boldsymbol{y} = \eta\sum_{i=1}^{N} e_i\kappa(\boldsymbol{x}_i, \boldsymbol{x}) \quad (6)$$

TABLE II
Pseudocodes of implementing the KLMS algorithm

Algorithm 2: KLMS Algorithm

Input: $(\boldsymbol{x}_n, d_n) = \big((x_n, x_{n-1}, \cdots, x_{n-L+1}), x_{n+1}\big)$, $L = 6$
Output: the weight $\boldsymbol{\Omega} = \sum_k^K \alpha_k\kappa(\cdot, \boldsymbol{x}_k)$
Initialization: step size η, Gaussian kernel size σ, center dictionary $\boldsymbol{C}_1 = \boldsymbol{C}_1\{\boldsymbol{x}_1\}$, coefficient vector $\boldsymbol{\alpha}_1 = [\eta d_1]$
While $\{(\boldsymbol{x}_i, d_i), i = 2, \cdots, N\}$ is available **do**
  $e_i = d_i - \sum_{j=1}^{i-1}\alpha_j\kappa\big(\boldsymbol{x}_i, \boldsymbol{C}_{i-1}\{\boldsymbol{x}_j\}\big)$
  $\boldsymbol{C}_i = \{\boldsymbol{C}_{i-1}, \boldsymbol{x}_i\}$, $\boldsymbol{\alpha}_i = [\boldsymbol{\alpha}_{i-1}, \eta e_i]$
end while

3) Kernel Recursive Least Square (KRLS) Algorithm

Similarly, the KRLS algorithm is formulated on the example sequence of $\{\varphi(\boldsymbol{x}_1), \cdots, \varphi(\boldsymbol{x}_N)\}$ and $\{d_1, \cdots, d_N\}$, and aims to minimize the cost function [3]:

$$J(\boldsymbol{\Omega}) = \sum_{i=1}^{N}\big(d_i - \boldsymbol{\Omega}^T\varphi(\boldsymbol{x}_i)\big)^2 + \lambda\|\boldsymbol{\Omega}\|^2 \quad (7)$$

By introducing

$$\boldsymbol{d}_i = [d_1, \cdots, d_i]^T \quad (8)$$

$$\boldsymbol{\Phi}_i = [\varphi(\boldsymbol{x}_1), \cdots, \varphi(\boldsymbol{x}_i)] \quad (9)$$

Then we have the equation of weight updates below:

$$\boldsymbol{\Omega}_i = \boldsymbol{\Phi}_i[\lambda\boldsymbol{I} + \boldsymbol{\Phi}_i\boldsymbol{\Phi}_i^T]^{-1}\mathbf{d}_i = \boldsymbol{\Phi}_i\mathbf{Q}_i\mathbf{d}_i = \boldsymbol{\Phi}_i\mathbf{a}_i \quad (10)$$

TABLE III
Pseudocodes of implementing the KRLS algorithm

Algorithm 3: KRLS Algorithm

Input: $(\boldsymbol{x}_n, d_n) = \big((x_n, x_{n-1}, \cdots, x_{n-L+1}), x_{n+1}\big)$, $L = 6$
Output: the weight $\boldsymbol{\Omega} = \sum_k^K a_k\kappa(\cdot, \boldsymbol{x}_k)$
Initialization: forgetting factor λ, Gaussian kernel size σ, center dictionary $\boldsymbol{C}_1 = \boldsymbol{C}_1\{\boldsymbol{x}_1\}$, coefficient vector $\boldsymbol{Q}_1 = \Big[\big(\lambda + \kappa(\boldsymbol{x}_1, \boldsymbol{x}_1)\big)^{-1}\Big]$, $\boldsymbol{a}_1 = [\boldsymbol{Q}_1 d_1]$
While $\{(\boldsymbol{x}_i, d_i), i = 2, \cdots, N\}$ is available **do**
  $\boldsymbol{h}_i = \big[\kappa\big(\boldsymbol{x}_i, \boldsymbol{x}_j\big)\big]^T, \boldsymbol{x}_j \in \boldsymbol{C}_{i-1}$
  $e_i = d_i - \boldsymbol{h}_i^T\boldsymbol{a}_{i-1}$
  $\boldsymbol{z}_i = \boldsymbol{Q}_{i-1}\boldsymbol{h}_i$
  $r_i = \lambda + \kappa(\boldsymbol{x}_i, \boldsymbol{x}_i) - \boldsymbol{z}_i^T\boldsymbol{h}_i$
  $\boldsymbol{Q}_i = \begin{bmatrix} \boldsymbol{Q}_{i-1} + \boldsymbol{z}_i r_i^{-1}\boldsymbol{z}_i^T & -\boldsymbol{z}_i r_i^{-1} \\ -\boldsymbol{z}_i^T r_i^{-1} & r_i^{-1} \end{bmatrix}$
  $\boldsymbol{a}_i = \begin{bmatrix} \boldsymbol{a}_{i-1} - \boldsymbol{z}_i r_i^{-1} e_i \\ r_i^{-1} e_i \end{bmatrix}$
  $\boldsymbol{C}_i = \{\boldsymbol{C}_{i-1}, \boldsymbol{x}_i\}$
end while

4) Online Vector Quantization (VQ) Algorithm

The core idea of the online VQ approach is to quantize the input space and assign one center to each quantized input space, and if the current quantized space has already been assigned a center, we don't need to add one new center and the new sample is used to update the coefficient of the existing center [4, 12]. Here, we adopt the Euclidean distance to measure the distances between the center dictionary and the inputting sample, and if

the distance satisfies certain threshold, the sample is adopted to update the coefficient; Otherwise, we assign the sample into a new quantized input space, as shown in Table IV.

