Hindawi Publishing Corporation
EURASIP Journal on Advances in Signal Processing
Volume 2008, Article ID 784292, 12 pages
doi:10.1155/2008/784292

Research Article

Kernel Affine Projection Algorithms

Weifeng Liu and José C. Príncipe

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

Correspondence should be addressed to Weifeng Liu, weifeng@cnel.ufl.edu

Received 27 September 2007; Revised 23 January 2008; Accepted 21 February 2008

Recommended by Aníbal Figueiras-Vidal

The combination of the famed kernel trick and affine projection algorithms (APAs) yields powerful nonlinear extensions, named collectively here, KAPA. This paper is a follow-up study of the recently introduced kernel least-mean-square algorithm (KLMS). KAPA inherits the simplicity and online nature of KLMS while reducing its gradient noise, boosting performance. More interestingly, it provides a unifying model for several neural network techniques, including kernel least-mean-square algorithms, kernel adaline, sliding-window kernel recursive-least squares (KRLS), and regularization networks. Therefore, many insights can be gained into the basic relations among them and the tradeoff between computation complexity and performance. Several simulations illustrate its wide applicability.

1. INTRODUCTION

The solid mathematical foundation, wide and successful applications are making kernel methods very popular. By the famed kernel trick, many linear methods have been recast in high dimensional reproducing kernel Hilbert spaces (RKHS) to yield more powerful nonlinear extensions, including support vector machines [1], principal component analysis [2], recursive least squares [3], Hebbian algorithm [4], Adaline [5], and so forth.

More recently, a kernelized least-mean-square (KLMS) algorithm was proposed in [6], which implicitly creates a growing radial basis function network (RBF) with a learning strategy similar to resource-allocating networks (RAN) proposed by Platt [7]. As an improvement, kernelized affine projection algorithms (KAPAs) are presented for the first time in this paper by reformulating the conventional affine projection algorithm (APA) [8] in general reproducing kernel Hilbert spaces (RKHS). The new algorithms are online, simple, and significantly reduce the gradient noise compared with the KLMS and thus improve performance.

More interestingly, the KAPA reduces to the kernel least-mean square (KLMS), sliding-window kernel recursive least squares (SW-KRLS), kernel adaline, and regularization networks naturally in special cases. Thus it provides a unifying model for these existing methods and helps better understand the basic relations among them and the tradeoff between complexity and performance. Moreover, it also advances our understanding on the resource-allocating networks. Exploiting the underlying linear structure of RKHS, a brief discussion on its well-posedness will be conducted.

The organization of the paper is as follows. In Section 2, the affine projection algorithms are briefly reviewed. Next, in Section 3, the kernel trick is applied to formulate the nonlinear affine projection algorithms. Other related algorithms are reviewed as special cases of the KAPA in Section 4. We detail the implementation of the KAPA in Section 5. Three experiments are studied in Section 6 to support our theory. Finally, Section 7 summarizes the conclusions and future lines of research.

The notation used throughout the paper is summarized in Table 1.

2. A REVIEW OF THE AFFINE PROJECTION ALGORITHMS

Let d be a zero-mean scalar-valued random variable, and let $\mathbf{u}$ be a zero-mean $L \times 1$ random variable with a positive-definite covariance matrix $\mathbf{R}_{\mathbf{u}} = E[\mathbf{u}\mathbf{u}^T]$. The cross-covariance vector of d and $\mathbf{u}$ is denoted by $\mathbf{r}_{d\mathbf{u}} = E[d\mathbf{u}]$. The weight vector $\mathbf{w}$ that solves

$$\min_{\mathbf{w}} E\left|d - \mathbf{w}^T\mathbf{u}\right|^2 \tag{1}$$

is given by $\mathbf{w}^o = \mathbf{R}_{\mathbf{u}}^{-1}\mathbf{r}_{d\mathbf{u}}$ [8].

Several methods that approximate $\mathbf{w}$ iteratively also exist, for example, the common gradient method

$$\begin{aligned}\mathbf{w}(0) &= \text{initial guess;}\\ \mathbf{w}(i) &= \mathbf{w}(i-1) + \eta\big[\mathbf{r}_{d\mathbf{u}} - \mathbf{R}_{\mathbf{u}}\mathbf{w}(i-1)\big],\end{aligned} \tag{2}$$

or the regularized Newton's recursion

$$\begin{aligned}\mathbf{w}(0) &= \text{initial guess;}\\ \mathbf{w}(i) &= \mathbf{w}(i-1) + \eta\big(\mathbf{R}_{\mathbf{u}} + \varepsilon\mathbf{I}\big)^{-1}\big[\mathbf{r}_{d\mathbf{u}} - \mathbf{R}_{\mathbf{u}}\mathbf{w}(i-1)\big],\end{aligned} \tag{3}$$

where ε is a small positive regularization factor and η is the step size specified by the designer.

Stochastic-gradient algorithms replace the covariance matrix and the cross-covariance vector by local approximations directly from data at each iteration. There are several ways for obtaining such approximations. The tradeoff is computation complexity, convergence performance, and steady-state behavior [8].

Assume that we have access to observations of the random variables d and $\mathbf{u}$ over time

$$\{d(1), d(2), \dots\}, \qquad \{\mathbf{u}(1), \mathbf{u}(2), \dots\}. \tag{4}$$

The Least-mean-square (LMS) algorithm simply uses the instantaneous values for approximations $\hat{\mathbf{R}}_{\mathbf{u}} = \mathbf{u}(i)\mathbf{u}(i)^T$ and $\hat{\mathbf{r}}_{d\mathbf{u}} = d(i)\mathbf{u}(i)$. The corresponding steepest-descent recursion (2) and Newton's recursion (3) become

$$\begin{aligned}&\mathbf{w}(i) = \mathbf{w}(i-1) + \eta\mathbf{u}(i)\big[d(i) - \mathbf{u}(i)^T\mathbf{w}(i-1)\big];\\ &\mathbf{w}(i) = \mathbf{w}(i-1) + \eta\mathbf{u}(i)\big[\mathbf{u}(i)^T\mathbf{u}(i) + \varepsilon\mathbf{I}\big]^{-1}\big[d(i) - \mathbf{u}(i)^T\mathbf{w}(i-1)\big].\end{aligned} \tag{5}$$

The affine projection algorithm however employs better approximations. Specifically, $\mathbf{R}_{\mathbf{u}}$ and $\mathbf{r}_{d\mathbf{u}}$ are replaced by the instantaneous approximations from the K most recent regressors and observations. Denoting

$$\begin{aligned}\mathbf{U}(i) &= \big[\mathbf{u}(i-K+1), \dots, \mathbf{u}(i)\big]_{L\times K},\\ \mathbf{d}(i) &= \big[d(i-K+1), \dots, d(i)\big]^T,\end{aligned} \tag{6}$$

one has

$$\begin{aligned}\hat{\mathbf{R}}_{\mathbf{u}} &= \frac{1}{K}\mathbf{U}(i)\mathbf{U}(i)^T,\\ \hat{\mathbf{r}}_{d\mathbf{u}} &= \frac{1}{K}\mathbf{U}(i)\mathbf{d}(i).\end{aligned} \tag{7}$$

Therefore, (2) and (3) become

$$\mathbf{w}(i) = \mathbf{w}(i-1) + \eta\mathbf{U}(i)\big[\mathbf{d}(i) - \mathbf{U}(i)^T\mathbf{w}(i-1)\big], \tag{8}$$

$$\mathbf{w}(i) = \mathbf{w}(i-1) + \eta\big[\mathbf{U}(i)\mathbf{U}(i)^T + \varepsilon\mathbf{I}\big]^{-1}\mathbf{U}(i)\big[\mathbf{d}(i) - \mathbf{U}(i)^T\mathbf{w}(i-1)\big], \tag{9}$$

and (9), by the matrix inversion lemma, is equivalent to [8]

$$\begin{aligned}\mathbf{w}(i) = \mathbf{w}(i-1) &+ \eta\mathbf{U}(i)\big[\mathbf{U}(i)^T\mathbf{U}(i) + \varepsilon\mathbf{I}\big]^{-1}\\ &\times \big[\mathbf{d}(i) - \mathbf{U}(i)^T\mathbf{w}(i-1)\big].\end{aligned} \tag{10}$$

Table 1: Notations.

	Description	Examples
Scalars	Small *italic* letters	d
Vectors	Small **bold** letters	$\mathbf{w}$, $\boldsymbol{\omega}$, $\mathbf{a}$
Matrices	Capital **BOLD** letters	$\mathbf{U}$, $\boldsymbol{\Phi}$
Time or iteration	Indices in parentheses	$\mathbf{u}(i)$, $d(i)$
Components of vectors or matrices	Subscript indices	$\mathbf{a}_j(i)$, $\mathbf{G}_{i,j}$

It is noted that this equivalence lets us deal with the matrix $\big[\mathbf{U}(i)^T\mathbf{U}(i) + \varepsilon\mathbf{I}\big]$ instead of $\big[\mathbf{U}(i)\mathbf{U}(i)^T + \varepsilon\mathbf{I}\big]$ and it plays a very important role in the derivation of kernel extensions. We call recursion (8) APA-1 and recursion (10) APA-2.

