Kernel least mean square algorithm with constrained growth

Puskal P. Pokharel *, Weifeng Liu, Jose C. Principe

Computational NeuroEngineering Laboratory, ECE Department, University of Florida, Gainesville, FL 32611, USA

ARTICLE INFO

Article history:
Received 9 December 2007
Received in revised form
18 August 2008
Accepted 18 August 2008
Available online 5 September 2008

Keywords:
Kernel least mean squares
Channel identification
Kernel recursive least squares

ABSTRACT

The linear least mean squares (LMS) algorithm has been recently extended to a reproducing kernel Hilbert space, resulting in an adaptive filter built from a weighted sum of kernel functions evaluated at each incoming data sample. With time, the size of the filter as well as the computation and memory requirements increase. In this paper, we shall propose a new efficient methodology for constraining the increase in length of a radial basis function (RBF) network resulting from the kernel LMS algorithm without significant sacrifice on performance. The method involves sequential Gaussian elimination steps on the Gram matrix to test the linear dependency of the feature vector corresponding to each new input with all the previous feature vectors. This gives an efficient method of continuing the learning as well as restricting the number of kernel functions used.

1. Introduction

In recent years, the machine learning community has shown considerable interest in kernel methods for classification and regression problems with numerous methods being proposed successfully in various applications. However, almost all the kernel methods (support vector machines, SVM [1], regularization networks [2], kernel principal component analysis, K-PCA [3]) are derived in batch mode. Most of these methods are not suitable for use in an on-line scenario where data are received one sample at a time. Some research has already been devoted in finding useful on-line methods [4,5]. The kernel least mean square (KLMS) algorithm [6] is probably the simplest to implement avoiding the need to utilize an explicit regularization that is a must in most kernel based algorithms. It can be shown [7] that this is a direct consequence of using the well-known least mean square (LMS) algorithm [8] in a very high dimensional space given by the kernel function.

LMS algorithm must be one of the most famous on-line methods. Introduced in the 1960s by Widrow and Hoff, the LMS algorithm [9] is the workhorse of adaptive signal processing due to its simplicity and applicability in a variety of signal processing applications. Simple manipulation of its weight update equation shows that this algorithm can be expressed solely in term of inner products, which by the famed kernel trick, allows us readily to construct a nonlinear version. We call this algorithm the KLMS [6]. Actually a similar leaky version has been introduced in [4] where a stochastic gradient approach is adopted to solve the regularized risk functional in a reproducing kernel Hilbert space (RKHS). Effectively, we demonstrated recently that KLMS is well-posed in the sense of Hadamard, and so explicit regularization which sacrifices performance is unnecessary [7]. Just like there is a linear combiner behind the LMS algorithm, we have to remember that behind the KLMS there is a radial basis function (RBF) network being adapted (when the kernel is a RBF). Training this RBF network with the KLMS is different from conventional RBF networks. First of all, for conventional RBF training, the kernel centers and their number have to be chosen...
heuristically or through a separate complex algorithm [10,11]. For our proposed approach, the centers and its number are chosen by the algorithm automatically during adaptation and do not need to be derived from a training set and hence it is truly online. Actually, the KLMS can readily be used in situations where a reliable training set is not known beforehand. Such examples include adaptive noise cancellation and equalization of a changing non-linear channels [7,12,13]. For instance, in [7,13], a set of real problems is solved by KLMS or its derivatives where our new approach can be readily used, hence making them practically more viable. Since the goal of this short paper is to present this new algorithm, kernel LMS with constrained growth (KLMS-CG) clearly but briefly, we shall not go into details of these applications. But a quick inspection obviates that our approach clearly solves one problem. The basic idea of kernel algorithms is to perform the linear LMS algorithm given by (2) in the kernel feature space. For this let us assume that $\Phi$ maps the point $x(n)$ in input space to $\Phi(x(n))$ in the kernel feature space with 