TABLE IV
PSEUDOCODES OF IMPLEMENTING THE VQ ALGORITHM

Algorithm 4: Online VQ Algorithm

Input: $\boldsymbol{x}_n = (x_n, x_{n-1}, \cdots, x_{n-L+1}), L = 6$
Output: center dictionary $\boldsymbol{C} = \boldsymbol{C}\{\boldsymbol{x}\}$
Initialization: quantization threshold δ, center dictionary $\boldsymbol{C}_1 = \boldsymbol{C}_1\{\boldsymbol{x}_1\}$
While $\{\boldsymbol{x}_i, i = 2, \cdots, N\}$ is available **do**
  $dist(\boldsymbol{x}_i, \boldsymbol{C}_{i-1}\{\boldsymbol{x}\}) = \min_{1 \le j \le K_{i-1}} \left\|\boldsymbol{x}_i - \boldsymbol{C}_{i-1}\{\boldsymbol{x}_j\}\right\|_2$
  $j^* = \arg\min_{1 \le j \le K_{i-1}} \left\|\boldsymbol{x}_i - \boldsymbol{C}_{i-1}\{\boldsymbol{x}_j\}\right\|_2$
  if $dist(\boldsymbol{x}_i, \boldsymbol{C}_{i-1}\{\boldsymbol{x}\}) \le \delta$ **then**
    $\boldsymbol{C}_i = \boldsymbol{C}_{i-1}$
  else
    $\boldsymbol{C}_i = \{\boldsymbol{C}_{i-1}, \boldsymbol{x}_i\}$
  end if
end while

5) **Two Cost Functions: Mean Square Error (MSE) and Maximum Correntropy Criterion (MCC)**

The MSE, as one of the most widely used cost function, uses the second order moments of error signals to characterize the difference between the desired response and filter output, which performs well especially for time series signals with Gaussian noises. As an alternative, the idea of MCC is to replace the 2nd order moments with a geometric statistical interpretation of data in probability space. In other words, correntropy covers higher order moments of data, not restricted to second order statistics. Here, we use the Gaussian kernel to describe the correntropy of the error as follows:

$$V_i = \kappa_e(e_i) = exp\left(-\frac{\|e_i\|^2}{\sigma_e^2}\right) \tag{11}$$

Then we apply the online VQ approach into the KLMS and KRLS algorithms trained with MSE and MCC, separately called QKLMS-MSE, QKLMS-MCC, QKRLS-MSE, QKRLS-MCC algorithms.

B. Quantized Kernel Least Mean Square (QKLMS) Algorithm

1) **QKLMS Algorithm Trained with MSE**

Compared with the KLMS algorithm, the QKLMS algorithm first calculates the Euclidean distance between the sample $\boldsymbol{x}_i$ and center dictionary $\boldsymbol{C}_{i-1}$, and if the minimal distance is less than certain threshold, the sample falls into one quantized input space and the "redundant" data are used to locally update the coefficient of the center in the quantized space. Otherwise, the sample is assigned into a new quantized input space. Intuitively, the coefficient update can increase the utilization efficiency of centers, reduce the computational costs, and yield an equivalent accuracy to the KLMS algorithm [12].

TABLE V
PSEUDOCODES OF IMPLEMENTING THE QKLMS ALGORITHM TRAINED WITH MSE

Algorithm 5: QKLMS Algorithm Trained with MSE

Input: $\boldsymbol{x}_n, d_n) = \big((x_n, x_{n-1}, \cdots, x_{n-L+1}), x_{n+1}\big), L = 6$
Output: the weight $\boldsymbol{\Omega} = \sum_k^K \alpha_k \kappa(\cdot, \boldsymbol{x}_k)$
Initialization: step size η, Gaussian kernel size σ, quantization threshold δ, center dictionary $\boldsymbol{C}_1 = \boldsymbol{C}_1\{\boldsymbol{x}_1\}$, coefficient vector $\boldsymbol{\alpha}_1 = [\eta d_1]$
While $\{(\boldsymbol{x}_i, d_i), i = 2, \cdots, N\}$ is available **do**
  $e_i = d_i - \sum_{j=1}^{K_{i-1}} \alpha_j \kappa\big(\boldsymbol{x}_i, \boldsymbol{C}_{i-1}\{\boldsymbol{x}_j\}\big)$
  $dist(\boldsymbol{x}_i, \boldsymbol{C}_{i-1}\{\boldsymbol{x}\}) = \min_{1 \le j \le K_{i-1}} \left\|\boldsymbol{x}_i - \boldsymbol{C}_{i-1}\{\boldsymbol{x}_j\}\right\|_2$
  $j^* = \arg\min_{1 \le j \le K_{i-1}} \left\|\boldsymbol{x}_i - \boldsymbol{C}_{i-1}\{\boldsymbol{x}_j\}\right\|_2$
  if $dist(\boldsymbol{x}_i, \boldsymbol{C}_{i-1}\{\boldsymbol{x}\}) \le \delta$ **then**
    $\boldsymbol{C}_i = \boldsymbol{C}_{i-1}, \boldsymbol{\alpha}_{i,j^*} = \boldsymbol{\alpha}_{i-1,j^*} + \eta e_i$
  else
    $\boldsymbol{C}_i = \{\boldsymbol{C}_{i-1}, \boldsymbol{x}_i\}, \boldsymbol{\alpha}_i = [\boldsymbol{\alpha}_{i-1}, \eta e_i]$
  end if
end while

2) **QKLMS Algorithm Trained with MCC**

The main difference between the QKLMS algorithm trained with MSE and MCC lies in the methods of coefficient updates. For QKLMS-MSE, we calculate the ηe_i form of the error as the coefficient update for each iteration while the QKLMS-MCC employs the Gaussian form $\frac{\eta}{\sigma_e^2}\kappa_e(e_i)e_i$ of the error as the coefficient update for each iteration [14].

