A Method for Improving the Convergence Characteristics of the Kernel LMS Algorithm based on the Repeating Method

Kiyoshi NISHIKAWA, Yoshiki OGAWA, and Koji MAKIZAKI
Dept. of Information and Communications Systems, Tokyo Metropolitan University
6-6 Asahigaoka, Hino-shi, Tokyo 191-0065 JAPAN
E-mail: knishikawa@m.ieice.org Tel: +81-42-585-8423

***Abstract*—In this paper, we propose a method for improving the convergence characteristics of the kernel least mean square (KLMS), especially, kernel nomalized LMS (KNLMS) adaptive algorithm. The proposed method is based on the concept of the repeating method for the linear LMS adaptive algorithm which uses the information of the past input and desired signals. We derive a method for applying it to the KLMS algorithm, and propose an efficient implementation method. We confirm the effectiveness of the proposed method through the computer simulations.**

I. Introduction

In this paper, we propose a method for improving the convergence characteristics of the kernel least mean square (KLMS) adaptive algorithm[1], [2] based on the concept of the repeating method[3], [4].

Kernel adaptive filters enable us to estimate non-linear systems, and are expected to be used in applications such as non-linear channel equalization[1]. Kernel adaptive filters are derived by applying the kernel method to linear adaptive filtering theory[1]. For the learning of the kernel adaptive filters, algorithms have been proposed based on the ones for linear filters, e.g., kernel recursive least squares (KRLS), kernel affine projection algorithm (KAPA), KLMS, and so force[1], [2], [5], [6].

The characteristics of those kernel algorithms resemble the counterpart of the linear algorithm. Namely, the kernel LMS algorithm requires less computational complexity, but slower convergence characteristics compared with the kernel RLS. We, therefore, are required to improve the convergence characteristics of the kernel LMS algorithm to extend the area of applications where the kernel adaptive filters could be used.

In this paper, we propose a method for improving the convergence characteristics of the KLMS using the concept of the repeating method. The repeating method was proposed by Nagumo and Noda in the original paper of the normalized LMS (NLMS), or the leaning method[3]. In the repeating method, the past input and desired signals of the fixed length are stored to be reused at later time. Using these stored data, the adaptive filter will be updated several times at each time instead of one time in the normal configuration. It is shown that, by reusing the past data, the rate of convergence of the algorithm could be accelerated[4], [7].

We derive the repeating method for the KNLMS algorithms by slightly modifying the original for the linear adaptive filters. Then, we consider an efficient implementation of the proposed method. It is shown that we could reduce the required amount of calculations by storing the results of computation at the previous times instead of input signals. Through the computer simulations, we verify the effectiveness of the proposed method.

II. Preparation

Here, we describe the repeating method for the linear NLMS algorithm. Then, we briefly summarize the conventional kernel method and kernel NLMS.

A. Repeating method

First, we describe the concept of the repeating method. The method was proposed in the original paper of the NLMS, or the learning method by Nagumo and Noda[3]. Later, the method was redeveloped by other researchers, and the method was renamed as the data reusing method in their papers[7], [8]. The idea of the repeating method is to store the r past input and desired signals and reusing them at later time.

It is well known that, in the standard NLMS algorithm, the filter coefficient vector $\boldsymbol{w}(n)$ will be updated at each time n using the equation below.

$$\boldsymbol{w}(n) = \boldsymbol{w}(n-1) + \eta \frac{e(n)\boldsymbol{x}(n)}{\epsilon + ||\boldsymbol{x}(n)||^2} \tag{1}$$

where $\boldsymbol{w}(n)$, the coefficient vector of the adaptive filter, and $\boldsymbol{x}(n)$, the input vector, are defined as

$$\boldsymbol{w}(n) = [w_0(n) \quad w_1(n) \quad \dots w_{N-1}(n)]^T \tag{2}$$

$$\boldsymbol{x}(n) = [x(n) \quad x(n-1) \quad \dots x(n-N+1)]^T \tag{3}$$

where N shows the length of the filter $\boldsymbol{w}(n)$, and $\|\cdot\|$ show the Euclidean norm of a vector. The error signal $e(n)$ is defined as $e(n) = d(n) - \boldsymbol{w}^T(n)\boldsymbol{x}(n)$; η $(0 \leq \eta < 2)$ is the step size parameter of the NLMS; and ϵ is a small constant to prevent the divergence of the update term in case of $||\boldsymbol{x}(n)|| = 0$.