In some circumstances, a regularized solution is needed instead of (1). The regularized LS problem is

$$\min_{\mathbf{w}} E\big|d - \mathbf{w}^T\mathbf{u}\big|^2 + \lambda\|\mathbf{w}\|^2, \tag{11}$$

where λ is the regularization parameter (not the regularization factor ε in Newton's recursion). The gradient method is

$$\begin{aligned}\mathbf{w}(i) &= \mathbf{w}(i-1) + \eta\big[\mathbf{r}_{d\mathbf{u}} - \big(\lambda\mathbf{I} + \mathbf{R}_{\mathbf{u}}\big)\mathbf{w}(i-1)\big]\\ &= (1-\eta\lambda)\mathbf{w}(i-1) + \eta\big[\mathbf{r}_{d\mathbf{u}} - \mathbf{R}_{\mathbf{u}}\mathbf{w}(i-1)\big].\end{aligned} \tag{12}$$

The Newton's recursion with $\varepsilon = 0$ is

$$\begin{aligned}\mathbf{w}(i) &= \mathbf{w}(i-1) + \eta\big(\lambda\mathbf{I} + \mathbf{R}_{\mathbf{u}}\big)^{-1}\big[\mathbf{r}_{d\mathbf{u}} - (\lambda\mathbf{I} + \mathbf{R}_{\mathbf{u}})\mathbf{w}(i-1)\big]\\ &= (1-\eta)\mathbf{w}(i-1) + \eta\big(\lambda\mathbf{I} + \mathbf{R}_{\mathbf{u}}\big)^{-1}\mathbf{r}_{d\mathbf{u}}.\end{aligned} \tag{13}$$

If the approximations (7) are used, we have

$$\mathbf{w}(i) = (1-\eta\lambda)\mathbf{w}(i-1) + \eta\mathbf{U}(i)\big[\mathbf{d}(i) - \mathbf{U}(i)^T\mathbf{w}(i-1)\big], \tag{14}$$

$$\mathbf{w}(i) = (1-\eta)\mathbf{w}(i-1) + \eta\big[\lambda\mathbf{I} + \mathbf{U}(i)\mathbf{U}(i)^T\big]^{-1}\mathbf{U}(i)\mathbf{d}(i). \tag{15}$$

which is, by the matrix inversion lemma, equivalent to

$$\mathbf{w}(i) = (1-\eta)\mathbf{w}(i-1) + \eta\mathbf{U}(i)\big[\lambda\mathbf{I} + \mathbf{U}(i)^T\mathbf{U}(i)\big]^{-1}\mathbf{d}(i). \tag{16}$$

For simplicity, recursions (14) and (16) are named here APA-3 and APA-4, respectively.

3. THE KERNEL AFFINE PROJECTION ALGORITHMS

A kernel [9] is a continuous, symmetric, positive-definite function $\kappa : \mathbb{U}\times\mathbb{U} \to \mathbb{R}$. $\mathbb{U}$ is the input domain, a compact subset of $\mathbb{R}^L$. The commonly used kernels include the Gaussian kernel (17) and the polynomial kernel (18):

$$\kappa(\mathbf{u}, \mathbf{u}') = \exp\big(-a\|\mathbf{u} - \mathbf{u}'\|^2\big), \tag{17}$$

$$\kappa(\mathbf{u}, \mathbf{u}') = \big(\mathbf{u}^T\mathbf{u}' + 1\big)^p. \tag{18}$$

The Mercer theorem [9, 10] states that any kernel $\kappa(\mathbf{u}, \mathbf{u}')$ can be expanded as follows:

$$\kappa(\mathbf{u}, \mathbf{u}') = \sum_{i=1}^{\infty} \varsigma_i \phi_i(\mathbf{u}) \phi_i(\mathbf{u}'), \tag{19}$$

where ς_i and ϕ_i are the eigenvalues and the eigenfunctions, respectively. The eigenvalues are nonnegative.

Therefore, a mapping φ can be constructed as

$$\begin{aligned} \varphi &: \mathbb{U} \longrightarrow \mathbb{F}, \\ \varphi(\mathbf{u}) &= \left[\sqrt{\varsigma_1}\phi_1(\mathbf{u}), \sqrt{\varsigma_2}\phi_2(\mathbf{u}), \dots\right], \end{aligned} \tag{20}$$

such that

$$\kappa(\mathbf{u}, \mathbf{u}') = \varphi(\mathbf{u})^T \varphi(\mathbf{u}'). \tag{21}$$

By construction, the dimensionality of $\mathbb{F}$ is determined by the number of strictly positive eigenvalues, which can be infinite in the Gaussian kernel case.

We utilize this theorem to transform the data $\mathbf{u}(i)$ into the feature space $\mathbb{F}$ as $\varphi(\mathbf{u}(i))$ and interpret (21) as the usual dot product. Denoting $\varphi(i) = \varphi(\mathbf{u}(i))$, we formulate the affine projection algorithms on the example sequence $\{d(1), d(2), \dots\}$ and $\{\varphi(1), \varphi(2), \dots\}$ to estimate the weight vector $\boldsymbol{\omega}$ that solves

$$\min_{\boldsymbol{\omega}} E\left| d - \boldsymbol{\omega}^T \varphi(\mathbf{u}) \right|^2. \tag{22}$$

By straightforward manipulation, (8) becomes

$$\boldsymbol{\omega}(i) = \boldsymbol{\omega}(i-1) + \eta \boldsymbol{\Phi}(i)\left[\mathbf{d}(i) - \boldsymbol{\Phi}(i)^T \boldsymbol{\omega}(i-1)\right], \tag{23}$$

and (10) becomes

$$\begin{aligned} \boldsymbol{\omega}(i) = \boldsymbol{\omega}(i-1) &+ \eta \boldsymbol{\Phi}(i)\left[\boldsymbol{\Phi}(i)^T \boldsymbol{\Phi}(i) + \varepsilon \mathbf{I}\right]^{-1} \\ &\times \left[\mathbf{d}(i) - \boldsymbol{\Phi}(i)^T \boldsymbol{\omega}(i-1)\right], \end{aligned} \tag{24}$$

where $\boldsymbol{\Phi}(i) = \left[\varphi(i-K+1), \dots, \varphi(i)\right]$.

Accordingly, (14) becomes

$$\boldsymbol{\omega}(i) = (1 - \lambda\eta)\boldsymbol{\omega}(i-1) + \eta \boldsymbol{\Phi}(i)\left[\mathbf{d}(i) - \boldsymbol{\Phi}(i)^T \boldsymbol{\omega}(i-1)\right], \tag{25}$$

and (16) becomes

$$\boldsymbol{\omega}(i) = (1 - \eta)\boldsymbol{\omega}(i-1) + \eta \boldsymbol{\Phi}(i)\left[\boldsymbol{\Phi}(i)^T \boldsymbol{\Phi}(i) + \lambda \mathbf{I}\right]^{-1} \mathbf{d}(i). \tag{26}$$

For simplicity, we refer to the recursions (23), (24), (25), and (26) as KAPA-1, KAPA-2, KAPA-3, and KAPA-4, respectively.

3.1. Kernel affine projection algorithm (KAPA-1)

It may be difficult to have direct access to the weights and the transformed data in feature space, so (23) needs to be modified. If we set the initial guess $\boldsymbol{\omega}(0) = 0$, the iteration of (23) will be

$$\begin{aligned} \boldsymbol{\omega}(0) &= 0, \\ \boldsymbol{\omega}(1) &= \eta d(1)\varphi(1) = \mathbf{a}_1(1)\varphi(1), \\ &\vdots \\ \boldsymbol{\omega}(i-1) &= \sum_{j=1}^{i-1} \mathbf{a}_j(i-1)\varphi(j), \\ \boldsymbol{\Phi}(i)^T \boldsymbol{\omega}(i-1) &= \left[\sum_{j=1}^{i-1} \mathbf{a}_j(i-1)\kappa_{i-K+1,j}, \dots, \right. \\ &\qquad \left. \sum_{j=1}^{i-1} \mathbf{a}_j(i-1)\kappa_{i-1,j}, \sum_{j=1}^{i-1} \mathbf{a}_j(i-1)\kappa_{i,j} \right]^T, \\ \mathbf{e}(i) &= \mathbf{d}(i) - \boldsymbol{\Phi}(i)^T \boldsymbol{\omega}(i-1), \\ \boldsymbol{\omega}(i) &= \boldsymbol{\omega}(i-1) + \eta \boldsymbol{\Phi}(i)\mathbf{e}(i) \\ &= \sum_{j=1}^{i-1} \mathbf{a}_j(i-1)\varphi(j) + \sum_{j=1}^{K} \eta \mathbf{e}_j(i)\varphi(i-j+K), \end{aligned} \tag{27}$$

where $\kappa_{i,j} = \kappa(\mathbf{u}(i), \mathbf{u}(j))$ for simplicity.

Note that during the iteration, the weight vector in the feature space assumes the following expansion:

$$\boldsymbol{\omega}(i) = \sum_{j=1}^{i} \mathbf{a}_j(i)\varphi(j) \quad \forall i > 0, \tag{28}$$

that is, the weight at time i is a linear combination of the previous transformed input. This result may seem simply a restatement of the representer theorem in [11]. However, it should be emphasized that this result does not rely on any explicit minimal norm constraint as required for the representer theorem. As pointed out in [12], the gradient search in (28) has an inherent regularization mechanism which guarantees the solution is in the data subspace under appropriate initialization. In general, the initialization $\boldsymbol{\omega}(0)$ can introduce whatever apriori information is available, which can be any linear combination of any transformed data in order to utilize the kernel trick.