TABLE VI
PSEUDOCODES OF IMPLEMENTING THE QKLMS ALGORITHM TRAINED WITH MCC

Algorithm 6: QKLMS Algorithm Trained with MCC

Input: $\boldsymbol{x}_n, d_n) = \big((x_n, x_{n-1}, \cdots, x_{n-L+1}), x_{n+1}\big), L = 6$
Output: the weight $\boldsymbol{\Omega} = \sum_k^K \alpha_k \kappa(\cdot, \boldsymbol{x}_k)$
Initialization: step size η, Gaussian kernel size σ, σ_e quantization threshold δ, center dictionary $\boldsymbol{C}_1 = \boldsymbol{C}_1\{\boldsymbol{x}_1\}$, coefficient vector $\boldsymbol{\alpha}_1 = [\eta d_1]$
While $\{(\boldsymbol{x}_i, d_i), i = 2, \cdots, N\}$ is available **do**
  $e_i = d_i - \sum_{j=1}^{K_{i-1}} \alpha_j \kappa\big(\boldsymbol{x}_i, \boldsymbol{C}_{i-1}\{\boldsymbol{x}_j\}\big)$
  $dist(\boldsymbol{x}_i, \boldsymbol{C}_{i-1}\{\boldsymbol{x}\}) = \min_{1 \le j \le K_{i-1}} \left\|\boldsymbol{x}_i - \boldsymbol{C}_{i-1}\{\boldsymbol{x}_j\}\right\|_2$
  $j^* = \arg\min_{1 \le j \le K_{i-1}} \left\|\boldsymbol{x}_i - \boldsymbol{C}_{i-1}\{\boldsymbol{x}_j\}\right\|_2$
  if $dist(\boldsymbol{x}_i, \boldsymbol{C}_{i-1}\{\boldsymbol{x}\}) \le \delta$ **then**
    $\boldsymbol{C}_i = \boldsymbol{C}_{i-1}, \boldsymbol{\alpha}_{i,j^*} = \boldsymbol{\alpha}_{i-1,j^*} + \frac{\eta}{\sigma_e^2}\kappa_e(e_i)e_i$
  else
    $\boldsymbol{C}_i = \{\boldsymbol{C}_{i-1}, \boldsymbol{x}_i\}, \boldsymbol{\alpha}_i = \left[\boldsymbol{\alpha}_{i-1}, \frac{\eta}{\sigma_e^2}\kappa_e(e_i)e_i\right]$
  end if
end while

C. Quantized Kernel Recursive Least Square (QKRLS) Algorithm

1) QKRLS Algorithm Trained with MSE

Compared with the KRLS algorithm, the QKRLS algorithm adopts a diagonal matrix $\mathbf{\Lambda}$ to track the center whose distance from the sample $\boldsymbol{x}_i$ is less than certain threshold and update the coefficient as follows [4]:

$$\mathbf{\Omega}_i = \mathbf{\Phi}_i[\lambda \boldsymbol{I} + \mathbf{\Lambda}_i \mathbf{\Phi}_i \mathbf{\Phi}_i^T]^{-1}\mathbf{d}_i = \mathbf{\Phi}_i \mathbf{Q}_i \mathbf{d}_i = \mathbf{\Phi}_i \mathbf{a}_i \quad (12)$$

TABLE VII
PSEUDOCODES OF IMPLEMENTING THE QKRLS ALGORITHM TRAINED WITH MSE

Algorithm 7: QKRLS Algorithm Trained with MSE

Input: $(\boldsymbol{x}_n, d_n) = \big((x_n, x_{n-1}, \cdots, x_{n-L+1}), x_{n+1}\big)$, $L = 6$
Output: the weight $\mathbf{\Omega} = \sum_k^K a_k \kappa(\cdot, \boldsymbol{x}_k)$
Initialization: forgetting factor λ, Gaussian kernel size σ, center dictionary $\boldsymbol{C}_1 = \boldsymbol{C}_1\{\boldsymbol{x}_1\}$, coefficient vector $\boldsymbol{Q}_1 = \left[\left(\lambda + \kappa(\boldsymbol{x}_1, \boldsymbol{x}_1)\right)^{-1}\right]$, $\boldsymbol{a}_1 = [\boldsymbol{Q}_1 d_1]$, $\mathbf{\Lambda}_1 = [1]$
While $\{(\boldsymbol{x}_i, d_i), i = 2, \cdots, N\}$ is available **do**
$\boldsymbol{h}_i = \left[\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j)\right]^T, \boldsymbol{x}_j \epsilon \boldsymbol{C}_{i-1}$
$e_i = d_i - \boldsymbol{h}_i^T \boldsymbol{a}_{i-1}$
$dist(\boldsymbol{x}_i, \boldsymbol{C}_{i-1}\{\boldsymbol{x}\}) = \min_{1 \le j \le K_{i-1}} \left\|\boldsymbol{x}_i - \boldsymbol{C}_{i-1}\{\boldsymbol{x}_j\}\right\|_2$
$j^* = \arg\min_{1 \le j \le K_{i-1}} \left\|\boldsymbol{x}_i - \boldsymbol{C}_{i-1}\{\boldsymbol{x}_j\}\right\|_2$
if $dist(\boldsymbol{x}_i, \boldsymbol{C}_{i-1}\{\boldsymbol{x}\}) \le \delta$ **then**
$\boldsymbol{C}_i = \boldsymbol{C}_{i-1}$
$\boldsymbol{\xi}_{j^*} = [0, \cdots, 1, \cdots, 0]^T$ where the j^*th element is 1, other elements are 0, and the size of $\boldsymbol{\xi}_{j^*}$ is equal to $\boldsymbol{C}_{i-1}$
$\mathbf{\Lambda}_i = \mathbf{\Lambda}_{i-1} + \boldsymbol{\xi}_{j^*} \boldsymbol{\xi}_{j^*}^T$
$\boldsymbol{h}_{j^*} = \left[\kappa\left(\boldsymbol{C}_{i-1}\{\boldsymbol{x}_{j^*}\}, \boldsymbol{x}_j\right)\right]^T, \boldsymbol{x}_j \epsilon \boldsymbol{C}_{i-1}$
$\boldsymbol{Q}_i = \boldsymbol{Q}_{i-1} - \frac{\boldsymbol{Q}_{i-1,j^*}\left[\boldsymbol{h}_{j^*}^T \boldsymbol{Q}_{i-1}\right]}{1 + \boldsymbol{h}_{j^*}^T \boldsymbol{Q}_{i-1,j^*}}$ where $\boldsymbol{Q}_{i-1,j^*}$ is the j^*th column of $\boldsymbol{Q}_{i-1}$.
$\boldsymbol{a}_i = \boldsymbol{a}_{i-1} + \frac{\boldsymbol{Q}_{i-1,j^*}\left[d_i - \boldsymbol{h}_{j^*}^T \boldsymbol{a}_{i-1}\right]}{1 + \boldsymbol{h}_{j^*}^T \boldsymbol{Q}_{i-1,j^*}}$
else
$\boldsymbol{C}_i = \{\boldsymbol{C}_{i-1}, \boldsymbol{x}_i\}$
$\mathbf{\Lambda}_i = \begin{bmatrix} \mathbf{\Lambda}_{i-1} & \mathbf{0} \\ \mathbf{0}^T & 1 \end{bmatrix}$
$\boldsymbol{z}_i = \boldsymbol{Q}_{i-1}^T \boldsymbol{h}_i$
$\boldsymbol{z}_{i,\Lambda} = \boldsymbol{Q}_{i-1} \mathbf{\Lambda}_{i-1} \boldsymbol{h}_i$
$r_i = \lambda + \kappa(\boldsymbol{x}_i, \boldsymbol{x}_i) - \boldsymbol{z}_{i,\Lambda}^T \boldsymbol{h}_i$
$\boldsymbol{Q}_i = \begin{bmatrix} \boldsymbol{Q}_{i-1} + \boldsymbol{z}_{i,\Lambda} r_i^{-1} \boldsymbol{z}_i^T & -\boldsymbol{z}_{i,\Lambda} r_i^{-1} \\ -\boldsymbol{z}_i^T r_i^{-1} & r_i^{-1} \end{bmatrix}$
$\boldsymbol{a}_i = \begin{bmatrix} \boldsymbol{a}_{i-1} - \boldsymbol{z}_{i,\Lambda} r_i^{-1} e_i \\ r_i^{-1} e_i \end{bmatrix}$
end if
end while