When we apply the repeating method, we store the r past input vectors and desired signals, namely,

$$\boldsymbol{x}(n-1), \dots, \boldsymbol{x}(n-r)$$
$$d(n), \dots, d(n-r).$$

At each time n, after the update of the filter $\boldsymbol{w}(n)$ using Eq. (1), $\boldsymbol{w}(n)$ will be updated using those stored past signals. Let us express the filter coefficient vector after i-th iteration at time n as $\boldsymbol{w}(n,i)$. Then, we modify the update equation of the NLMS algorithm as,

$$\begin{aligned}\text{For } i = 0 \text{ to } r\\ \boldsymbol{w}(n,i) =& \boldsymbol{w}(n,i-1)\\ &+ \eta\frac{e(n-i)\boldsymbol{x}(n-i)}{\epsilon + ||\boldsymbol{x}(n-i)||^2}\end{aligned} \tag{4}$$

where $e(n-i)$ is defined as

$$e(n-i) = d(n-i) - \boldsymbol{w}^T(n,i-1)\boldsymbol{x}(n-i) \tag{5}$$

and we set $\boldsymbol{w}(n,-1) = \boldsymbol{w}(n-1,r)$. Note that this formula includes Eq. (1) as a case of $i = 0$ and by letting $r = 0$, the algorithm is reduced to the NLMS.

Using the repeating method, we could update the filter r times at each time. It is shown that we could increase the rate of convergence by the repeating method[3], [4]

B. Kernel method

Next, we briefly describe the kernel adaptive filtering[1]. In this case, the input signal $x(n)$ is transformed into a high-dimensional feature space F. By denoting the transformation to the space Fas $\Phi(\cdot)$, the output signal of the adaptive filter is expressed as

$$f\left(\boldsymbol{x}(n)\right) = \Phi^T\left(\boldsymbol{x}(n)\right)\boldsymbol{w}(n). \tag{6}$$

Here, let us assume that the filter vector $\boldsymbol{w}(n)$ can be expressed as a linear combination of m training vectors $\Phi(\boldsymbol{y}(j))$ as

$$\boldsymbol{w}(n) = \sum_{j=1}^{m}\alpha_j\Phi\left(\boldsymbol{y}(j)\right). \tag{7}$$

The vectors $\boldsymbol{y}(j)$ are subset of $\boldsymbol{x}(\ell)$ $(\ell = 0,1,\ldots,n-1)$ and the detail will be described in the next sub-section, and α_j is the weight corresponding to $\boldsymbol{y}(j)$. Then, the output in (6) is expressed[1] as

$$f\left(\boldsymbol{x}(n)\right) = \sum_{j=1}^{m}\left(\Phi^T\left(\boldsymbol{x}(n)\right)\Phi\left(\boldsymbol{y}(j)\right)\right)\alpha_j. \tag{8}$$

By defining $\boldsymbol{\alpha}$ as $\boldsymbol{\alpha} = [\alpha_1,\ldots,\alpha_m]^T$, $\boldsymbol{\alpha}$ could be regarded as a new set of coefficients of the filter. The kernel adaptive algorithms are derived[2], [9] to estimate the optimum $\boldsymbol{\alpha}$.