By (28), the updating on the weight vector reduces to the updating on the expansion coefficients

$$\mathbf{a}_k(i) = \begin{cases} \eta\left(d(i) - \displaystyle\sum_{j=1}^{i-1} \mathbf{a}_j(i-1)\kappa_{i,j}\right), & k = i, \\ \mathbf{a}_k(i-1) + \eta\left(d(k) - \displaystyle\sum_{j=1}^{i-1} \mathbf{a}_j(i-1)\kappa_{k,j}\right), & \\ \qquad\qquad i - K + 1 \le k \le i - 1, & \\ \mathbf{a}_k(i-1), \quad 1 \le k < i - K + 1. & \end{cases} \tag{29}$$

Initialization:
learning step η
$\mathbf{a}_1(1) = \eta d(1)$
while $\{\mathbf{u}(i), d(i)\}$ available **do**
  %allocate a new unit
  $\mathbf{a}_i(i-1) = 0$
  for $k = \max(1, i-K+1)$ to i **do**
    %evaluate outputs of the current network
    $y(i,k) = \sum_{j=1}^{i-1} \mathbf{a}_j(i-1)\kappa_{k,j}$
    %compute errors
    $e(i,k) = d(k) - y(i,k)$
    %update the $\min(i,K)$ most recent units
    $\mathbf{a}_k(i) = \mathbf{a}_k(i-1) + \eta e(i,k)$
  end for
  if $i > K$ **then**
    %keep the remaining
    for $k = 1$ to $i-K$ **do**
    $\mathbf{a}_k(i) = \mathbf{a}_k(i-1)$
    end for
  end if
end while

Algorithm 1: Kernel affine projection algorithm (KAPA-1).

Since $\mathbf{e}_{i+1-k}(i) = d(k) - \sum_{j=1}^{i-1} \mathbf{a}_j(i-1)\kappa_{k,j}$ is the prediction error of data $\{\mathbf{u}(k), d(k)\}$ by the network $\boldsymbol{\omega}(i-1)$, the interpretation of (29) is straightforward: allocate a new unit with coefficient $\eta\mathbf{e}_1(i)$ and update the coefficients for the other $K-1$ most recent units by $\eta\mathbf{e}_{i+1-k}(i)$ for $i-K+1 \le k \le i-1$.

The pseudocode for KAPA-1 is listed in Algorithm 1.

3.2. Normalized KAPA (KAPA-2)

Similarly, the regularized Newton's recursion (24) can be factorized into the following steps:

$$\begin{aligned}
\boldsymbol{\omega}(i-1) &= \sum_{j=1}^{i-1} \mathbf{a}_j(i-1)\varphi(j),\\
\mathbf{e}(i) &= \mathbf{d}(i) - \boldsymbol{\Phi}(i)^T\boldsymbol{\omega}(i-1),\\
\mathbf{G}(i) &= \boldsymbol{\Phi}(i)^T\boldsymbol{\Phi}(i),\\
\boldsymbol{\omega}(i) &= \boldsymbol{\omega}(i-1) + \eta\boldsymbol{\Phi}(i)\left[\mathbf{G}(i) + \varepsilon\mathbf{I}\right]^{-1}\mathbf{e}(i).
\end{aligned} \tag{30}$$

In practice, we do not have access to the transformed weight $\boldsymbol{\omega}$ or any transformed data, so the update has to be on the expansion coefficient $\mathbf{a}$ like in KAPA-1. The whole recursion is similar to the KAPA-1 except that the error is normalized by a $K \times K$ matrix $\left[\mathbf{G}(i) + \varepsilon\mathbf{I}\right]^{-1}$.

3.3. Leaky KAPA (KAPA-3)

The feature space may be infinite dimensional depending on the chosen kernel, which may cause the cost function (22) to be ill posed in the conventional empirical risk minimization (ERM) sense [13]. The common practice is to constrain the solution norm:

$$\min_{\boldsymbol{\omega}} E\left|d - \boldsymbol{\omega}^T\varphi(\mathbf{u})\right|^2 + \lambda\|\boldsymbol{\omega}\|^2. \tag{31}$$

As we have already shown in (25), the leaky KAPA is

$$\boldsymbol{\omega}(i) = (1-\lambda\eta)\boldsymbol{\omega}(i-1) + \eta\boldsymbol{\Phi}(i)\left[\mathbf{d}(i) - \boldsymbol{\Phi}(i)^T\boldsymbol{\omega}(i-1)\right]. \tag{32}$$

Again, the iteration will be on the expansion coefficient $\mathbf{a}$, which is similar to the KAPA-1:

$$\mathbf{a}_k(i) = \begin{cases} \eta\left(d(i) - \sum_{j=1}^{i-1}\mathbf{a}_j(i-1)\kappa_{i,j}\right), & k = i,\\ (1-\lambda\eta)\mathbf{a}_k(i-1) + \eta\left(d(k) - \sum_{j=1}^{i-1}\mathbf{a}_j(i-1)\kappa_{k,j}\right), & \\ \qquad\qquad i-K+1 \le k \le i-1, & \\ (1-\lambda\eta)\mathbf{a}_k(i-1), \quad 1 \le k < i-K+1. & \end{cases} \tag{33}$$

The only difference is that KAPA-3 has a scaling factor $(1-\lambda\eta)$ multiplying the previous weight, which is usually less than 1, and it imposes a forgetting mechanism so that the training data in the far past are scaled down exponentially. Furthermore, since the network size is growing over training, any transformed data can be pruned from the expansion easily if its coefficient is smaller than some prespecified threshold. For large data sets, the growing nature of this family of algorithms poses a big problem for implementations, therefore, network size control is very important. We will discuss this issue more in the sparsification section.

3.4. Leaky KAPA with Newton's recursion (KAPA-4)

As before, the KAPA-4 (26) reduces to

$$\mathbf{a}_k(i) = \begin{cases} \eta d(i), & k = i,\\ (1-\eta)\mathbf{a}_k(i-1) + \eta d(k), & \\ \quad i-K+1 \le k \le i-1, & \\ (1-\eta)\mathbf{a}_k(i-1), \quad 1 \le k < i-K+1. & \end{cases} \tag{34}$$

Among these four algorithms, the first three require the error information to update the network which is computationally expensive. Therefore, the different update rule in KAPA-4 has a huge significance in terms of computation since it only needs a $K \times K$ matrix inversion, which, by using the sliding-window trick, only requires $O(K^2)$ operations [14].

We summarize the four KAPA update equations in Table 2 for convenience.

Table 2: Comparison of four KAPA update rules.

Algorithm	Update equation
KAPA-1	$\boldsymbol{\omega}(i) = \boldsymbol{\omega}(i-1) + \eta\boldsymbol{\Phi}(i)[\mathbf{d}(i) - \boldsymbol{\Phi}(i)^T\boldsymbol{\omega}(i-1)]$
KAPA-2	$\boldsymbol{\omega}(i) = \boldsymbol{\omega}(i-1) + \eta\boldsymbol{\Phi}(i)[\boldsymbol{\Phi}(i)^T\boldsymbol{\Phi}(i) + \varepsilon\mathbf{I}]^{-1}[\mathbf{d}(i) - \boldsymbol{\Phi}(i)^T\boldsymbol{\omega}(i-1)]$
KAPA-3	$\boldsymbol{\omega}(i) = (1-\lambda\eta)\boldsymbol{\omega}(i-1)+\eta\boldsymbol{\Phi}(i)[\mathbf{d}(i)-\boldsymbol{\Phi}(i)^T\boldsymbol{\omega}(i-1)]$
KAPA-4	$\boldsymbol{\omega}(i) = (1-\eta)\boldsymbol{\omega}(i-1)+\eta\boldsymbol{\Phi}(i)[\boldsymbol{\Phi}(i)^T\boldsymbol{\Phi}(i)+\lambda\mathbf{I}]^{-1}\mathbf{d}(i)$

4. A TAXONOMY FOR RELATED ALGORITHMS

4.1. Kernel least-mean-square algorithm (KAPA-1, $K = 1$)

If $K = 1$, KAPA-1 reduces to the following kernel least-mean-square algorithm (KLMS) introduced in [6]:

$$\boldsymbol{\omega}(i) = \boldsymbol{\omega}(i-1) + \eta\varphi(i)\big[d(i) - \varphi(i)^T\boldsymbol{\omega}(i-1)\big]. \tag{35}$$

It is not difficult to verify that the weight vector assumes the following expansion:

$$\boldsymbol{\omega}(i) = \sum_{j=1}^{i} e(j)\varphi(j), \tag{36}$$

where $e(j) = d(j) - \boldsymbol{\omega}(j-1)^T\varphi(j)$ is the apriori error.

It is seen that the KLMS allocates a new unit when a new training data comes in with the input $\mathbf{u}(i)$ as the center and the prediction error as the coefficient (scaled by the step size). In other words, once the unit is allocated, the coefficient is fixed. It mimics the resource-allocating step in the RAN algorithm whereas it neglects the adaptation step. In this sense, the KAPA algorithms, that allocate a new unit for the present input and also adapt the other $K-1$ most recent allocated units, are closer to the original RAN.

The normalized version of the KLMS is as follows (NKLMS):

$$\boldsymbol{\omega}(i) = \boldsymbol{\omega}(i-1) + \frac{\eta\varphi(i)}{\varepsilon + \kappa_{i,i}}\big[d(i) - \varphi(i)^T\boldsymbol{\omega}(i-1)\big], \tag{37}$$

Notice that for translation invariant kernels, that is, $\kappa_{i,i} = \text{const}$, the KLMS is automatically normalized. Sometimes we use KLMS-1 and KLMS-2 to distinguish the two.