2) QKRLS Algorithm Trained with MCC

When the minimal distance between the sample $\boldsymbol{x}_i$ and center dictionary $\boldsymbol{C}_{i-1}$ is less than certain threshold, the QKRLS-MCC algorithm shares the same method of coefficient updates as the QKRLS-MSE algorithm. When the minimal distance is higher than the threshold, we introduce the correntropy $\kappa_e(e_i)$ of the error to replace the forgetting factor λ with a multiplication of $\lambda \kappa_e(e_i)$ that can compact the error distribution [10].

TABLE VIII
PSEUDOCODES OF IMPLEMENTING THE QKRLS ALGORITHM TRAINED WITH MCC

Algorithm 8: QKRLS Algorithm Trained with MCC

Input: $(\boldsymbol{x}_n, d_n) = \big((x_n, x_{n-1}, \cdots, x_{n-L+1}), x_{n+1}\big)$, $L = 6$
Output: the weight $\mathbf{\Omega} = \sum_k^K a_k \kappa(\cdot, \boldsymbol{x}_k)$
Initialization: forgetting factor λ, Gaussian kernel size σ, σ_e, $g_1 = e^{-\frac{\|d_1\|^2}{\sigma_e^2}}$ center dictionary $\boldsymbol{C}_1 = \boldsymbol{C}_1\{\boldsymbol{x}_1\}$, coefficient vector $\boldsymbol{Q}_1 = \left[\left(\lambda \mathrm{g}_1 + \kappa(\boldsymbol{x}_1, \boldsymbol{x}_1)\right)^{-1}\right]$, $\boldsymbol{a}_1 = [\boldsymbol{Q}_1 d_1]$, $\mathbf{\Lambda}_1 = [1]$
While $\{(\boldsymbol{x}_i, d_i), i = 2, \cdots, N\}$ is available **do**
$\boldsymbol{h}_i = \left[\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j)\right]^T, \boldsymbol{x}_j \epsilon \boldsymbol{C}_{i-1}$
$e_i = d_i - \boldsymbol{h}_i^T \boldsymbol{a}_{i-1}$
$dist(\boldsymbol{x}_i, \boldsymbol{C}_{i-1}\{\boldsymbol{x}\}) = \min_{1 \le j \le K_{i-1}} \left\|\boldsymbol{x}_i - \boldsymbol{C}_{i-1}\{\boldsymbol{x}_j\}\right\|_2$
$j^* = \arg\min_{1 \le j \le K_{i-1}} \left\|\boldsymbol{x}_i - \boldsymbol{C}_{i-1}\{\boldsymbol{x}_j\}\right\|_2$
if $dist(\boldsymbol{x}_i, \boldsymbol{C}_{i-1}\{\boldsymbol{x}\}) \le \delta$ **then**
$\boldsymbol{C}_i = \boldsymbol{C}_{i-1}$
$\boldsymbol{\xi}_{j^*} = [0, \cdots, 1, \cdots, 0]^T$ where the j^*th element is 1, other elements are 0, and the size of $\boldsymbol{\xi}_{j^*}$ is equal to $\boldsymbol{C}_{i-1}$
$\mathbf{\Lambda}_i = \mathbf{\Lambda}_{i-1} + \boldsymbol{\xi}_{j^*} \boldsymbol{\xi}_{j^*}^T$
$\boldsymbol{h}_{j^*} = \left[\kappa\left(\boldsymbol{C}_{i-1}\{\boldsymbol{x}_{j^*}\}, \boldsymbol{x}_j\right)\right]^T, \boldsymbol{x}_j \epsilon \boldsymbol{C}_{i-1}$
$\boldsymbol{Q}_i = \boldsymbol{Q}_{i-1} - \frac{\boldsymbol{Q}_{i-1,j^*}\left[\boldsymbol{h}_{j^*}^T \boldsymbol{Q}_{i-1}\right]}{1 + \boldsymbol{h}_{j^*}^T \boldsymbol{Q}_{i-1,j^*}}$ where $\boldsymbol{Q}_{i-1,j^*}$ is the j^*th column of $\boldsymbol{Q}_{i-1}$.
$\boldsymbol{a}_i = \boldsymbol{a}_{i-1} + \frac{\boldsymbol{Q}_{i-1,j^*}\left[d_i - \boldsymbol{h}_{j^*}^T \boldsymbol{a}_{i-1}\right]}{1 + \boldsymbol{h}_{j^*}^T \boldsymbol{Q}_{i-1,j^*}}$
else
$\boldsymbol{C}_i = \{\boldsymbol{C}_{i-1}, \boldsymbol{x}_i\}$
$\mathbf{\Lambda}_i = \begin{bmatrix} \mathbf{\Lambda}_{i-1} & \mathbf{0} \\ \mathbf{0}^T & 1 \end{bmatrix}$
$\boldsymbol{z}_i = \boldsymbol{Q}_{i-1}^T \boldsymbol{h}_i$
$\boldsymbol{z}_{i,\Lambda} = \boldsymbol{Q}_{i-1} \mathbf{\Lambda}_{i-1} \boldsymbol{h}_i$
$g_i = e^{-\frac{\|e_i\|^2}{\sigma_e^2}}$
$r_i = \lambda g_i + \kappa(\boldsymbol{x}_i, \boldsymbol{x}_i) - \boldsymbol{z}_{i,\Lambda}^T \boldsymbol{h}_i$
$\boldsymbol{Q}_i = \begin{bmatrix} \boldsymbol{Q}_{i-1} + \boldsymbol{z}_{i,\Lambda} r_i^{-1} \boldsymbol{z}_i^T & -\boldsymbol{z}_{i,\Lambda} r_i^{-1} \\ -\boldsymbol{z}_i^T r_i^{-1} & r_i^{-1} \end{bmatrix}$
$\boldsymbol{a}_i = \begin{bmatrix} \boldsymbol{a}_{i-1} - \boldsymbol{z}_{i,\Lambda} r_i^{-1} e_i \\ r_i^{-1} e_i \end{bmatrix}$
end if
end while