To estimate $\boldsymbol{\alpha}$, we should calculate the inner product $\Phi^T(\boldsymbol{x}(n))\Phi(\boldsymbol{y}(j))$ in Eq. (8). The kernel adaptive filters use the kernel function to calculate the inner product. A kernel function $k(\cdot,\cdot)$ is given as

$$\forall\mathbf{a},\mathbf{b}\in X \qquad k\left(\boldsymbol{a},\boldsymbol{b}\right) = \Phi^T(\mathbf{a})\Phi\left(\mathbf{b}\right) \tag{9}$$

and is used to calculate the inner product in the space F. This method is referred to as the kernel trick[1]. Note that, we do not need to know the transformed vectors $\Phi(\boldsymbol{a})$, or $\Phi(\boldsymbol{b})$ themselves to calculate the inner product.

In the following, we use the Gaussian kernel defined as below that is widely used in the kernel adaptive filters[2], [9].

$$\kappa\left(\boldsymbol{a},\boldsymbol{b}\right) = \exp\left(-\left\|\boldsymbol{a}-\boldsymbol{b}\right\|^2/2\beta_0^2\right) \tag{10}$$

where β_0 is called the kernel bandwidth.

C. Kernel adaptive filter

By applying the kernel method to the linear adaptive filters, the concept of the kernel adaptive filter is derived[1], [2]. The description below, and also our proposed method, are based on the method for designing sparse kernel adaptive filters proposed in [2].

First, we rewrite Eq. (8) as

$$\begin{aligned}f\left(\boldsymbol{x}(n)\right) =& \begin{bmatrix}\Phi\left(\boldsymbol{x}(n)\right)^T\Phi\left(\mathbf{y}(1)\right)\\ \Phi\left(\boldsymbol{x}(n)\right)^T\Phi\left(\mathbf{y}(2)\right)\\ \vdots\\ \Phi\left(\boldsymbol{x}(n)\right)^T\Phi\left(\mathbf{y}(m)\right)\end{bmatrix}^T\boldsymbol{\alpha} = \begin{bmatrix}k\left(\boldsymbol{x}(n),\boldsymbol{y}(1)\right)\\ k\left(\boldsymbol{x}(n),\boldsymbol{y}(2)\right)\\ \vdots\\ k\left(\boldsymbol{x}(n),\boldsymbol{y}(m)\right)\end{bmatrix}^T\boldsymbol{\alpha}\\ =& \boldsymbol{h}(n)\boldsymbol{\alpha}\end{aligned} \tag{11}$$

where we set $\boldsymbol{h}(n)$ as

$$\boldsymbol{h}(n) = \left[\kappa\left(\boldsymbol{x}(n),\boldsymbol{y}(1)\right),\ldots,\kappa\left(\boldsymbol{x}(n),\boldsymbol{y}(m)\right)\right]^T. \tag{12}$$

Then, the filter $\boldsymbol{\alpha}$ can be updated using a linear adaptive algorithm by regarding $\boldsymbol{h}(n)$ as the input vectors to $\boldsymbol{\alpha}$[1].

Here, we define the matrix D as

$$\mathbf{D} = \begin{bmatrix}\boldsymbol{y}(1) & \ldots & \boldsymbol{y}(m)\end{bmatrix} \tag{13}$$

and $\boldsymbol{D}$ is called the dictionary[2]. The vectors stored in the dictionary $\boldsymbol{D}$ are m $(m \le n)$ past input vectors $\boldsymbol{x}_\ell$ where m is a variable determined by the algorithm below, and, in general, m increases as n.

Let us denote $\boldsymbol{D}$ at time n by $\boldsymbol{D}_n$. Then, $\boldsymbol{D}_n$ and $\boldsymbol{h}(n)$ are updated according to the following pseudo algorithm:

Initialization

$$\begin{aligned}&\boldsymbol{D}_1 = \boldsymbol{y}(1) = \boldsymbol{x}(1)\\ &\boldsymbol{h}_1 = k\left(\boldsymbol{x}(1),\boldsymbol{y}(1)\right)\\ &\boldsymbol{\alpha}(1) = 0 \quad , \quad m = 1\end{aligned}$$

for $n = 2,3,\cdots$

if $\max_{j=1,\ldots,m}|\kappa\left(\boldsymbol{x}(n),\boldsymbol{y}(j)\right)| > \mu_0$ (14)

$$\boldsymbol{D}_n = \boldsymbol{D}_{n-1}$$

$$\boldsymbol{h}_n = \begin{bmatrix}\kappa\left(\boldsymbol{x}(n),\boldsymbol{y}(1)\right) & \cdots & k\left(\boldsymbol{x}(n),\boldsymbol{y}(m)\right)\end{bmatrix}^T \tag{15}$$

else

$$\begin{aligned}&m = m+1\\ &\boldsymbol{D}_n = \boldsymbol{D}_{n-1}\cup\{\boldsymbol{x}(n)\}\\ &\boldsymbol{h}_n = \begin{bmatrix}\kappa\left(\boldsymbol{x}(n),\boldsymbol{y}(1)\right) & \cdots & \kappa\left(\boldsymbol{x}(n),\boldsymbol{y}(m)\right)\end{bmatrix}^T\end{aligned} \tag{16}$$

end if

end for

In Eq. (14), μ_0 is a threshold in the range $0 < \mu_0 < 1$ and its value is determined according to the sparseness of the filter.

The input vector $\boldsymbol{x}(n)$ will be compared with the vectors in $\boldsymbol{D}_n$ by the condition shown in Eq. (14). If the condition met, $\boldsymbol{x}(n)$ will be stored in $\boldsymbol{D}_n$ as a new training vector.

D. Kernel NLMS algorithm

For updating $\boldsymbol{\alpha}$, several algorithms based on the ones for linear adaptive filters were proposed, namely, NLMS[2], RLS[9], ERLS-DCD[6] and so on. We consider the kernel NLMS algorithm in this paper.

In the kernel NLMS algorithm[2], the filter coefficient vector $\boldsymbol{\alpha}(n)$ is updated according to

$$\begin{aligned}\boldsymbol{\alpha}(n) =&\boldsymbol{\alpha}(n-1)\\ &+\frac{\eta}{\epsilon+||\boldsymbol{h}(n)||^2}\left(d(n)-\boldsymbol{h}^T(n)\boldsymbol{\alpha}(n-1)\right)\boldsymbol{h}(n)\end{aligned} \quad (17)$$

where $\boldsymbol{h}(n)$ is given as

$$\boldsymbol{h}(n)=\begin{bmatrix}\kappa\left(\boldsymbol{x}(n),\boldsymbol{y}(1)\right) & \cdots & \kappa\left(\boldsymbol{x}(n),\boldsymbol{y}(m)\right)\end{bmatrix}^T. \quad (18)$$

From the equation, we can see that the update formula for the kernel NLMS is almost identical to that of the linear NLMS. However, we should remind that the length of the adaptive filter will become longer as adaptation progress.

III. Proposed method

Let us consider applying the repeating method to the kernel NLMS algorithm. Then, we notice that we could not directly apply the method to the KNLMS because of the variation of the length of the filter coefficients. Besides, the direct implementation requires the large amount of computation for calculating the kernel functions repeatedly. Hence, we consider an efficient implementation method.

A. Repeating method for the kernel LMS algorithm

For deriving the repeating method for the kernel NLMS algorithm, we consider to store the signals of the previous time at each time n as in the repeating method for the linear adaptive filters. Namely, we store r signals, i.e.,

$$\begin{aligned}&\boldsymbol{x}(n-1),\ldots,\boldsymbol{x}(n-r)\\ &\quad d(n),\ldots,d(n-r)\end{aligned} \quad (19)$$

