ONLINE LEARNING WITH KERNELS: OVERCOMING THE GROWING SUM PROBLEM

Abhishek Singh, Narendra Ahuja and Pierre Moulin

Beckman Institute for Advanced Science and Technology
University of Illinois at Urbana-Champaign
abhishek_singh@ieee.org, {n-ahuja, pmoulin}@illinois.edu

ABSTRACT

Online kernel algorithms have an important computational drawback. The computational complexity of these algorithms grow linearly over time. This makes these algorithms difficult to use for real time signal processing applications that need to continuously process data over prolonged periods of time. In this paper, we present a way of overcoming this problem. We do so by approximating kernel evaluations using finite dimensional inner products in a randomized feature space. We apply this idea to the Kernel Least Mean Square (KLMS) algorithm, that has recently been proposed as a non-linear extension to the famed LMS algorithm. Our simulations show that using the proposed method, constant computational complexity can be achieved, with no observable loss in performance.

1. INTRODUCTION

Online learning is a key concept used in several domains such as controls (system identification and tracking), computer vision (visual tracking, video surveillance), signal processing (active noise cancellation, echo cancellation) etc [4].

Over the past decade or so, there has been considerable research in trying to incorporate kernel methods for online signal processing problems, using stochastic gradient based learning. The motivation has been to combine two ideas - 1) Kernel methods offer a principled and a mathematically sound way of learning non-linear functions, and, 2) Stochastic gradient based learning allows for making updates after seeing each sample, making the learning process 'online'.

A number of related methods have been proposed that perform online learning in a functional space induced by kernels. Examples of such algorithms are the NORMA algorithm [5], the PEGASOS algorithm [12], the Kernel Least Mean Square (KLMS) algorithm [6], among others.

In this paper, we investigate an important limitation of learning online in functional spaces. The time and memory complexity of online kernel algorithms grows linearly with time. This is a natural fallout of the Representer theorem, which expresses the function to be learnt as a linear combination of kernel functions centered on the training samples. As the number of training samples grows, this linear combination grows as well. This prevents these algorithms from being applicable for real time signal processing applications, which often need to process streaming data over prolonged period of time. Most existing approaches to curb this problem are based on suitably sparsifying the dataset to carefully weed out 'uninformative' samples so that the growth in complexity is sublinear.

In this paper, instead of approximating the training data, we approximate the kernel functions. We use the recent idea of approximating kernel evaluations using finite dimensional inner products. More specifically, we use the *random Fourier features* proposed in [9] to map the input data into a finite dimensional space, where inner products closely approximate kernel evaluations. We propose to use these features in an online stochastic gradient optimization setting of the KLMS algorithm. We show that on doing so, we can achieve constant time and memory complexity, just like the simple linear LMS algorithm. The proposed algorithm, called the RFF-KLMS algorithm, enables the practical use of online algorithms like KLMS for real time and large scale signal processing applications.

In the next section, we formalize the online learning setting, and present a generic formulation for kernel based online learning algorithms. In Section 3, review the Kernel LMS algorithm, that has been recently proposed as a non-linear extension to the LMS algorithm. In Sections 4 we discuss the problem of the growing sum in the KLMS algorithm and the other related algorithms, and review some existing methods that have attempted to alleviate this problem. In Section 5, we describe the proposed RFF-KLMS algorithm that approximates the KLMS algorithm but in constant time/memory complexity. In Section 6, we present our simulation results.

2. ONLINE LEARNING WITH KERNELS

Consider a set of T pairs of training input vectors and corresponding labels, $\{(\mathbf{x}_t, y_t)\}_{t=1}^{T}$. Let $f_t(.)$ be the trained predictor at time t. When a new sample $\mathbf{x}_t$ is observed at time t, the predictor makes a prediction $\hat{y}_t = f_t(\mathbf{x}_t)$. The true label y_t is then observed, and a loss function $l(\hat{y}_t, y_t)$ is evaluated. The goal of online learning is to update the predictor

978-1-4673-1026-0/12/$31.00 ©2012 IEEE

$f_t(.) \rightarrow f_{t+1}(.)$ such that the cumulative loss $\sum_t l(\hat{y}_t, y_t)$ is minimized.

The function f is usually assumed to be of the form,

$$f(\mathbf{x}) = \langle \Omega, \phi(\mathbf{x}) \rangle, \tag{1}$$

where $\phi(\mathbf{x}) = \kappa(\mathbf{x}, \cdot)$ maps the input vector x into a functional space induced by the positive definite kernel function κ. Learning the function f amounts to learning the weight vector Ω in the functional space.