$$
\langle \Phi(x(n)), \Phi(x(m)) \rangle = \kappa(x(n), x(m)),
$$

where $\langle \cdot, \cdot \rangle$ represents the inner product in the kernel Hilbert space. This transformation to the feature space for most kernels is nonlinear, and depending on the choice of the kernel this space can be infinite dimensional. The Gaussian kernel that we use here will correspond to an infinite dimensional Hilbert space. Since this feature space is linear, $\Phi(x(n))$ can be considered an infinite dimensional column vector with usual vector inner products. Let $\Omega$ be the weight vector in this space such that the output is $y(n) = \langle \Omega(n), \Phi(x(n)) \rangle$. $\Omega(n)$ is $\Omega$ at time $n$. Let $d(n)$ be the desired response. Now, due to the linear structure of the RKHS cost function $J_{\Omega}(n) = E[(d(n) - y(n))^2]$ can be minimized with respect to $\Omega$. This can be done in the same way as done in (2) using the stochastic instantaneous estimate of the gradient vector, which yields 

$$
\Omega(n + 1) = \Omega(n) + 2\eta e(n)\Phi(x(n)).
$$

Like before, $\eta$ is the step-size parameter that controls the convergence, speed, and misadjustment of the adaptation algorithm [17,18]. The only catch here is that $\Omega$ in (4) is in the infinite dimensional feature space and it would be practically impossible to update for $\Omega$ directly. Instead we shall use (4) to relate each $\Omega(n)$ to its initialization $\Omega(0)$. This would easily give

$$
\Omega(n) = \Omega(0) + 2\eta \sum_{i=0}^{n-1} e(i)\Phi(x(i)).
$$

### 2. Background

#### 2.1. Kernel methods

In the past years, a number of kernel methods, including SVM [1], K-PCA [3], and kernel independent component analysis (K-ICA) [14] have been proposed and applied to machine learning and signal processing problems. The basic idea of kernel algorithms is to transform the data $x_i$ from the input space to a high dimensional feature space of vectors $\Phi(x_i)$, where the inner products can be computed using a positive definite kernel function satisfying Mercer’s conditions [1]:

$$
\kappa(x_i, x_j) = \langle \Phi(x_i), \Phi(x_j) \rangle.
$$

This simple idea allows us to obtain nonlinear versions of any linear algorithm expressed in terms of inner products, without even knowing the exact mapping $\Phi$.

A particularly interesting characteristic of the feature space is that it is a RKHS: i.e., the span of functions $\{\kappa(\cdot, \cdot) : \cdot \in \mathcal{F}\}$ defines a unique functional Hilbert space [15,16]. The crucial property of these spaces is the reproducing property of the kernel

$$
\phi(x) = \langle \kappa(\cdot, x), f \rangle, \quad \forall f \in \mathcal{F}.
$$

In particular, a nonlinear mapping from the input space to an RKHS can be defined as $\Phi(x) = \kappa(\cdot, x)$ such that

$$
\langle \Phi(x), \Phi(y) \rangle = \langle \kappa(\cdot, x), \kappa(\cdot, y) \rangle = \kappa(x, y),
$$

and thus $\Phi(x) = \kappa(\cdot, x)$ defines the Hilbert space associated with the kernel, and can be thought of as a nonlinear transformation from the input to feature space. Without loss of generality, in this paper we will only consider the translation-invariant radial basis (Gaussian) kernel, which is the most widely used Mercer kernel.

$$
\kappa(x, y) = \exp \left( -\frac{|x - y|^2}{2\sigma^2} \right).
$$

### 2.2. LMS algorithm

The LMS algorithm, introduced in 1960 by Widrow, is a very simple and elegant method of training a linear adaptive system to minimize mean square error. Given a quadratic cost function $J_w(n)$ (where $w$ is the tap-weight vector and $n$ is the time index), it can be shown that with exact measurements of the gradient vector $\nabla J_w(n)$ and a suitable chosen step-size parameter $\eta$, the weight vector updated by using the steepest-descent algorithm converges in the mean to the optimum Wiener solution. The LMS algorithm, instead of using the exact gradient to update the weight vector, uses the instantaneous estimate given by $\nabla J_w(n) = -2e(n)x(n)$ resulting in the following stochastic gradient descent update rule:

$$
w(n + 1) = w(n) + 2\eta e(n)x(n).
$$

The detailed analysis including that of convergence and misadjustment is given in [17] and is not the focus of this paper.