4.2. Norma (KAPA-3, $K = 1$)

Similarly, the KAPA-3 (25) reduces to the Norma algorithm introduced by Kivinen in [15]:

$$\boldsymbol{\omega}(i) = (1-\eta\lambda)\boldsymbol{\omega}(i-1) + \eta\varphi(i)\big[d(i) - \varphi(i)^T\boldsymbol{\omega}(i-1)\big]. \tag{38}$$

4.3. Kernel Adaline (KAPA-1, $K = N$)

Assume that the size of the training data is finite N. If we set $K = N$, then the update rule of the KAPA-1 becomes

$$\boldsymbol{\omega}(i) = \boldsymbol{\omega}(i-1) + \eta\boldsymbol{\Phi}\big[\mathbf{d} - \boldsymbol{\Phi}^T\boldsymbol{\omega}(i-1)\big], \tag{39}$$

where the full data matrices are

$$\boldsymbol{\Phi} = [\varphi(1), \ldots, \varphi(N)], \qquad \mathbf{d} = [d(1), \ldots, d(N)]. \tag{40}$$

It is easy to check that the weight vector also assumes the following expansion:

$$\boldsymbol{\omega}(i) = \sum_{j=1}^{N} \mathbf{a}_j(i)\varphi(j), \tag{41}$$

and the updating on the expansion coefficients is

$$\mathbf{a}_j(i) = \mathbf{a}_j(i-1) + \eta\big[d(j) - \varphi(j)^T\boldsymbol{\omega}(i-1)\big]. \tag{42}$$

This is nothing but the kernel adaline introduced in [5]. Notice the fact that the kernel adaline is not an online method.

4.4. Recursively adapted radial basis function networks (KAPA-3, $\eta\lambda = 1$, $K = N$)

Assume the size of the training data is N as above. If we set $\eta\lambda = 1$ and $K = N$, the update rule of KAPA-3 becomes

$$\boldsymbol{\omega}(i) = \eta\boldsymbol{\Phi}\big[\mathbf{d} - \boldsymbol{\Phi}^T\boldsymbol{\omega}(i-1)\big], \tag{43}$$

which is the recursively adapted RBF (RA-RBF) network introduced in [16]. This is a very intriguing algorithm using the "global" error directly to compose the new network. By contrast, the KLMS-1 uses the apriori errors to compose the network.

4.5. Sliding-window Kernel RLS (KAPA-4, $\eta = 1$)

In KAPA-4, if we set $\eta = 1$, we have

$$\boldsymbol{\omega}(i) = \boldsymbol{\Phi}(i)\big[\boldsymbol{\Phi}(i)^T\boldsymbol{\Phi}(i) + \lambda\mathbf{I}\big]^{-1}\mathbf{d}(i), \tag{44}$$

which is the sliding-window kernel RLS (SW-KRLS) introduced in [14]. The inverse operation of the sliding-window Gram matrix can be simplified to $O(K^2)$.

4.6. Regularization networks (KAPA-4, $\eta = 1$, $K = N$)

We assume there are only N training data and $K = N$. Equation (26) becomes directly

$$\boldsymbol{\omega}(i) = \boldsymbol{\Phi}\big[\boldsymbol{\Phi}^T\boldsymbol{\Phi} + \lambda\mathbf{I}\big]^{-1}\mathbf{d}, \tag{45}$$

which is the regularization network (RegNet) [13].

We summarize all the related algorithms in Table 3 for convenience.

5. KAPA IMPLEMENTATION

In this section, we will discuss the implementation of the KAPA algorithms in detail.

Table 3: List of related algorithms.

Algorithm	Update equation	Relation to KAPA
KLMS	$\boldsymbol{\omega}(i) = \boldsymbol{\omega}(i-1) + \eta\varphi(i)[d(i) - \varphi(i)^T\boldsymbol{\omega}(i-1)]$	KAPA-1, $K = 1$
NKLMS	$\boldsymbol{\omega}(i) = \boldsymbol{\omega}(i-1) + \frac{\eta\varphi(i)}{(\varepsilon + \kappa_{i,i})}[d(i) - \varphi(i)^T\boldsymbol{\omega}(i-1)]$	KAPA-2, $K = 1$
Norma	$\boldsymbol{\omega}(i) = (1-\eta\lambda)\boldsymbol{\omega}(i-1) + \eta\varphi(i)[d(i) - \varphi(i)^T\boldsymbol{\omega}(i-1)]$	KAPA-3, $K = 1$
Kernel Adaline	$\boldsymbol{\omega}(i) = \boldsymbol{\omega}(i-1) + \eta\boldsymbol{\Phi}[\mathbf{d} - \boldsymbol{\Phi}^T\boldsymbol{\omega}(i-1)]$	KAPA-1, $K = N$
RA-RBF	$\boldsymbol{\omega}(i) = \eta\boldsymbol{\Phi}[\mathbf{d} - \boldsymbol{\Phi}^T\boldsymbol{\omega}(i-1)]$	KAPA-3, $\eta\lambda = 1$, $K = N$
SW-KRLS	$\boldsymbol{\omega}(i) = \boldsymbol{\Phi}(i)[\boldsymbol{\Phi}(i)^T\boldsymbol{\Phi}(i) + \lambda\mathbf{I}]^{-1}\mathbf{d}(i)$	KAPA-4, $\eta = 1$
RegNet	$\boldsymbol{\omega}(i) = \boldsymbol{\Phi}[\boldsymbol{\Phi}^T\boldsymbol{\Phi} + \lambda\mathbf{I}]^{-1}\mathbf{d}$	KAPA-4, $\eta = 1$, $K = N$

5.1. Error reusing

As we see in KAPA-1, KAPA-2, and KAPA-3, the most time-consuming part of the computation is to obtain the error information. For example, suppose $\boldsymbol{\omega}(i-1) = \sum_{j=1}^{i-1}\mathbf{a}_j(i-1)\varphi(j)$. We need to calculate $e(i,k) = d(k) - \boldsymbol{\omega}(i-1)^T\varphi(k)$ $(i-K+1 \le k \le i)$ to compute $\boldsymbol{\omega}(i)$, which consists of $(i-1)K$ kernel evaluations. As i increases, this dominates the computation time. In this sense, the computation complexity of the KAPA is K times of the KLMS. However, after a careful manipulation, we can shrink the complexity gap between KAPA and the KLMS.

Assume that we store all the K errors $e(i-1,k) = d(k) - \boldsymbol{\omega}(i-2)^T\varphi(k)$ for $i-K \le k \le i-1$ from the previous iteration. At the present iteration, we have

$$\begin{aligned} e(i,k) &= d(k) - \varphi(k)^T\boldsymbol{\omega}(i-1) \\ &= d(k) - \varphi(k)^T\Big[\boldsymbol{\omega}(i-2) + \eta\sum_{j=i-K}^{i-1} e(i-1,j)\varphi(j)\Big] \\ &= \big[d(k) - \varphi(k)^T\boldsymbol{\omega}(i-2)\big] + \eta\sum_{j=i-K}^{i-1} e(i-1,j)\kappa_{j,k} \\ &= e(i-1,k) + \sum_{j=i-K}^{i-1}\eta e(i-1,j)\kappa_{j,k}. \end{aligned} \tag{46}$$

Since $e(i-1,i)$ has not been computed yet, we have to calculate $e(i,i)$ by $i-1$ times kernel evaluation anyway. Overall the computation complexity of the KAPA-1 is $O(i+K^2)$, which is $O(K^2)$ more than the KLMS.

5.2. Sliding-window gram matrix inversion

In KAPA-2 and KAPA-4, another computation difficulty is to invert a $K \times K$ matrix, which normally requires $O(K^3)$. However, in the KAPA, the data matrix $\boldsymbol{\Phi}(i)$ has a sliding window structure, therefore, a trick can be used to speed up the computation. The trick is based on the matrix inversion formula and was introduced in [14]. We outline the basic calculation steps here. Suppose the sliding matrices share the same sub-matrix $\mathbf{D}$:

$$\mathbf{G}(i-1) + \lambda\mathbf{I} = \begin{bmatrix} a & \mathbf{b}^T \\ \mathbf{b} & \mathbf{D} \end{bmatrix}, \qquad \mathbf{G}(i) + \lambda\mathbf{I} = \begin{bmatrix} \mathbf{D} & \mathbf{h} \\ \mathbf{h}^T & g \end{bmatrix}, \tag{47}$$

and we know from the previous iteration that

$$\big(\mathbf{G}(i-1) + \lambda\mathbf{I}\big)^{-1} = \begin{bmatrix} e & \mathbf{f}^T \\ \mathbf{f} & \mathbf{H} \end{bmatrix}. \tag{48}$$

First, we need to calculate the inverse of $\mathbf{D}$ as

$$\mathbf{D}^{-1} = \mathbf{H} - \mathbf{f}\mathbf{f}^T/e. \tag{49}$$

Then, we can update the inverse of the new Gram matrix as

$$\big(\mathbf{G}(i) + \lambda\mathbf{I}\big)^{-1} = \begin{bmatrix} \mathbf{D}^{-1} + (\mathbf{D}^{-1}\mathbf{h})(\mathbf{D}^{-1}\mathbf{h})^T s & -(\mathbf{D}^{-1}\mathbf{h})s \\ -(\mathbf{D}^{-1}\mathbf{h})^T s & s \end{bmatrix} \tag{50}$$

with $s = (g - \mathbf{h}^T\mathbf{D}^{-1}\mathbf{h})^{-1}$. s^{-1} is the Schur complement of $\mathbf{D}$ in $(\mathbf{G}(i) + \lambda\mathbf{I})$, which actually measures the distance of the new data to the other $K-1$ most recent data in the feature space. The overall complexity is $O(K^2)$.