For the description of the proposed method, we redefine the error signal as

$$e(n,i)=d(n-i)-\boldsymbol{h}^T(n-i)\boldsymbol{\alpha}(n,i-1) \quad (20)$$

where $\boldsymbol{\alpha}(n,i)$ shows the filter coefficient vector of i-th iteration at time n. We set $\boldsymbol{\alpha}(n,-1)$ as the last state of the filter coefficient at time $n-1$, i.e.,

$$\boldsymbol{\alpha}(n,-1)=\boldsymbol{\alpha}(n-1,r). \quad (21)$$

By letting $r=0$, this selection makes the formula of the repeating method to that of the normal NLMS, or KNLMS without the repetition. Besides, we use $\boldsymbol{h}(n-i)$ whose expression is given as

$$\boldsymbol{h}(n-i)=[\kappa\left(\boldsymbol{x}(n-i),\boldsymbol{y}(1)\right)\cdots\kappa\left(\boldsymbol{x}(n-i),\boldsymbol{y}(m)\right)]^T. \quad (22)$$

Using the stored data, the filter coefficient vector $\boldsymbol{\alpha}(n,i)$ will be update r times at n namely,

For i = 0 to r

$$\begin{aligned}\boldsymbol{\alpha}(n,i) =&\boldsymbol{\alpha}(n,i-1)\\ &+\frac{\eta}{\epsilon+||\boldsymbol{h}(n-i)||^2}e(n,i)\boldsymbol{h}(n-i)\end{aligned} \quad (23)$$

However, known from Eq. (22), we need to calculate m kernel functions at each iteration, and these calculation are rather heavy computational load. Therefore, we should consider an efficient implementation method next.

B. Proposed implementation method

Here, let us consider an implementation method of the repeating method for the KNLMS described in Sec. III-A.

For implementing the method, we should calculate the kernel functions to obtain $\boldsymbol{h}(n)$ in Eq. (22). The amount of calculation increases as the number of repeat r increases. Hence, we consider an efficient way to implement the method.

Let us consider to store the past $\boldsymbol{h}(n)$ instead of $\boldsymbol{x}(n)$ and reuse it for updating. However, there arise a problem as below.

For simplicity, we show the case of $\boldsymbol{h}(n-1)$, i.e.,

$$\boldsymbol{h}(n-1)=[\kappa\left(\boldsymbol{x}(n-1),\boldsymbol{y}(1)\right)\cdots\kappa\left(\boldsymbol{x}(n-1),\boldsymbol{y}(m)\right)]^T. \quad (24)$$

We should note that the value of m, the length of the filter, at time n could be different from that at time $n-1$. Hence, the following two cases should be distinguished, namely,

1) m increased at time $n-1$ because $\boldsymbol{x}(n-1)$ was added to the dictionary.
2) m remains the same as that of time $n-1$, or $\boldsymbol{x}(n-1)$ was not added to the dictionary.

When the case 2) above, all the elements of $\boldsymbol{h}(n-1)$ were calculated at time $n-1$, and hence, we can reuse the vector at time n.

On the other hand, if m increased at time $n-1$, then the last term of $\boldsymbol{h}(n-1)$, namely, $\kappa(\boldsymbol{x}(n-1),\boldsymbol{y}(m))$ was not calculated previously. This means that, when m increases, we can reuse it at time n, by calculating and adding the term $\kappa(\boldsymbol{x}(n-1),\boldsymbol{y}(m))$.

In general, by expressing the vector as $\hat{\boldsymbol{h}}(i)$, it is defined as

$$\hat{\boldsymbol{h}}(i)=\begin{cases}[\boldsymbol{h}(n-i)] & m \text{ not increased}\\ [\boldsymbol{h}(n-i)\ \ \kappa(\boldsymbol{x}(n-1),\boldsymbol{y}(m))] & m \text{ increased}\end{cases} \quad (25)$$

By storing the vectors $\hat{\boldsymbol{h}}(i)$ and reusing at n, we could reduce the amount of calculation to implement the repeating method for the kernel NLMS algorithm.