#### 2.3. Kernel LMS

In this section we present the KLMS algorithm. The basic idea is to perform the linear LMS algorithm given by (2) in the kernel feature space. Let us assume that $\Phi$ maps the point $x(n)$ in input space to $\Phi(x(n))$ in the kernel feature space with

$$
\langle \Phi(x(n)), \Phi(x(m)) \rangle = \kappa(x(n), x(m)),
$$

where $\langle \cdot, \cdot \rangle$ represents the inner product in the kernel Hilbert space. This transformation to the feature space for most kernels is nonlinear, and depending on the choice of the kernel this space can be infinite dimensional. The Gaussian kernel that we use here will correspond to an infinite dimensional Hilbert space. Since this feature space is linear, $\Phi(x(n))$ can be considered an infinite dimensional column vector with usual vector inner products. Let $\Omega$ be the weight vector in this space such that the output is $y(n) = \langle \Omega(n), \Phi(x(n)) \rangle$. $\Omega(n)$ is $\Omega$ at time $n$. Let $d(n)$ be the desired response. Now, due to the linear structure of the RKHS cost function $J_{\Omega}(n) = E[(d(n) - y(n))^2]$ can be minimized with respect to $\Omega$. This can be done in the same way as done in (2) using the stochastic instantaneous estimate of the gradient vector, which yields

$$
\Omega(n + 1) = \Omega(n) + 2\eta e(n)\Phi(x(n)).
$$

Like before, $\eta$ is the step-size parameter that controls the convergence, speed, and misadjustment of the adaptation algorithm [17,18]. The only catch here is that $\Omega$ in (4) is in the infinite dimensional feature space and it would be practically impossible to update for $\Omega$ directly. Instead we shall use (4) to relate each $\Omega(n)$ to its initialization $\Omega(0)$. This would easily give

$$
\Omega(n) = \Omega(0) + 2\eta \sum_{i=0}^{n-1} e(i)\Phi(x(i)).
$$
For convenience we shall choose $\Omega(0)$ to be zero (hence $e(0) = d(0)$). The final expression for $\Omega(n)$ becomes

$$
\Omega(n) = 2\eta \sum_{i=0}^{n-1} e(i)\Phi(x(i)).
$$

(6)

It is here we shall exploit the kernel trick. Given $\Omega(n)$ from (6) and the input $\Phi(x(n))$ the output at $n$ is given by

$$
y(n) = \langle \Omega(n), \Phi(x(n)) \rangle = \eta \sum_{i=0}^{n-1} e(i)\Phi(x(i)), \Phi(x(n))
$$

$$
= \eta \sum_{i=0}^{n-1} e(i)\Phi(x(i), x(n)).
$$

(7)

We call (7) the KLMS algorithm. Note that this expression can be interpreted as the output of an RBF network, with the weights given by the error at each iteration. Notice also that the network structure grows linearly with the number of data samples, which is undesirable. It is clear that, given the kernel, KLMS has a unique solution because it is solving a quadratic problem in feature space. The kernel size (the variance parameter in the RBF) is a free parameter which affects the overall performance of the algorithm like in any kernel based algorithm and can be chosen through a quick cross validation step. Silverman’s rule [19] of thumb is another alternative. For our simulations the kernel size chosen by the variance of the data works reasonable well. Here, just like for the linear LMS algorithm, the stepsize controls the variance of the data works reasonable well. Here, just like for the linear LMS algorithm, the stepsize controls the variance of the data. The initial errors in the adaptation tend to make KLMS especially useful for on-line nonlinear signal processing, but the complexity is upper bounded by the largest eigenvalue of the data covariance (in the feature space) which is difficult to estimate and dependent upon the kernel size.