5.3. Sparsification

A sparse model is desired because it reduces the complexity in terms of computation and memory, and it usually yields better generalization [3]. On the other hand, in the context of adaptive filtering, training data may just be available sequentially, that is, one at a time. As we see in the formulation of KAPA, the network size increases linearly with the number of training data, which may pose a big problem for the KAPA algorithms to be applied in online applications. The sparse model idea is inspired by Vapnik's support vector machines. It is also introduced in [7] with the novelty criterion and extensively studied in [3] under approximate linear dependency (ALD). There are many other ways to achieve sparseness that require the creation of a basis dictionary and storage of the corresponding coefficients. Suppose the present dictionary is $\mathcal{D}(i) = \{\mathbf{c}_j\}_{j=1}^{m(i)}$, where $\mathbf{c}_j$ is the jth center and $m(i)$

is the cardinality. When a new data pair $\{\mathbf{u}(i+1), d(i+1)\}$ is presented, a decision is made immediately whether $\mathbf{u}(i+1)$ should be added into the dictionary as a center.

The novelty criterion introduced by Platt is relatively simple. First, it calculates the distance of $\mathbf{u}(i+1)$ to the present dictionary $\text{dis}_1 = \min_{\mathbf{c}_j \in \mathcal{D}(i)} \|\mathbf{u}(i+1) - \mathbf{c}_j\|$. If it is smaller than some preset threshold, say δ_1, $\mathbf{u}(i+1)$ will not be added into the dictionary. Otherwise, the method computes the prediction error $e(i+1, i+1) = d(i+1) - \varphi(i+1)^T \boldsymbol{\omega}(i)$. Only if the prediction error is larger than another preset threshold, say δ_2, $\mathbf{u}(i+1)$ will be accepted as a new center.

The ALD test introduced in [3] is more computationally involved. It tests the following cost $\text{dis}_2 = \min_{\forall \mathbf{b}} \|\varphi(\mathbf{u}(i+1)) - \sum_{\mathbf{c}_j \in \mathcal{D}(i)} \mathbf{b}_j \varphi(\mathbf{c}_j)\|$ which indicates the distance of the new input to the linear span of the present dictionary in the feature space. It turns out that dis_2 is the Schur complement of the Gram matrix of the present dictionary. As we saw in the previous section, this result can be used to get the new Gram matrix inverse if $\mathbf{u}(i+1)$ is accepted into the dictionary. Therefore, this method is more suitable for the KAPA-2 and KAPA-4 because of efficiency. This link is very interesting since it reveals that the ALD test actually guarantees the invertibility of the new Gram matrix.

In the sparse model, if the new data is determined to be "novel," the $K-1$ most recent data points in the dictionary are used to form the data matrix $\boldsymbol{\Phi}(i)$ together with the new data. Therefore, a new unit is allocated and the update is on the $K-1$ most recent units in the dictionary. If the new data is determined to be not "novel," it is simply discarded in this paper, but a different strategy can be employed to utilize the information like in [3, 7].

The important consequences of the sparsification procedure are as follows.

(1) If the input domain $\mathbb{U}$ is a compact set, the cardinality of the dictionary is always finite and upper bounded. This statement is not hard to prove using the finite covering theorem of the compact set and the fact that elements in the dictionary are δ-separable [3]. Here is the brief idea: suppose spheres with diameter δ are used to cover $\mathbb{U}$ and the optimal covering number is N. Then, because any two centers in the dictionary can not be in the same sphere, the total number of the centers will be no greater than N regardless of the distribution and temporal structure of $\mathbf{u}$. Of course, this is a worst case upper bound. In the case of finite training data, the network size will be finite anyway. This is true in applications like channel equalization, where the training sequence is part of each transmission frame. In a stationary environment, the network converges quickly and the threshold on prediction errors plays its part to constrain the network size. We will validate this claim in the simulation section. In a nonstationary environment, more sophisticated pruning methods should be used to constrain the network size. Simple strategies include pruning the oldest unit in the dictionary [14], pruning randomly [17], and pruning the unit with the least coefficient or similar [18, 19]. Another alternative approach is to solve the problem in the primal space [20, 21] directly by using the low rank approximation methods such as Nyström method [22], incomplete Cholesky factorization [23], and kernel principal component analysis [2]. It should be pointed out that the scalability issue is at the core of the kernel methods and so all the kernel methods need to deal with it in one way or the other. Indeed, the sequential nature of the KAPA enables active learning [24, 25] on huge data sets which is impossible in batch mode algorithms like regularization networks. The discussion on active learning with the KAPA is out of the scope of this paper and will be part of the future work.

(2) Based on (1), we can prove that the solution norms of KLMS-1 and KAPA-1 are upper bounded [12].

The significance of (1) is of practical interest because it states that the system complexity is controlled by the novelty criterion parameters, and designers can estimate a worst case upper bound. The significance of (2) is of theoretical interest because it guarantees the well-posedness of the algorithms. The well-posedness of the KAPA-3 and KAPA-4 is mostly ensured by the regularization term, see [13, 14] for details.

Table 4: Performance comparison in MG time series prediction.

Algorithm	Parameters	Test mean square error
LMS	$\eta = 0.04$	0.0208 ± 0.0009
KLMS	$\eta = 0.02$	0.0052 ± 0.00022
SW-KRLS	$K = 50, \lambda = 0.1$	0.0052 ± 0.00026
KAPA-1	$\eta = 0.03, K = 10$	0.0048 ± 0.00023
KAPA-2	$\eta = 0.03, K = 10, \varepsilon = 0.1$	0.0040 ± 0.00028
KRLS	$\lambda = 0.1$	0.0027 ± 0.00009

6. SIMULATIONS

6.1. Time series prediction

The first example is the short-term prediction of the Mackey-Glass (MG) chaotic time series [26, 27]. It is generated from the following time delay ordinary differential equation:

$$\frac{dx(t)}{dt} = -bx(t) + \frac{ax(t-\tau)}{1 + x(t-\tau)^{10}}, \tag{51}$$

with $b = 0.1$, $a = 0.2$, and $\tau = 30$. The time series is discretized at a sampling period of 6 seconds. The time embedding is 7, that is, $\mathbf{u}(i) = [x(i-7), x(i-6), \ldots, x(i-1)]^T$ are used as the input to predict the present one $x(i)$ which is the desired response here. A segment of 500 samples is used as the training data and another 100 points as the test data (in the testing phase, the filter is fixed). All the data is corrupted by Gaussian noise with zero mean and 0.001 variance.

We compare the prediction performance of KLMS, KAPA-1, KAPA-2, KRLS, and a linear combiner trained with LMS. A Gaussian kernel with kernel parameter $a = 1$ in (17) is chosen for all the kernel-based algorithms. One hundred Monte Carlo simulations are run with different realizations of noise. The results are summarized in Table 4. Figure 1 is the learning curves for the LMS, KLMS-1, KAPA-1, KAPA-2 ($K = 10$), and KRLS, respectively. As expected, the KAPA outperforms the KLMS.

As we can see in Table 4, the performance of the KAPA-2 is substantially better than the KLMS. All the results in

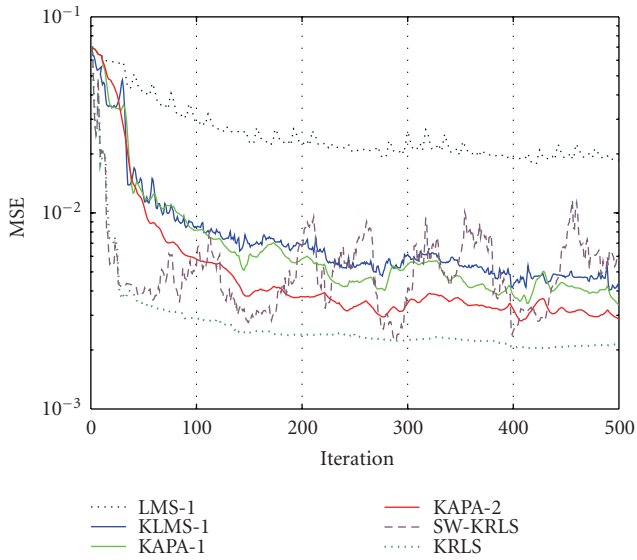

Figure 1: The learning curves of the LMS, KLMS, KAPA-1 ($K = 10$), KAPA-2 ($K = 10$), SW-KRLS ($K = 50$), and KRLS.