D. Convergence Analysis and Computational Complexity

Using the stochastic gradient method, the LMS, KLMS, and KRLS algorithms trained with MSE and MCC are presented in Table I-VII, respectively, and their convergence analysis have

been derived in the literature [3-5, 10, 12, 14]. Here, we focus on the computational complexity of each adaptive algorithm, as shown in Table IX, where K denotes the network size and L refers to the order of filter. For LMS, the overall cost is $O(L)$; For KLMS, the overall cost is $O(K)$; For KRLS, the overall cost is $O(K^2)$. The cost for updating the weights $\boldsymbol{\alpha}$ (KLMS) and $\boldsymbol{\alpha}$ (KRLS) is $O(K)$, as well as the online VQ. However, the cost for updating the $\boldsymbol{Q}$ (KRLS) is $O(K^2)$.

TABLE IX
COMPARISON OF COMPUTATIONAL COSTS AMONG ALGORITHMS

Algorithm	Cost Function	Computational Cost
LMS	MSE	$O(L)$
	MCC	$O(L)$
KLMS	MSE	$O(K)$
	MCC	$O(K)$
KRLS	MSE	$O(K^2)$
	MCC	$O(K^2)$

IV. SIMULATION RESULTS

We test the performance of QKLMS and QKRLS algorithms trained with MSE and MCC in the online time series prediction. Specifically, we use the first 200 samples of the sunspot time series data as the training set and the reminder as the testing set to characterize the performances of different algorithms.

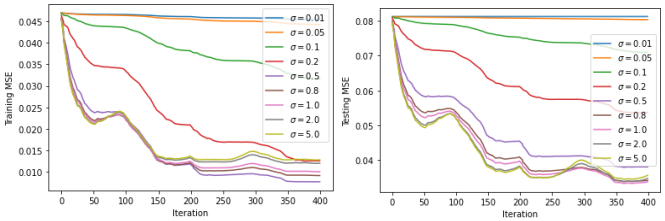
A. Performance of Algorithms in Sunspot Time Series

To guarantee the convergence of each algorithm, we repeat using the training set (the first 200 samples) twice to make the MSE converge to a small value and achieve a good prediction performance.

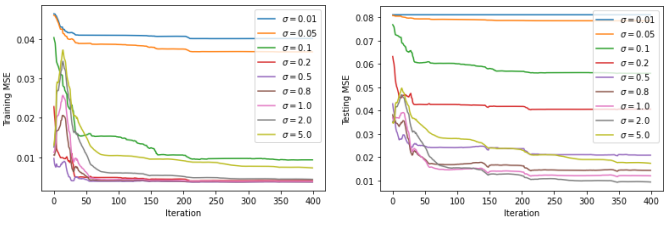
1) Effect of Hyperparameters in Kernel Methods

a. Kernel size in QKLMS and QKRLS algorithms

We vary the kernel size of QKLMS and QKRLS algorithms from 0.01 to 5 to test the effect of the kernel size, as shown in Fig. 1. For QKLMS, when the kernel size rises from 0.01 to 0.5, the training MSE shows a decline, and a further increase in the kernel size brings a rise in the training MSE whereas the testing MSE reaches the lowest when the kernel size is 1.0. Similarly, when the kernel size of QKRLS is 2.0, the testing MSE reaches the minimum.



(a) (b)

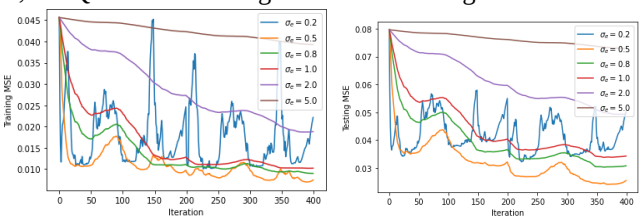


(c) (d)

Fig. 1 Learning curve ($\delta = 0.2$) by varying the kernel size. (a) QKLMS, training MSE. (b) QKLMS, testing MSE. (c) QKRLS, training MSE. (4) QKRLS, testing MSE

b. Kernel size in MCC

We vary the kernel size in MCC from 0.2 to 5.0 and find that when the kernel size is 0.5, the training and testing MSE both reach the minimum. When the kernel size is less than 0.5, the QKLMS-MCC cannot converge; When the kernel size is above 0.5, the QKLMS-MCC algorithm shows a higher MSE.