Each new input sample results in an output and hence a corresponding error, which is never modified further and it is incorporated in the estimate of the next output. This recursive computation makes KLMS especially useful for on-line nonlinear signal processing, but the complexity of the algorithm increases linearly as new error samples are used. The initial errors in the adaptation tend to prevent the algorithm from over fitting the data. The initial errors are the most dominant since the errors decay quickly and seem to have a prominent role in the estimates of the output. We believe this nonparametric improvement in the output samples is a reason for not requiring any kind of explicit regularization. A more detailed and rigorous analysis of this intriguing property of KLMS is provided in [7]. The major contribution of this paper is to present a means of restricting the growth of the RBF network obtained from (7), and hence of the computation complexity. We present this in the next section.

3. KLMS-CG

3.1. Sparsity condition for the feature vectors

By checking the linear dependency of the input feature vectors, one can find the means of reducing the growing architecture of the KLMS algorithm. In this subsection, the basic method of checking the linear dependency is presented.

Let $\{\Phi(x(i)) : i \in r_{n-1}\}$ be the dictionary of linearly independent vectors with $r_{n-1}$ being the list of indices of the elements of the dictionary till time step $n - 1$. Let $M(n)$ be the length of $r_n$. A vector $\Phi(x(n))$ is linearly dependent to the elements in the dictionary if

$$
\Phi(x(n)) = \sum_{k=1}^{M_n} \beta_k \Phi(x(r_{n-1},k))
$$

with at least one nonzero $\beta_k$, where $r_{n-1,k}$ is the $k$th element of $r_{n-1}$. So we can use the following cost function to estimate $\beta_k$ and to check for linear dependency:

$$
J(\overline{\beta}) = \| \Phi(x(n)) - \sum_{k=1}^{M_n} \beta_k \Phi(x(r_{n-1},k)) \|^2
$$

$$
= \| \Phi(x(n)) - \overline{\beta} \|^2
$$

$$
= \kappa(x(n), x(n)) - 2\Phi(x(n))^T \overline{\beta} n_{n-1} + \overline{\beta}^T \overline{G}_{n-1} \overline{\beta}
$$

$$
= \kappa(x(n), x(n)) - 2k_{n-1}^T \overline{G}_{n-1} \overline{\beta} + \overline{G}_{n-1} \overline{\beta}
$$

(9)

where $k_{n-1} = [\kappa(x(n), x(r_{n-1},1)), \kappa(x(n), x(r_{n-1},2)), \ldots, \kappa(x(n), x(r_{n-1,M_n})), \ldots]$, $\overline{G}_{n-1}$ is the Gram matrix formed with the vectors in $\{\Phi(x(i)) : i \in r_{n-1}\}$, and $\overline{G}_{n-1}$ is a matrix whose $k$th column is $\Phi(x(r_{n-1,k}))$. Minimizing (9) would result in

$$
\overline{G}_{n-1} \overline{\beta} = k_{n-1}^T
$$

or

$$
[-\overline{\beta}]^T [G_{n-1} \overline{r}_{n-1}] = 0.
$$

(10)

It is easy to see that $[G_{n-1}^{-1} \ k_{n-1} \ \kappa(x(n), x(n))]$ is the Gram matrix $G_n$ with $n$ added to the list $r_{n-1}$. Thus we just need to find the vector $\overline{\beta}$ such that all but maybe the last element of $[-\overline{\beta}]^T G_{n-1} \overline{r}_{n-1}$ are zeros. Let $T_n$ be the transform on $G_n$ such that $T_n G_n$ is upper triangular. Then, the last row $t_n$ of $T_n$ satisfies $t_n [k_{n-1}] = 0$ and hence from (10), $t_n = [-\overline{\beta}]^T 1$.

Thus $t_n$ gives the least square solution to the cost function given in (9). Now, if the cost function at the least square solution is sufficiently small (say less than a small positive $\delta$) then we shall consider this new feature vector to be linearly dependent on the vectors given by $r_{n-1}$. Otherwise, the vector is considered to be almost or sufficiently linearly independent to the vectors given by $r_{n-1}$, adding its index $n$ to the list $r_{n-1}$.