We denote the stored values

$$\boldsymbol{H}=\begin{bmatrix}\hat{\boldsymbol{h}}(1) & \hat{\boldsymbol{h}}(2) & \cdots & \hat{\boldsymbol{h}}(r)\end{bmatrix}^T \quad (26)$$

and

$$\boldsymbol{d}=\begin{bmatrix}d(n-1) & d(n-2) & \cdots & d(n-r)\end{bmatrix}^T \quad (27)$$

TABLE I
COMPARISON OF COMPUTATIONAL COST PER ITERATION OF KRLS, KNLMS AND THE PROPOSED METHOD. IN THE PROPOSED METHOD r SHOWS THE NUMBER OF REPEAT.

	KRLS	KNLMS	Proposed
$\times$	$4m^2+4m$	$3m+1$	$(3m+1)r$
$+$	$4m^2+4m+1$	$3m$	$(3m)r$
kernel	M	M	M

Using the stored data in $\boldsymbol{H}$ and $\boldsymbol{d}$, we update the filter coefficients according to

$$\begin{aligned} &\text{For } i = 0 \text{ to } r \\ &\quad \boldsymbol{\alpha}(n,i) = \boldsymbol{\alpha}(n,i-1) \\ &\qquad + \frac{\eta}{\epsilon + ||\hat{\boldsymbol{h}}(i)||^2} e(n,i)\hat{\boldsymbol{h}}(i) \end{aligned} \tag{28}$$

at each time.

We show a comparison of the computational cost per one iteration of the KNLMS and the proposed method in Table I. The required calculation in the proposed method is proportional to the number of repeat$r$.

IV. SIMULATION RESULTS

Here, we show results of computer simulations using the proposed method to demonstrate its effectiveness.

A. Forward prediction

In the simulations, we generated the input signal using the equation[2]

$$\begin{aligned} x(n) = &\left(0.8 - 0.5\exp\left(-x(n-1)^{-2}\right)\right)x(n-1) \\ &- \left(0.3 + 0.9\exp\left(-x(n-1)^{2}\right)\right)x(n-2) \\ &+ 0.1\sin\left(x(n-1)\pi\right) \end{aligned} \tag{29}$$

and the initial values of x_{-1} and x_{-2} were given as random numbers of uniform distribution in the region $(0,1)$.

The order M of the input vector $\boldsymbol{x}(n)$ was set as four, and we added a white Gaussian noise of SNR 30dB to the signal $x(n)$. The results were evaluated in terms of the mean squared error (MSE) and the ensemble averages of 10000 independent trials are shown.

We compared the conventional KNLMS, KRLS, and the proposed method. For the proposed method, we set the number of repeat r as $r = 2$. The threshold value μ_0 in the KNLMS and the proposed methods were set as $\mu_0 = 0.8$, and the length of the signal was 1000.

The results are shown in Fig. 1. From the figure, we could confirm that the proposed method provides faster rate of convergence than that of the KLMS, and almost same as that of the KRLS.

Thus, we could confirm the effectiveness of the proposed method.

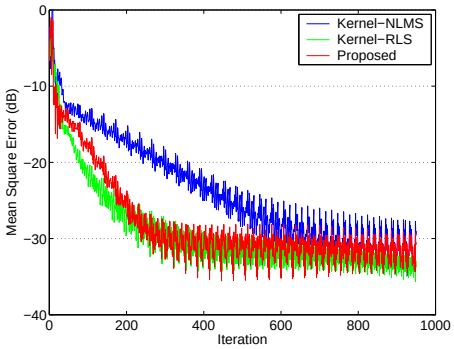

Fig. 1. Comparison of convergence characteristics of KNLMS, KRLS, and the proposed method. The proposed algorithm provides almost identical characteristics to that of the conventional KRLS although its computational load is lower.