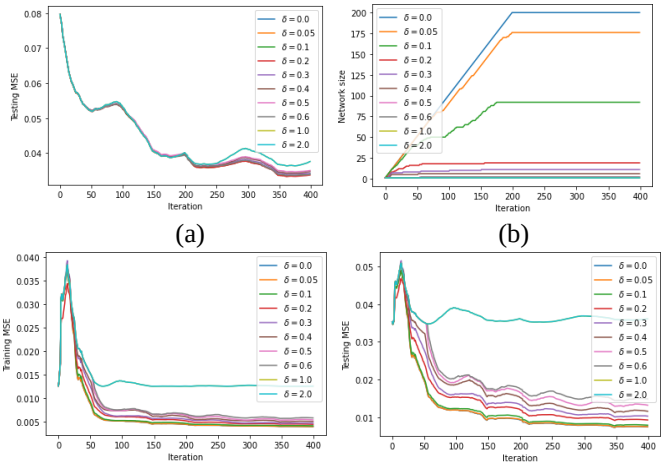


(a) (b)

Fig. 2 Learning curve by varying the kernel size in MCC. (a) QKLMS, training MSE. (b) QKLMS, testing MSE.

c. Quantization size in online VQ algorithm

We vary the quantization size of QKLMS and QKRLS from 0 to 2 and observe that the quantization size hardly affects the training (not shown) and testing MSE of QKLMS except when $\delta \geq 1$. For QKRLS, as the quantization size rises, the training and testing MSE gradually increase, as well as the network size. To balance the computational costs and testing MSE, we set the quantization size as 0.05.



(a) (b)

(c) (d)

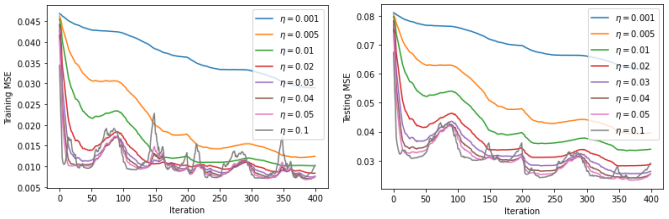
Fig. 3 Learning curve and network size by varying the quantization size. (a) QKLMS, testing MSE. (b) Network size. (c) QKRLS, training MSE. (d) QKRLS, testing MSE.

d. Step size in QKLMS algorithm

To guarantee the convergence of the QKLMS algorithm, the step size η should satisfy [15]:

$$0 < \eta < \frac{1}{\lambda_{max}} < \frac{1}{L\|\boldsymbol{x}\|^2} \tag{13}$$

Where L denotes the order of filter, $\|\boldsymbol{x}\|^2$ refers to the signal power, and λ_{max} means the max eigenvalue. To study the effect of step size on the filter performance, we vary the step size from 0.001 to 0.1 and find the best step size is 0.01 that exhibits lower and stable training and testing MSE.

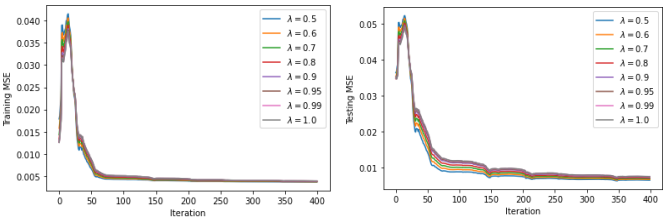


(a) (b)

Fig. 4 Learning curve by varying the step size in QKLMS. (a) Training MSE. (b) Testing MSE.

e. Forgetting factor in QKRLS algorithm

We vary the forgetting factor of QKRLS from 0.5 to 1 and find that the forgetting factor hardly shows significant impacts on the training or testing MSE. Here, we set the forgetting factor as 1.



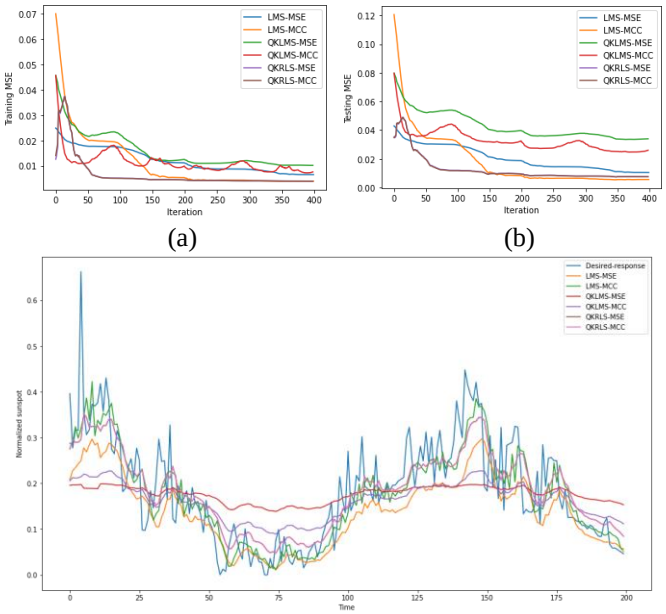
(a) (b)

Fig. 5 Learning curve by varying the forgetting factor in QKRLS. (a) Training MSE. (b) Testing MSE.

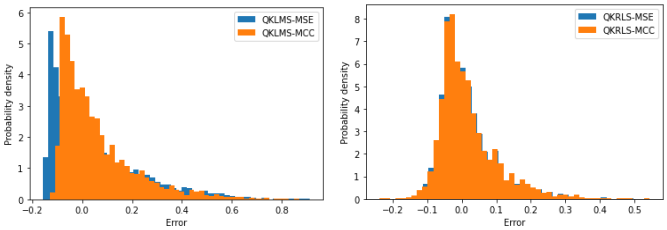
In summary, the best hyperparameters are listed below: kernel size $\sigma = 1$ in QKLMS and $\sigma = 2$ in QKRLS, gaussian kernel size $\sigma_e = 0.5$ in MCC, quantization size $\delta = 0.05$, step size $\eta = 0.01$, forgetting factor $\lambda = 1$.

2) **Performance Comparison of Different Algorithms**

We compare the performance of each algorithm and find that QKRLS-MCC achieves the best performance, followed by the QKRLS-MSE, LMS-MCC, LMS-MSE, QKLMS-MCC, and QKLMS-MSE, in terms of convergence speed, training and testing MSE and fitting curve between desired response and filter output. Furthermore, the cost function of MCC outperforms the MSE, which can be confirmed by the error distributions on the testing set that MCC generates more concentrated error peaks and smaller errors than MSE.