Using the optimal value given by (10), the cost function would be

$$
J_{opt} = \kappa(x(n), x(n)) - k_{n-1}^T \overline{\beta}.
$$

(11)

It is easy to see that $J_{opt}$ is nothing but the last element of $t_n G_n$ or accordingly the last diagonal element of the upper triangular matrix $T_n G_n$. So the method of testing the linear dependency simply requires two steps—first, use a linear transform to make the Gram matrix upper triangular and second, compare the element in the last row and last column of the resulting matrix to $\delta$ as mentioned before.
Next, we explain the exact numerical technique using Gaussian elimination sequentially with each new incoming sample resulting in the improved algorithm. It should also be noted that the computational burden of these steps will be less than the conventional Gaussian elimination because it can be shown that for positive definite matrices, the usual pivoting is not required [20]. This is also the case here.

3.2. Algorithm

Let the new feature input \( \Phi(x(n)) \) in (7) be linearly dependent on the previous feature vectors, \( \Phi(x(i)) \), \( i = 0, 1, \ldots, n - 1 \). Then for some real \( \alpha_i \)’s,

\[
\Phi(x(n)) = \sum_{i=0}^{n-1} \alpha_i \Phi(x(i)) \tag{12}
\]

Substituting this in (7),

\[
y(n + 1) = \eta \sum_{i=0}^{n-1} e_{z,n}(i)\kappa(x(i), x(n + 1)), \tag{13}
\]

where \( e_{z,n}(i) = e(i) + x_i e(n) \). Thus the computation of \( y(n + 1) \) requires one less kernel evaluation than that given by (7). If more such input feature vectors are linearly dependent to the previous ones, growth of the algorithm suggested by (7) would reduce significantly. Of course, if the data samples in the input space are distinct and when a positive definite kernel function is used, none of the corresponding feature vectors will be exactly linearly dependent. But in practice they can be awfully close. That is why large Gram matrices are usually ill conditioned. In fact, we shall effectively employ one step of Gaussian elimination (with out pivoting) at each time instant on the Gram matrix given by the kernel to determine the linear dependency of the new input feature vector with the previous ones. Before we explain the approach let us define a few variables. \( G_n \) is the kernel Gram matrix corresponding to the inputs \( \{x(i), i \in r_n\} \), and

\[
v_n = [\kappa(x(n), x(r_{n1})), \kappa(x(n), x(r_{n2})), \ldots]^{T}, \tag{14}
\]

where \( r_n \) is the vector of indexes to the elements of the dictionary (of linearly independent vectors) till time step \( n \) and \( r_{nk} \) is the kth element of \( r_n \). The definition of \( r_n \) will be clear when the steps of the algorithm are presented next (the corresponding computation complexities are presented in parenthesis; though in most matrix operations, the computation will be almost half than usual with the matrices being upper triangular):

1. \( \{O(1)\} \) Initialize \( T_1 = \|_1, G_1 = \|_1; r_1 = 1; e_{z,1} = 1 \).
2. \( \{O(M_n)\} \) The output and the error are calculated as

\[
y(n) = \eta e_{z,n}v_n
\]

\[
e(n) = d(n) - y(n) \tag{15}
\]

and \( e_{z,n}(i) \) is the ith element of \( e_{z,n} \).

3. \( \{O(M_n^2)\} \) \( G_n = \begin{bmatrix} G_{n-1} & T_{n-1,v_n} \\ v_n & 1 \end{bmatrix} \).
4. \( \{O(1)\} \) \( T_n = \begin{bmatrix} T_{n-1} & 0 \\ 0 & 1 \end{bmatrix} \). It is easy to see that \( T_n G_n = G_n \).
5. \( \{O(M_n^2)\} \) Perform \( M_n \) Gaussian elimination row operations on \( G_n \) and the same operations on \( T_n \) to obtain \( G_n^* \) and \( T_n^* \). Then at each step \( G_n^* \) is upper triangular and \( T_n^G_n = G_n^* \).
6. \( \{O(1)\} \) If the last element \( g_n \) of the last row (and hence the whole row) of \( G_n^* \) is zero (or less than a small user defined quantity \( \delta \)), it is obvious that the Gram matrix \( G_n^* \) is ill-conditioned and hence the new feature vector, \( \Phi(x(n)) \) is (almost) linearly dependent on the previous feature vectors. If this is true and the last row of \( T_n^* \) is \( t_n \), then