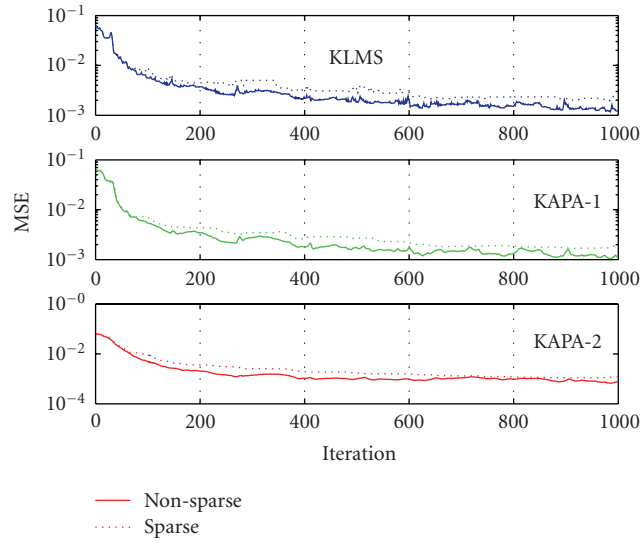

Figure 2: The learning curves of the KLMS-1, KAPA-1 ($K = 10$), and KAPA-2 ($K = 10$) with and without sparsification.

Table 5: Complexity comparison at iteration i.

Algorithm	Computation	Memory
LMS	$O(L)$	$O(L)$
KLMS	$O(i)$	$O(i)$
SW-KRLS	$O(K^2)$	$O(K^2)$
KAPA-1	$O(i+K^2)$	$O(i+K)$
KAPA-2	$O(i+K^2)$	$O(i+K^2)$
KAPA-4	$O(K^2)$	$O(i+K^2)$
KRLS	$O(i^2)$	$O(i^2)$

the tables are in the form of "average ± standard deviation." Table 5 summarizes the computational complexity of these algorithms. The KLMS and KAPA effectively reduce the computational complexity and memory storage when compared with the KRLS. KAPA-3 and sliding-window KRLS are also tested on this problem. It is observed that the performance of the KAPA-3 is similar to KAPA-1 when the forgetting term is very close to 1 as expected, and the results are severely biased when the forgetting term is reduced further. The reason can be found in [12]. The performance of the sliding-window KRLS is included in Figure 1 and Table 4 with $K = 50$. It is observed that KAPA-4 (including the sliding-window KRLS) does not perform well with small K (< 50).

Next, we test how the novelty criterion affects the performance. A segment of 1000 samples is used as the training data and another 100 as the test data. All the data is corrupted by Gaussian noise with zero mean and 0.001 variance. The thresholds in the novelty criterion are set as $\delta_1 = 0.02$ and $\delta_2 = 0.06$. The learning curves are shown in Figure 2 and the results are summarized in Table 6. It is seen that the complexity can be reduced dramatically with the novelty criterion with slight performance degeneration. Here, SKLMS and SKAPA denote the sparse KLMS and the sparse KAPA, respectively.

Several comments follow: although formally being adaptive filters, these algorithms can be viewed as efficient alternatives to batch mode RBF networks; therefore, it is practical to freeze their weights during the test phase. Moreover, when compared with other nonlinear filters such as RBF's, we divide the data in training and testing as normally done in neural networks. Of course, it is also feasible to use the apriori prediction error as a performance indicator like in conventional adaptive filtering literature.

6.2. Noise cancellation

Another important problem in signal processing is noise cancellation in which an unknown interference has to be removed based on some reference measurement. The basic structure of a noise cancellation system is shown in Figure 3. The primary signal is $s(i)$ and its noisy measurement $d(i)$ acts as the desired signal of the system. $n(i)$ is a white noise process which is unknown, and $u(i)$ is its reference measurement, that is, a distorted version of the noise process through some distortion function, which is unknown in general. Here, $u(i)$ is the input of the adaptive filter. The objective is to use $u(i)$ as the input to the filter and to obtain, as the filter output, an estimate of the noise source $n(i)$. Therefore, the noise can be subtracted from $d(i)$ to improve the signal-noise ratio.

In this example, the noise source is assumed white, uniformly distributed between $[-0.5, 0.5]$. The interference distortion function is assumed to be

$$u(i) = n(i) - 0.2u(i-1) - u(i-1)n(i-1) + 0.1n(i-1) + 0.4u(i-2). \quad (52)$$

Table 6: Performance comparison in MG time series prediction on novelty criterion.

Algorithm	Parameters	Test mean square error	Dictionary size
KLMS-1	$\eta = 0.02$	0.0015 ± 0.00012	1000
SKLMS-1	$\eta = 0.02$	0.0021 ± 0.00017	220
KAPA-1	$\eta = 0.03$	0.0012 ± 0.00014	1000
SKAPA-1	$\eta = 0.03$	0.0017 ± 0.00016	209
KAPA-2	$\eta = 0.03, \epsilon = 0.1$	0.0007 ± 0.00010	1000
SKAPA-2	$\eta = 0.03, \epsilon = 0.1$	0.0011 ± 0.00016	195

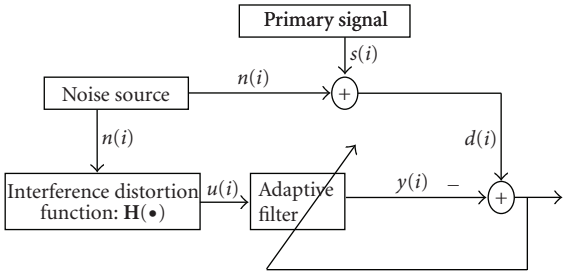

Figure 3: The basic structure of the noise cancellation system.

As we see, the distortion function has infinite impulsive response, which, on the other hand, means it is impossible to recover $n(i)$ from a finite time delay embedding of $u(i)$. We rewrite the distortion function as

$$n(i) = u(i) + 0.2u(i-1) - 0.4u(i-2) + (u(i-1) - 0.1)n(i-1). \quad (53)$$

Therefore, the present value of the noise source $n(i)$ depends not only on the reference noise measure $[u(i), u(i-1), u(i-2)]$, but also on the previous value $n(i-1)$, which in turn depends on $[u(i-1), u(i-2), u(i-3)]$, and so on. It means we need a very long time embedding (infinite long theoretically) in order to recover $n(i)$ accurately. However, the recursive nature of the adaptive system provides a feasible alternative, that is, we feedback the output of the filter $\hat{n}(i-1)$, which is the estimate of $n(i-1)$, to estimate the present one, pretending $\hat{n}(i-1)$ is the true value of $n(i-1)$. Therefore, the input of the adaptive filter can be in the form of $[u(i), u(i-1), u(i-2), \hat{n}(i-1)]$. It can be seen that the system is inherently recurrent. In the linear case with a DARMA model, it is studied under *output error methods* [28]. However, it will be nontrivial to generalize the results concerning convergence and stability to nonlinear cases, and we will address it in the future work.

We assume the primary signal $s(i) = 0$ during the training phase. And the system simply tries to reconstruct the noise source from the reference measure. We use a linear filter trained with the normalized LMS, two nonlinear filters trained with the SKLMS-1, and the SKAPA-2 ($K = 10$), respectively. 2000 training samples are used and 400 Monte Carlo simulations are run to get the ensemble learning curves as shown in Figure 4. The step size and regularization parameter for the NLMS are 0.2 and 0.005. The step sizes for SKLMS-1 and SKAPA-2 are 0.5 and 0.2, respectively.

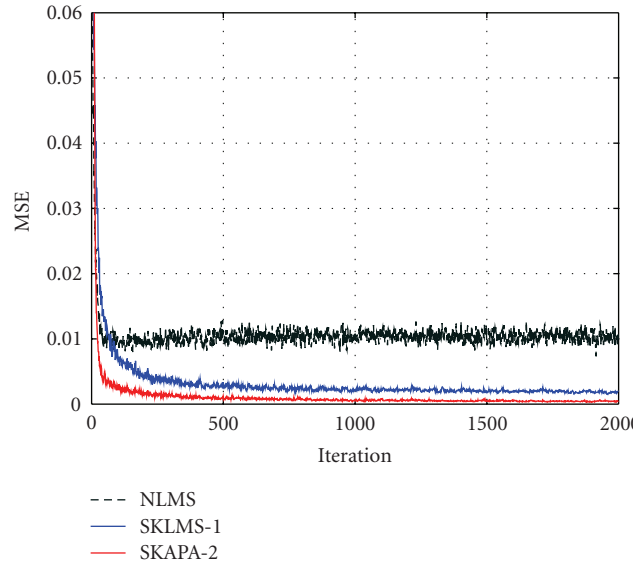

Figure 4: Ensemble learning curves of NLMS, SKLMS-1, and SKAPA-2 ($K = 10$) in noise cancellation.

Table 7: Noise reduction comparison in noise cancellation.

Algorithm	Network size	NR(dB)
NLMS	N/A	9.40
SKLMS-1	581	16.97
SKAPA-2	507	22.99

The Gaussian kernel is used for both KLMS and KAPA with kernel parameter $a = 1$. The tolerance parameters for KLMS and KAPA are $\delta_1 = 0.15$ and $\delta_2 = 0.01$, and the noise reduction factor (NR), which is defined as $10\log_{10}\{E[n^2(i)]/E[n(i) - y(i)]^2\}$, is listed in Table 7. The performance improvement of SKAPA-2 is obvious when compared with SKLMS-1.

6.3. Nonlinear channel equalization

In this example, we consider a nonlinear channel equalization problem, where the nonlinear channel is modeled by a nonlinear Wiener model. The nonlinear Wiener model consists of a serial connection of a linear filter and a memoryless nonlinearity (See Figure 5). This kind of model has been used to model digital satellite communication channels [29] and digital magnetic recording channels [30].

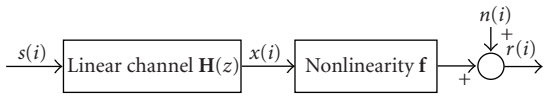

Figure 5: Basic structure of the nonlinear channel.