(a) (b) (c)



(d) (e)

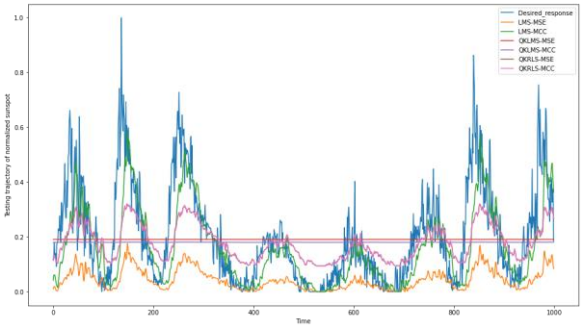
Fig. 6 Performance of each algorithm. (a) Learning curve of training MSE. (b) Learning curve of testing MSE. (c) Fitting curve on the training set. (d)-(e) Error distributions of QKLMS and QKRLS trained with MSE and MCC

B. Prediction Performances of Sunspot Time Series

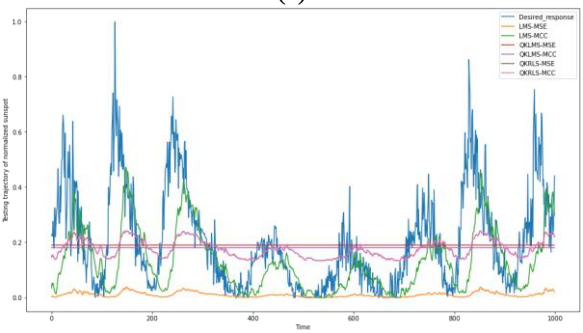
1) **Prediction of the Next 1st, 10th, and 20th Samples**

We compare MSE of each algorithm using one sample lag in predicting the next 1st, 10th, and 20th samples on the testing set and find that when predicting the next 1 sample, LMS-MCC and QKRLS achieves better performances than LMS-MSE and QKLMS. Then as the prediction step increases, the testing MSE of each algorithm all exhibits a gradual rise, particularly for QKRLS. The LMS-MCC shows better performances than other algorithms in predicting the next 10th sample. When predicting the next 20th sample, LMS-MCC and QKLMS-MSE achieve the equivalent performances and outperform others. However, it is worth noting that although QKLMS brings a lower MSE, it hardly fits the changing trend of the sunspot time series data. By contrast, the QKRLS algorithms basically fit the changing trend even though with a larger prediction error power.

When using a delay of 10 and 20 samples to predict the next 10th and 20th samples, QKRLS achieves the best performance, followed by LMS and QKLMS. This implies that the prediction method using a delay of 10 or 20 samples is more suitable for QKRLS than a means of using a recurrent structure. It is worth noting that the two methods of predicting the next 10th and 20th samples hardly affect the performance of LMS and QKLMS.



(a)



(b)

Fig. 7 Fitting curve of each trained model in predicting the next (a) 10th and (b) 20th samples using a delay of 1 sample

TABLE X

PREDICTION ERROR POWER OF ALGORITHMS FOR THE NEXT 1ST, 10TH, AND 20TH SAMPLES AHEAD USING A DELAY OF 1, 10, 20

Algorithm	z^{-1}			z^{-10}	z^{-20}	Time (Sec)
	z^{1}	z^{10}	z^{20}	z^{10}	z^{20}	
LMS-MSE	0.010	0.038	0.063	0.023	0.043	0.02
LMS-MCC	0.006	0.017	0.036	0.019	0.036	0.02
QKLMS-MSE	0.034	0.036	0.036	0.038	0.045	42.68
QKLMS-MCC	0.026	0.037	0.040	0.039	0.046	41.96
QKRLS-MSE	0.008	0.264	0.358	0.019	0.034	44.01
QKRLS-MCC	0.007	0.264	0.359	0.019	0.033	44.22

2) **Prediction of Sunspot Time Series Trajectories**

To generate a trajectory of the sunspot time series, we create a recurrent system into the trained models by the LMS, QKLMS, QKRLS algorithms. Here, we set 20 different initial conditions (various starting points of the sunspot time series) to separately predict the trajectory and computes average errors between the desired response and the predicted value. Also, we set an error threshold (1/3 of the standard deviation of the time series data) and count how many samples fall into the threshold, as shown in Fig. 8 and Table XI.

Fig. 8 reveals that the algorithms hardly generate the desired sunspot trajectory. Relatively, the LMS-MCC and LMS-MSE can generate better sunspot trajectory than others, followed by QKRLS and QKLMS. It can also be found in Table XI that the prediction steps within the error threshold have a small mean value and large standard deviation, showing that the algorithms perform badly in generating the sunspot trajectory and largely depend on the starting point of the trajectory generation.

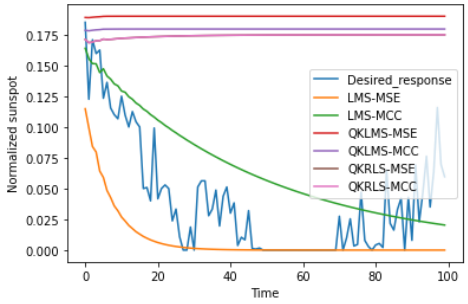


Fig. 8 Performance comparison of the predicted trajectories generated by each trained model.

TABLE XI

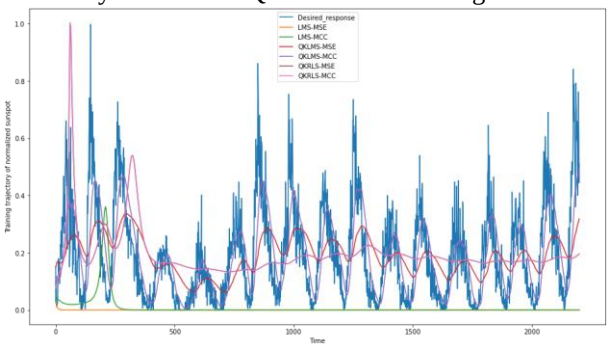
SUMMARY OF PREDICTION STEPS WITHIN ERROR THRESHOLD BY DIFFERENT TRAINED MODELS

Algorithm	Cost Function	Prediction Step
LMS	MSE	4.90±6.56
	MCC	5.65±7.03
QKLMS	MSE	1.35±0.79
	MCC	1.80±1.81
QKRLS	MSE	3.35±3.21
	MCC	3.35±3.21

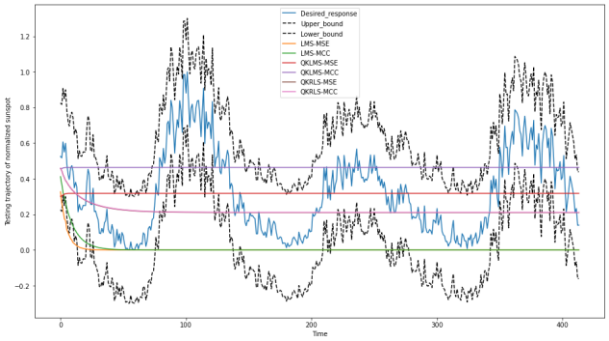
C. Iterative Prediction of Sunspot Time Series Trajectories

As an alternative, we use the delayed filter outputs rather than the samples as inputs to train the trained models again by LMS, KLMS and QKLMS until the error stabilizes at a small value, aiming to iteratively generate one desired sunspot trajectory. Similarly, we set a threshold of normalized error (0.3) to count how many consecutive samples predicted fall into the range on the testing set, as shown in Fig. 9 and Table XII.