\[
t_n G_n^* = 0 \tag{16}
\]

and hence \( t_n \) gives the weights with which \( \Phi(x(n)) \) is linearly dependent with the previous vectors \( \{\Phi(x(i)), i \in r_n\} \). In this case the feature vector \( \Phi(x(n)) \) is redundant and the matrices \( T_n^* \) and \( G_n^* \) are reverted back to \( T_{n-1}^* \) and \( G_{n-1}^* \) by simply deleting the last row and the last column. The update \( e_{z,n}(i) = e_{z,n-1}(i) + \alpha_i e(n) \) is also performed where \( e_{z,n}(i) \) is the ith element of \( e_{z,n} \) and \( t_i = -t_{n-1}^* \), where \( t_{n-1}^* \) is the ith element of \( t_n \).

7. \( \{O(1)\} \) If \( g_n \geq \delta \), then \( \Phi(x(n)) \) is certainly linearly independent from the previous vectors and the updates \( r_n = [r_{n-1}, n] \) and \( e_{z,n} = [e_{z,n-1}, e(n)] \) are performed.

4. Experimental comparison between KLMS and KLMS-CG

To demonstrate the performance of the proposed method we will present some simulation results. The mean square error will be used to compare the performances of the KLMS-CG, the linear LMS algorithm, and the RBF network with the weights adapted using the linear LMS algorithm for the one step prediction of the Mackey-Glass (MG30) time series. It must be noted that the RBF network trained with LMS requires one to know the centers before hand and hence all the data needs to be known prior to training. This in essence prevents it from being a true on-line method. There have been suggestions of using stochastic gradients [21] or even Genetic algorithm for the selection of the centers [10,11], but these algorithms are complex with unreliable and slow convergence.

For these experiments, an embedding dimension of six was used which is the optimal value according to Takens embedding theorem [22]. The simulations implement Eq. (15) for the KLMS-CG and Eq. (2) for the linear LMS. The kernel size and the step size were determined for best results after scanning the parameters. The data were normalized to unity variance beforehand. The kernel size, \( \sigma^2 \) was chosen to be 1 for these experiments. It was also observed that the performance was not very sensitive to the kernel sizes between 1 and 4. The values of the step size for the linear LMS and the KLMS-CG algorithms were chosen as 0.01 and 0.5, respectively. The step size for RBF with LMS training for the weights is 0.5. The centers for this RBF network was chosen though repetitive nearest
neighbor clustering, using the same number of centers as provided by KLMS-CG at the end of the training. This provides an extra overhead in computation, though the LMS adaptation is very simple. This requires to have a set of training data beforehand and is not truly on-line, unlike the linear LMS and KLMS-CG.

The plots presented include the learning curve and comparisons of MSE values for different values of the tolerance threshold $\delta$. Note that for $\delta = 0$ we obtain the ever growing RBF network. To plot the learning curve, after each update the learning was frozen and a new batch of 300 data samples was used to estimate the mean square error. Fig. 1 shows the learning curve for LMS and KLMS-CG algorithms (the step-size values were chosen for fastest convergence). It can be observed that there is a slight penalty in performance for larger values of $\delta$ since the RBF filter size is actually very different in these three cases (see Figs. 3 and 4). The speeds of convergence for three different values of $\delta$ are practically the same in the beginning since all the three algorithms are still growing, but after the 500th sample they stabilize (i.e. the learning curves are almost parallel to each other). This shows that the learning process still continues in the algorithm even when the new kernel centers are not added. Fig. 2 shows the performance of the algorithm for a range of values of $\delta$ and Fig. 3 shows the number of total kernel centers accumulated using 600 training data samples. As expected the number of kernel centers reduces drastically from 600 to less than 100 as $\delta$ increases. Yet the degradation in performance for this data set is comparatively much less and the overall filter performs much better than the adaptive linear combiner trained with LMS. Because the RBF network with LMS training of weights uses the optimal number of kernel centers from the very beginning it performs better than KLMS-CG (which starts with one center) in the initial phase of learning (Fig. 1).