The problem setting is as follows: a binary signal $\{s(1), s(2), \ldots, s(N)\}$ is fed into the nonlinear channel. At the receiver end of the channel, the signal is further corrupted by additive i.i.d. Gaussian noise and is then observed as $\{r(1), r(2), \ldots, r(N)\}$. The aim of channel equalization (CE) is to construct an *inverse* filter that reproduces the original signal with as low an error rate as possible. It is easy to formulate CE as a regression problem, with input-output examples $\{(r(t+D), r(t+D-1), \ldots, r(t+D-l+1)), s(t)\}$, where l is the time embedding length, and D is the equalization time lag.

In this experiment, the nonlinear channel model is defined by $x(t) = s(t) + 0.5s(t-1)$, $r(t) = x(t) - 0.9x(t)^2 + n(t)$, where $n(t)$ is the white Gaussian noise with a variance of σ^2. We compare the performance of the LMS1, the APA1, the SKLMS1, the SKAPA1 ($K = 10$), and the SKAPA2 ($K = 10$). The Gaussian kernel with $a = 0.1$ is used in the SKLMS and SKAPA selected with cross validation. $l = 3$ and $D = 2$ in the equalizer. The noise variance is fixed here $\sigma = 0.1$. The learning curve is plotted in Figure 6. The MSE is calculated between the continuous output (before taking the hard decision) and the desired signal. For the SKLMS1, SKAPA1, and SKAPA2, the novelty criterion is employed with $\delta_1 = 0.07$, $\delta_2 = 0.08$. The incremental growth of the network is also plotted in Figure 7 over the training. It can be seen that at the beginning, the network sizes increase quickly, but after convergence, the network sizes increase slowly. And in fact, we can stop adding new centers after convergence by cross-validation by noticing that the MSE does not change after convergence.

Next, different noise variances are set. To make the comparison fair, we tune the novelty criterion parameters to make the network size almost the same (around 100) in each scenario by cross validation. For each setting, 20 Monte Carlo simulations are run with different training data and different testing data. The size of the training data is 1000 and the size of the testing data is 10^5. The filters are fixed during the testing phase. The results are presented in Figure 8. The normalized signal-noise ratio (SNR) is defined as $10\log_{10}(1/\sigma^2)$. It is clearly shown that the SKAPA-2 outperforms the SKLMS-1 substantially in terms of the bit error rate (BER). The linear methods never really work in this simulation regardless of the SNR. The improvement of the SKAPA-1 on the SKLMS-1 is marginal but it exhibits a smaller variance. The variability in the curves is mostly due to the variance from the stochastic training.

In the last simulation, we test the tracking ability of the proposed methods by introducing an abrupt change during training. The training data is 1500. For the first 500 data, the channel model is kept the same as before, but for the last 1000 data, the nonlinearity of the channel is switched to $r(t) = -x(t) + 0.9x(t)^2 + n(t)$. The ensemble learning curves from

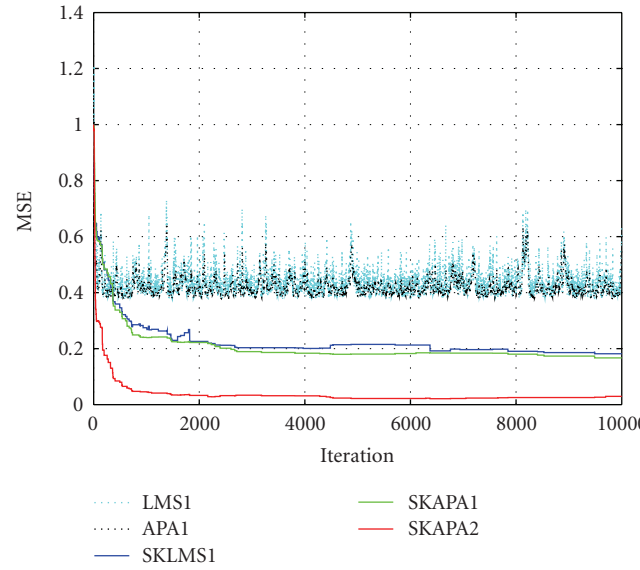

Figure 6: The learning curves of the LMS1, APA1, SKLMS1, SKAPA1, and SKAPA2 in the nonlinear channel equalization ($\sigma = 0.1$).

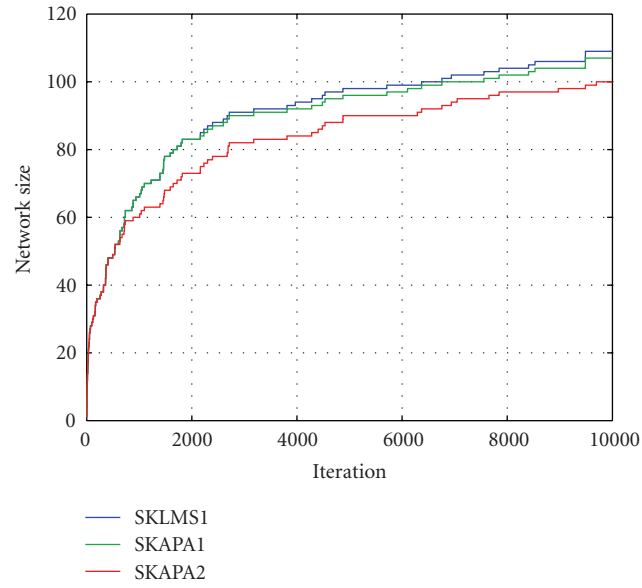

Figure 7: Network size over training in the nonlinear channel equalization.

100 Monte Carlo simulations are plotted in Figure 9, and the dynamic change of the network size is plotted in Figure 10. It is seen that the SKAPA-2 outperforms other methods with its fast tracking speed. It is also noted that the network sizes increase right after the change to the channel model.

7. DISCUSSION AND CONCLUSION

This paper proposes the KAPA algorithm family which is intrinsically a stochastic gradient methodology to solve the Least Squares problem in RKHS. It is a follow-up study of the

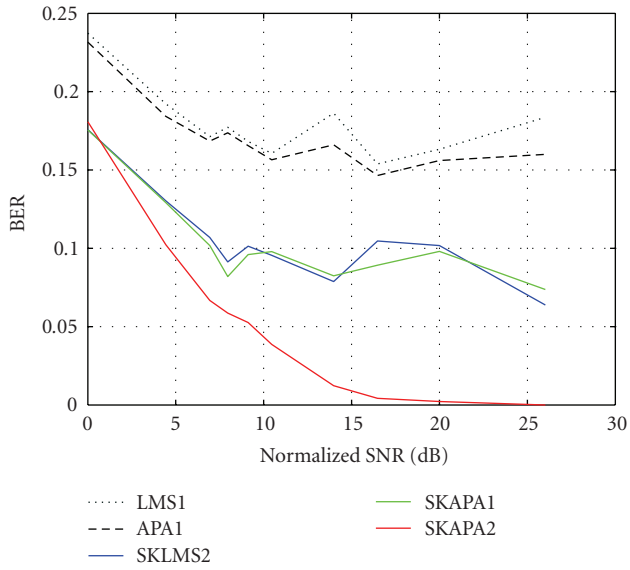

Figure 8: Performance comparison with different SNR in the nonlinear channel equalization.

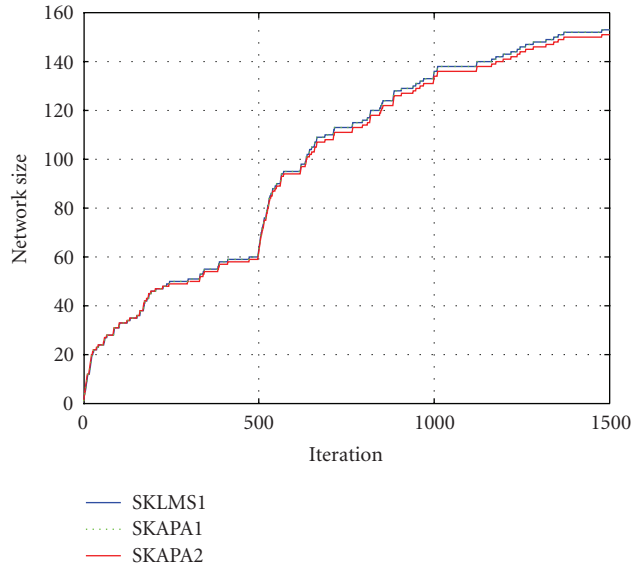

Figure 10: Network size over training in the nonlinear channel equalization with an abrupt change at iteration 500.