Fig. 9 presents that the QKRLS and QKLMS can learn well the trajectory of the sunspot time series data on the training set, especially for QKLMS, whereas the LMS algorithms trained with MSE and MCC almost do not work. When generating the trajectory itself, the QKRLS algorithms trained with MSE and MCC both outperform other algorithms due to more samples falling into the error threshold range between upper bound and lower bound, followed by KLMS-MSE, KLMS-MCC, LMS-MCC, and LMS-MSE. This can be further reflected in Table XII that the prediction step of QKRLS shows larger horizons of predictability than that of QKLMS and LMS algorithms.



(a)



(b)

Fig. 9 Performance comparison of each trained model. (a) Learning the trajectory on the training set. (b) Generating the trajectory on the testing set where the upper and lower bounds denote the max and min error thresholds, respectively.

TABLE XII
SUMMARY OF PREDICTION STEPS WITHIN ERROR THRESHOLD BY EACH TRAINED MODEL ON LEARNING THE TRAJECTORY

Algorithm	Cost Function	Prediction Step
LMS	MSE	2
	MCC	22
QKLMS	MSE	43
	MCC	31
QKRLS	MSE	79
	MCC	79

V. CONCLUSIONS

In this project, we systematically compare the performance of quantized kernel least mean square (QKLMS) and kernel recursive least square (QKRLS) trained with mean square error (MSE) and maximum correntropy criterion (MCC) in predicting the next sample and trajectory of sunspot time series. Simulation results show that QKRLS algorithms outperform the QKLMS and baseline model (least mean square, LMS) when predicting the next sample in the sunspot time series, with lower training and testing MSE and faster convergence speed. However, the QKRLS performs badly in predicting the next 10^{th} and 20^{th} samples using a delay of one sample, worse than LMS and QKLMS, whereas the QKRLS performs better using a delay of 10 and 20 samples. While for cost function, the MCC performs better than the MSE in more concentrated error peaks around 0 and smaller errors between the desired responses and filter outputs.

When predicting the trajectory of the sunspot time series, we adopt two methods of using the trained models with a recurrent structure and training the trained models again to learn sunspot trajectory itself. The results suggest that the former cannot well predict the sunspot trajectory where the prediction steps within the error threshold have a smaller mean value and large standard deviation. For the latter, the QKRLS algorithms trained with MSE and MCC perform better than other algorithms, followed by the QKLMS and LMS algorithms, in terms of the horizon of predictability for generating the sunspot trajectory.

REFERENCES

1 Paul, T.K., and Ogunfunmi, T.: 'A kernel adaptive algorithm for quaternion-valued inputs', IEEE Trans Neural Netw Learn Syst, 2015, 26, (10), pp. 2422-2439
2 Wang, G., Yang, X., Wu, L., Fu, Z., Ma, X., He, Y., and Peng, B.: 'A kernel recursive minimum error entropy adaptive filter', Signal Processing, 2022, 193
3 Engel, Y., Mannor, S., and Meir, R.: 'The kernel recursive least-squares algorithm', Ieee T Signal Proces, 2004, 52, (8), pp. 2275-2285
4 Chen, B., Zhao, S., Zhu, P., and Principe, J.C.: 'Quantized kernel recursive least squares algorithm', IEEE Trans Neural Netw Learn Syst, 2013, 24, (9), pp. 1484-1491
5 Liu, W., Pokharel, P.P., and Principe, J.C.: 'The kernel least-mean-square algorithm', Ieee T Signal Proces, 2008, 56, (2), pp. 543-554
6 Liu, W., and Príncipe, J.C.: 'Kernel affine projection algorithms', EURASIP Journal on Advances in Signal Processing, 2008, 2008, (1)
7 Middleton, D.: 'Adaptive processing of underwater acoustic signals in non-Gaussian noise environments: I. Detection in the space-time threshold regimes', in Urban, H.G. (Ed.): 'Adaptive Methods in Underwater Acoustics' (Springer Netherlands, 1985), pp. 527-535
8 Liu, W., Park, I., and Principe, J.C.: 'An information theoretic approach of designing sparse kernel adaptive filters', IEEE Trans Neural Netw, 2009, 20, (12), pp. 1950-1961
9 Chen, B., Yuan, Z., Zheng, N., and Príncipe, J.C.: 'Kernel minimum error entropy algorithm', Neurocomputing, 2013, 121, pp. 160-169
10 Shen, T., Ren, W., and Han, M.: 'Quantized generalized maximum correntropy criterion based kernel recursive least squares for online time series prediction', Engineering Applications of Artificial Intelligence, 2020, 95
11 Platt, J.: 'A resource-allocating network for function interpolation', Neural Computation, 1991, 3, (2), pp. 213-225
12 Chen, B., Zhao, S., Zhu, P., and Principe, J.C.: 'Quantized kernel least mean square algorithm', IEEE Trans Neural Netw Learn Syst, 2012, 23, (1), pp. 22-32
13 Yuu-Seng Lau, Zahir M. Hussian, and Harris, R.: 'Performance of adaptive filtering algorithms: A comparative study', 2003
14 Wang, S., Zheng, Y., Duan, S., Wang, L., and Tan, H.: 'Quantized kernel maximum correntropy and its mean square convergence analysis', Digital Signal Processing, 2017, 63, pp. 164-176
15 Principe, J.C.: 'Kernel filtering: KLMS, KRLS, and QKLMS algorithms', 2022