5. Application to nonlinear system identification

5.1. Motivation

As we have already mentioned in the previous sections, in addition to Vapnik’s widely known SVM [1] there have been numerous kernel methods proposed in the recent years like, K-PCA [3], kernel Fisher discriminant analysis [23], and kernel canonical correlation analysis [14,24], with applications in classification and nonlinear regression. Though they have shown enormous potential in solving complex problems, their usefulness has been rather limited in many real systems requiring on-line operation. These algorithms cannot be employed online because of the difficulties posed by the basic kernel method, such as the time and memory complexity and the need to avoid overfitting.

One of the early on-line nonlinear regression algorithms with a well tailored complexity is the resource allocation network (RAN) [25]. The algorithm was presented as growing RBF network where for each new data sample that satisfies some novelty condition, one RBF unit is added to the network. At each step the weights to the RBFs is adapted using the LMS algorithm [18]. Though the RAN is simple and effective it requires the user to heuristically choose two threshold parameters for the novelty test, which may create inconsistent results. More recently a kernel RLS (KRLS) algorithm has been proposed that dealt with the complexity issues by applying a sparsification procedure to limit the size of the kernel Gram matrix [5]. This algorithm allows for on-line training but cannot handle time-varying data. An improvement has also been presented to this approach through the sliding window KRLS algorithm [12], where an efficient update rule is presented to compute the least square solution for the kernel weights with data in window that slides one sample at time. This approach has one simple

![Fig. 1. Learning curve of linear LMS, LMS trained RBF and KLMS with constrained growth for three values of $\delta$.](image-url)
flaw, the response time of the system to any abrupt change in the data is roughly the same as the window length that is chosen a priori. In addition, it requires the choice of a regularization parameter, which is not trivial. These algorithms have been used for nonlinear system identification. In certain cases, like for estimating a communication channel, it is preferred that the implementation is online and that it responds to nonstationary variations. To this cause we shall employ the KLMS algorithm that we presented in a previous chapter. As we shall see through simulations this approach has certain advantages over the other methods.

5.2. Identification of a Wiener system

The Wiener system is a well-known simple non-linear system which consists of a linear system followed by a static (memoryless) nonlinearity (see Fig. 5). Such a channel can be encountered in digital satellite communications and in digital magnetic recording. It is logical that one would prefer to use an on-line technique that can also adjust slight variations in the system. Traditionally, this type of nonlinear equalization or channel estimation has been tackled by nonlinear structures such as neural networks and MLPs [26]. Here we shall compare our...
results with the sliding window KRLS, since it has already been shown just recently that KRLS performs better than the MLPs [12] for this case. In fact, this sliding window KRLS is an improvement on the original KRLS proposed in [5]. The addition of the sliding window approach makes KRLS useful when the system changes abruptly or gradually during adaptation, which suitable for our comparison.

In addition we shall also show the results with the RAN as a reference. We should note that the RAN is the least complex with computation of $O(N)$ for each time step, where $N$ is the number of RBF (kernel) units present at each step. Our method, KLMS-CG, and KRLS have a computation complexity of $O(N^2)$. For KLMS-CG, $N$ is the number of kernel units present at each step and for KRLS, it is the size of the sliding window. As we shall see shortly, in certain cases, the number of kernel units required for KLMS-CG can be much smaller than the size of the sliding window for KRLS for the same performance, hence KRLS would be far more costly.

5.3. Experimental results

Supervised learning is used for the identification of an unknown Wiener system. To test the tracking capability of the various algorithms, we shall switch the coefficients of the linear system at a given instant. As mentioned in [12] we would expect the response time of KRLS algorithm for this change to be roughly the window size that is chosen. We would also expect the KLMS and RAN algorithms to be much faster since they do not employ any windowing scheme.