B. Channel equalization

We also applied the proposed method for the channel equalization of the channel modeled as Rayleigh channel[5]. The parameters of the channel were set as the following. The number of paths M_{ch} were set as $M_{ch} = 2$, the maximum Doppler frequency f_D as $f_D = 100$ [Hz], sampling rate T_s as $T_s = 0.8$ [μ s], and signal length as 1000. We added a white Gaussian noise of SNR = 30[dB]. For simulating the non-linearity, the output of the channel was applied to $\tanh$ as[5]

$$d(n) = \tanh(\boldsymbol{x}(n) * \boldsymbol{w}_o') + v(n) \tag{30}$$

where $\boldsymbol{w}_o$ shows the optimum filter.

Again, the threshold value μ_0 was set as 0.8 for the KNLMS and the proposed method and r for the proposed as $r = 2$. The performance of the KLMS, KRLS, and the proposed methods were compared in terms of the MSE and the ensemble averages of 10000 independent trials are shown.

The results are shown in Fig.2. From the figure, we could confirm that the rate of convergence of the proposed method was almost same as that of the KRLS at the first stage of the learning. Besides, in the steady state, the mis-adjustment level was almost same of that of the KLMS algorithm.

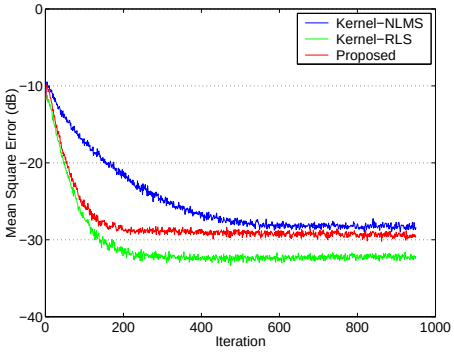

Fig. 2. Comparison of convergence characteristics of KNLMS, KRLS, and the proposed method for the Rayleigh channel estimation. The proposed algorithm provides better convergence characteristics than that of the KLMS algorithm.

V. Conclusion

In this paper, we proposed the repeating method for the kernel NLMS algorithm to improve the convergence characteristics. Through computational simulations, we confirmed that the proposed method could provide faster rate of convergence compared with that of the KLMS algorithm. Besides, it approaches to that of the KRLS algorithm under some conditions.

As a future work, we will consider the theoretical analysis of the proposed method.

References

[1] W. Liu, J. C. Principe, and S. Haykin, *Kernel Adaptive Filtering*. Wiley, 2010.

[2] C. Richard, J. C. M. Bermudez, and P. Honeine, "Online Prediction of Time Series Data With Kernels," *IEEE Transactions on Signal Processing*, vol. 57, pp. 1058–1067, Mar. 2009.

[3] J. Nagumo and A. Noda, "A learning method for system identification," *IEEE Transactions on Automatic Control*, vol. 12, pp. 282–287, June 1967.

[4] K. Nishikawa and H. Kiya, "A technique to improve convergence speed of the LMS algorithm," in *Proceedings of IEEE International Symposium on Circuits and Systems - ISCAS '94*, pp. 405–408, IEEE.

[5] W. Liu, I. M. Park, Y. Wang, and J. C. Principe, "Extended Kernel Recursive Least Squares Algorithm," *IEEE Transactions on Signal Processing*, vol. 57, pp. 3801–3814, Oct. 2009.

[6] Y. Ogawa and K. Nishikawa, "A Kernel Adaptive Filter based on ERLS-DCD Algorithm," in *Proc. of Intl Tech. Conf. Circuits Systems, Computer, Communications 2011*, (Gyeongju), 2011.

[7] A. Sayed, "Mean-square performance of data-reusing adaptive algorithms," *IEEE Signal Processing Letters*, vol. 12, pp. 851–854, Dec. 2005.

[8] R. Soni, K. Gallivan, and W. Jenkins, "Low-Complexity Data Reusing Methodsin Adaptive Filtering," *IEEE Transactions on Signal Processing*, vol. 52, pp. 394–405, Feb. 2004.

[9] Y. Engel, S. Mannor, and R. Meir, "The Kernel Recursive Least-Squares Algorithm," *IEEE Transactions on Signal Processing*, vol. 52, pp. 2275–2285, Aug. 2004.