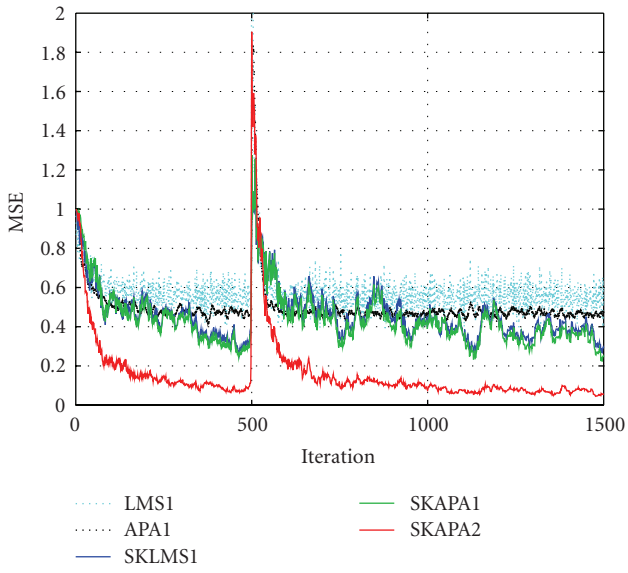

Figure 9: Ensemble learning curves in the nonlinear channel equalization with an abrupt change at iteration 5000.

recently introduced KLMS. Since the KAPA update equation can be written as inner products, KAPA can be efficiently computed in the input space. The good approximation ability of the KAPA stems from the fact that the transformed data $\varphi(\mathbf{u})$ includes possibly infinite different features of the original data. In the framework of stochastic projection, the space spanned by $\varphi(\mathbf{u})$ is so large that the projection error of the desired signal could be very small [31], as is well known from Cover's theorem [32]. This capability includes modeling of nonlinear systems, which is the main reason why the KAPA can achieve good performance in the Mackey-Glass system prediction, adaptive noise cancellation, and nonlinear channel equalization.

Comparing with the KLMS, KRLS, and regularization networks (batch mode training), KAPA gives yet another way of calculating the coefficients for shallow RBF like neural networks. The performance of the KAPA is somewhere between the KLMS and KRLS, which is specified by the window length K. Therefore, it not only provides a further theoretical understanding of RBF like neural networks, but it also brings much flexibility for application design with the constraints on performance and computation resources.

Three examples are studied in the paper, namely, time series prediction, nonlinear channel equalization, and nonlinear noise cancellation. In all examples, the KAPA demonstrates superior performance when compared with the KLMS, which is expected from the classic adaptive filtering theory.

As pointed out, the study of the KLMS and KAPA has a close relation with the resource-allocating networks, but in the framework of RKHS, any Mercer kernel can be used instead of restricting the architecture to the Gaussian kernel. An important avenue for further research is how to choose the optimal kernel for a specific problem. A lot of work [33–35] has been done in the context of classical machine learning, which is usually derived in a strict optimization manner. Notice that with stochastic gradient methods, the solution obtained is not strictly the optimal solution, therefore, further investigation is warranted. As we mentioned before, how to control the network size is still a big issue, which needs further study.

ACKNOWLEDGMENT

This work was partially supported by NSF, Grant no. ECS-0601271.

REFERENCES

[1] V. Vapnik, *The Nature of Statistical Learning Theory*, Springer, New York, NY, USA, 1995.

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

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

[4] K. I. Kim, M. O. Franz, and B. Schölkopf, "Iterative kernel principal component analysis for image modeling," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 9, pp. 1351–1366, 2005.

[5] T.-T. Frieb and R. F. Harrison, "A kernel-based adaline," in *Proceedings of the 7th European Symposium on Artificial Neural Networks (ESANN '99)*, pp. 245–250, Bruges, Belgium, April 1999.

[6] P. P. Pokharel, W. Liu, and J. C. Príncipe, "Kernel LMS," in *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP '07)*, vol. 3, pp. 1421–1424, Honolulu, Hawaii, USA, April 2007.

[7] J. Platt, "A resource-allocating network for function interpolation," *Neural Computation*, vol. 3, no. 2, pp. 213–225, 1991.

[8] A. Sayed, *Fundamentals of Adaptive Filtering*, John Wiley & Sons, New York, NY, USA, 2003.

[9] N. Aronszajn, "Theory of reproducing kernels," *Transactions of the American Mathematical Society*, vol. 68, no. 3, pp. 337–404, 1950.

[10] C. J. C. Burges, "A tutorial on support vector machines for pattern recognition," *Data Mining and Knowledge Discovery*, vol. 2, no. 2, pp. 121–167, 1998.

[11] B. Schölkopf, R. Herbrich, and A. J. Smola, "A generalized representer theorem," in *Proceedings of the 14th Annual Conference on Computational Learning Theory and 5th European Conference on Computational Learning Theory*, pp. 416–426, Amsterdam, The Netherlands, July 2001.

[12] W. Liu, P. P. Pokharel, and J. C. Príncipe, "The kernel least mean square algorithm," *IEEE Transactions on Signal Processing*, vol. 56, no. 2, pp. 543–554, 2008.

[13] F. Girosi, M. Jones, and T. Poggio, "Regularization theory and neural networks architectures," *Neural Computation*, vol. 7, no. 2, pp. 219–269, 1995.

[14] S. Van Vaerenbergh, J. Vía, and I. Santamaría, "A sliding-window kernel RLS algorithm and its application to nonlinear channel identification," in *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP '06)*, vol. 5, pp. 789–792, Toulouse, France, May 2006.

[15] J. Kivinen, A. J. Smola, and R. C. Williamson, "Online learning with kernels," *IEEE Transactions on Signal Processing*, vol. 52, no. 8, pp. 2165–2176, 2004.

[16] W. Liu, P. P. Pokharel, and J. C. Príncipe, "Recursively adapted radial basis function networks and its relationship to resource allocating networks and online kernel learning," in *Proceedings of IEEE International Workshop on Machine Learning for Signal Processing (MLSP '07)*, pp. 300–305, Thessaloniki, Greece, August 2007.

[17] G. Cavallanti, N. Cesa-Bianchi, and C. Gentile, "Tracking the best hyperplane with a simple budget Perceptron," *Machine Learning*, vol. 69, no. 2-3, pp. 143–167, 2007.

[18] S. Yonghong, P. Saratchandran, and N. Sundararajan, "A direct link minimal resource allocation network for adaptive noise cancellation," *Neural Processing Letters*, vol. 12, no. 3, pp. 255–265, 2000.

[19] O. Dekel, S. Shalev-Shwartz, and Y. Singer, "The forgetron: a kernel-based perceptron on a fixed budget," in *Advances in Neural Information Processing Systems 18*, pp. 1342–1372, MIT Press, Cambridge, Mass, USA, 2006.

[20] A. Navia-Vázquez, F. Pérez-Cruz, A. Artés-Rodríguez, and A. R. Figueiras-Vidál, "Weighted least squares training of support vector classifiers leading to compact and adaptive schemes," *IEEE Transactions on Neural Networks*, vol. 12, no. 5, pp. 1047–1059, 2001.

[21] J. A. K. Suykens, T. V. Gestel, J. D. Brabanter, B. D. Moor, and J. Vandewalle, *Least Squares Support Vector Machines*, World Scientific, Singapore, 2002.

[22] C. K. I. Williams and M. Seeger, "Using the Nyström method to speed up kernel machines," in *Advances in Neural Information Processing Systems 13*, T. K. Leen, T. G. Dietterich, and V. Tresp, Eds., pp. 682–688, chapter 13, MIT Press, Cambridge, Mass, USA, 2001.

[23] S. Fine and K. Scheinberg, "Efficient svm training using low-rank kernel representations," *Journal of Machine Learning Research*, vol. 2, pp. 242–264, 2001.

[24] A. Bordes, S. Ertekin, J. Weston, and L. Bottou, "Fast kernel classifiers with online and active learning," *Journal of Machine Learning Research*, vol. 6, pp. 1579–1619, 2005.

[25] K. Fukumizu, "Active learning in multilayer perceptrons," in *Advances in Neural Information Processing Systems 8*, D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, Eds., pp. 295–301, MIT Press, Cambridge, Mass, USA, 1996.

[26] L. Glass and M. Mackey, *From Clocks to Chaos: The Rhythms of Life*, Princeton University Press, Princeton, NJ, USA, 1988.

[27] S. Mukherjee, E. Osuna, and F. Girosi, "Nonlinear prediction of chaotic time series using support vector machines," in *Proceedings of the 7th IEEE Workshop on Neural Networks for Signal Processing*, J. C. Príncipe, L. Giles, N. Morgan, and E. Wilson, Eds., pp. 511–520, IEEE Press, Amelia Island, Fla, USA, September 1997.

[28] G. C. Goodwin and K. S. Sin, *Adaptive Filtering Prediction and Control*, Prentice Hall, Upper Saddle River, NJ, USA, 1984.

[29] G. Kechriotis, E. Zervas, and E. S. Manolakos, "Using recurrent neural networks for adaptive communication channel equalization," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 267–278, 1994.

[30] N. P. Sands and J. M. Cioffi, "Nonlinear channel models for digital magnetic recording," *IEEE Transactions on Magnetics*, vol. 29, no. 6, part 2, pp. 3996–3998, 1993.

[31] E. Parzen, "Statistical methods on time series by hilbert space methods," Tech. Rep. 23, Applied Mathematics and Statistics Laboratory, Stanford University, Stanford, Calif, USA, 1959.

[32] S. Haykin, *Neural Networks: A Comprehensive Foundation*, Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, 1998.

[33] C. A. Micchelli and M. Pontil, "Learning the kernel function via regularization," *Journal of Machine Learning Research*, vol. 6, pp. 1099–1125, 2005.

[34] A. Argyriou, C. A. Micchelli, and M. Pontil, "Learning convex combinations of continuously parameterized basic kernels," in *Proceedings of the18th Annual Conference on Computational Learning Theory (COLT '05)*, pp. 338–352, Bertinoro, Italy, June 2005.

[35] O. Chapelle, V. Vapnik, O. Bousquet, and S. Mukherjee, "Choosing multiple parameters for support vector machines," *Machine Learning*, vol. 46, no. 1–3, pp. 131–159, 2002.