During the first part of the simulation, the linear channel is $H_1(z) = 1 + 0.0668z^{-1} - 0.4764z^{-2} + 0.8070z^{-3}$ and after receiving 600 symbols it is changed to $H_2(z) = 1 + 0.4326z^{-1} - 0.0656z^{-2} + 0.7153z^{-3}$. A binary signal ($x_n \in [-1, +1]$) is sent through this channel after which the signal is transformed nonlinearly according to the nonlinear function, $v(n) = \tanh(u(n))$, where $v(n)$ is the linear filter output. A scatter plot of $u(n)$ versus $v(n)$ can be found in Fig. 6. Finally, 15 dB of additive white Gaussian noise is added. The Wiener system is treated as a black box of which only input and output are known. The sliding window RLS algorithm was applied starting from the time index equal to the window length since that much of data is required.

System identification was performed by using three techniques: (1) KLMS-CG, (2) sliding window KRLS, and (3) RAN. For all the methods we applied time embedding techniques in which the length $L$ of the linear channel was known. The input vectors for the algorithms were time delayed vectors of length $L$, $X_n = [x_{n-L}, \ldots, x_n]^T$. Since the length of the linear channels $H_1$ and $H_2$ was four, we also used $L = 4$. Here are the parameter values that were chosen, mostly heuristically, for each method so as to obtain the best performance: (1) KLMS-CG: threshold for the linear dependency test, $\delta = 0.2$; step size = 0.2; kernel size, $\sigma^2 = 1$. (2) KRLS: window size $N = 120$; regularization parameter $= 0.001$; kernel size, $\sigma^2 = 1$. (3) RAN: the novelty test parameters—error parameter $= 0.17$ and distance parameter $= 0.17$; step size $= 0.2$; kernel size, $\sigma^2 = 1$. 

![Fig. 5. A nonlinear Wiener system.](image-url)
The MSE values at different time instances (learning curve) are given in Fig. 7. As expected the KRLS approach is the slowest and takes a time span approximately equal to the length of the sliding window to adjust for a sudden change. The RAN and the proposed KLMS-CG algorithm both seem to work similarly with same response time to an abrupt change in the channel. Though the RAN is computationally more efficient it requires one more parameter to be decided a priori. Also, both RAN and KLMS-CG restrict their architecture to only 16 RBFs in the network, automatically through their respective criteria. But using 16 as the sliding window length (and hence 16 RBFs) for KRLS is so poor that we have not included the results. KRLS required a window size of around 100 for comparable results. This is because it is only using 16 samples at any given time for estimation, whereas RAN and KLMS-CG (both LMS based algorithms), though use only 16 RBFs, incorporate the information of the past data until the given instance to estimate the output. RAN and KLMS-CG are continuously adapting at each step, so these two approaches are better than KRLS. So overall for this example, in terms of design simplicity and performance,
KLMS-CG is the most appropriate choice since it has lesser parameters than RAN, but in terms of the computation speed, RAN could be used.

6. Summary

We have derived directly Widrow’s least mean squares algorithm in an infinite dimensional kernel Hilbert space and modified the algorithm to contain growth of the resulting nonlinear filter. The algorithm uses the kernel trick to obtain the final output, effectively resulting in the adaptation of a nonlinear filter without the complexities of backpropagation and utilizing the data pretty much as the conventional linear filter trained with LMS. Although stochastic learning is known in the machine learning community, the problem of growing complexity had still not been addressed for this class of problems. Here we have proposed a set of sequential steps to test the linear dependency (in the feature space) of each incoming data with the previous input samples. This, in turn, provides a means of distributing the present error to weight the kernel functions centered at the previous input samples and restrict the growth of the algorithm complexity. This method can also be potentially used for other on-line kernel based algorithms where a batch of data is not available a priori and the complexity of the algorithm needs to be constrained sequentially one sample at a time.

It is also noteworthy that the KLMS and the improved KLMS-CG algorithms basically solve the least squares problem (in the feature space) implying that the gradient search is on a smooth quadratic performance surface, resulting in reliable convergence without the hassles of local minima. Another interesting fact when the Gaussian (or any translation invariant) kernel is utilized, is that the transformed data lies on the surface of a sphere i.e., the transformed data is automatically normalized, behaving like the normalized LMS algorithm, which is an important advantage.

References