Instead of optimizing the cumulative loss directly, we optimize a stochastic estimate of the loss in order to facilitate online learning. We can therefore define an instantaneous risk as,

$$R_{inst}(\Omega, \lambda, t) = l(f(\mathbf{x}_t), y_t) + \frac{\lambda}{2}\|\Omega\|^2. \tag{2}$$

Using the stochastic gradient descent procedure, the weight vector Ω can be iteratively updated using the following update rule:

$$\Omega_{t+1} = \Omega_t - \mu \left[\frac{\partial R_{inst}(f, \lambda, t)}{\partial \Omega}\right]_{\Omega=\Omega_t}. \tag{3}$$

The gradient of the instantaneous risk can be evaluated as,

$$\left[\frac{\partial R_{inst}(\Omega, \lambda, t)}{\partial \Omega}\right]_{\Omega=\Omega_t} = l'(f_t(\mathbf{x}_t), y_t)\kappa(\mathbf{x}_t, \cdot) + \lambda\Omega_t. \tag{4}$$

The stochastic gradient update rule therefore becomes,

$$\begin{aligned} \Omega_{t+1} &= (1-\mu\lambda)\Omega_t - \mu l'(f_t(\mathbf{x}_t), y_t)\kappa(\mathbf{x}_t, \cdot) & (5)\\ \Rightarrow \quad f_{t+1}(\mathbf{x}) &= (1-\mu\lambda)f_t(\mathbf{x}) - \mu l'(f_t(\mathbf{x}_t), y_t)\kappa(\mathbf{x}_t, \mathbf{x}) & (6) \end{aligned}$$

(5) is the fundamental idea behind most online learning algorithms in literature. However, (5) is not directly applicable in practical settings, since the weight vector Ω belongs to a possibly infinite dimensional functional space, and is not directly accessible. Therefore, for practical computations, the function f_t is expressed as an expansion using the Representer theorem as,

$$f_t(\mathbf{x}) = \sum_{i=1}^{t-1} \alpha_i \kappa(\mathbf{x}_i, \mathbf{x}) \tag{7}$$

Using Eqns. 6 and 7, we obtain,

$$\sum_{i=1}^{t} \alpha_i \kappa(\mathbf{x}_i, \mathbf{x}) = (1-\mu\lambda)\sum_{i=1}^{t-1} \alpha_i \kappa(\mathbf{x}_i, \mathbf{x}) - \mu l'(f_t(\mathbf{x}_t), y_t)\kappa(\mathbf{x}_t, \mathbf{x}) \tag{8}$$

Therefore, we can formulate an update rule for the coefficients α_i as,

$$\alpha_i^+ = \begin{cases} -\mu l'(f_t(\mathbf{x}_t), y_t), & \text{if } i = t. \\ (1-\mu\lambda)\alpha_i, & \text{if } i < t. \end{cases} \tag{9}$$

This general class of algorithms has been called NORMA (Naive Online R_{reg} Minimization Algorithm) [5]. A closely related method called PEGASOS (Primal Estimated Sub-Gradient Solver for SVM) [12] has also been proposed using a similar framework for the special case of using the hinge loss function.

Another interesting case arises when the square loss function is used, without the explicit regularization term. The resultant algorithm is called the Kernel LMS algorithm [6], which we describe in the next section.

3. KERNEL LMS ALGORITHM

The KLMS algorithm attempts to extend the classical LMS algorithm for learning non-linear functions, using kernels [6]. By using a kernel function $\Phi(.)$, the KLMS algorithm first maps the input vectors $\mathbf{x}$ to $\Phi(\mathbf{x})$. An instantaneous risk functional using the square loss can be defined as,

$$R_{inst}(\Omega, t) = (\langle \Omega, \Phi(\mathbf{x}_t) \rangle - y_t)^2. \tag{10}$$

The stochastic gradient descent update rule for the infinite dimensional weight vector Ω can be formulated as,

$$\begin{aligned} \Omega_{t+1} &= \Omega_t - \mu \left[\frac{\partial R_{inst}(\Omega, t)}{\partial \Omega}\right]_{\Omega=\Omega_t} & (11)\\ \Rightarrow \quad \Omega_{t+1} &= \Omega_t + \mu\left(y_t - \langle \Omega_t, \Phi(\mathbf{x}_t)\rangle\right)\Phi(\mathbf{x}_t) & (12)\\ &= \Omega_t + \mu e_t \Phi(\mathbf{x}_t) & (13) \end{aligned}$$

As before, since the sample $(\mathbf{x}_t, y_t)$ is not used for computing Ω_t the error $e_t = y_t - \langle \Omega_t, \Phi(\mathbf{x}_t)\rangle$ is the *test* error.

Again, (13) is not practically usable since the weight vector Ω_t resides in a possibly infinite dimensional functional space. Therefore, (13) can be recursively applied, starting with $\Omega_1 = 0$, to obtain,

$$\Omega_{t+1} = \mu \sum_{i=1}^{t} e_i \Phi(\mathbf{x}_i). \tag{14}$$

To obtain the prediction $f(\mathbf{x})$ for a test sample $\mathbf{x}$, we now simply evaluate the inner product,

$$\begin{aligned} f(\mathbf{x}) &= \langle \Omega_{t+1}, \Phi(\mathbf{x}) \rangle = \mu \sum_{i=1}^{t} e_i \langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}) \rangle & (15)\\ &= \mu \sum_{i=1}^{t} e_i \kappa(\mathbf{x}_i, \mathbf{x}). & (16) \end{aligned}$$

(16) is obtained by using the kernel-trick. The above equation is the KLMS algorithm, as proposed in [6].

It has been shown in [6] that in the finite data case, the KLMS algorithm is well posed in the functional space, without the addition of an extra regularizer. Not having to choose a regularization parameter makes the KLMS algorithm simpler to use from a practical standpoint.

4. THE GROWING SUM PROBLEM

From the update rule(s) in (9), we see that a new coefficient is introduced for every new training sample, while all the previous ones are updated. Therefore, the number of coefficients grows with the number of training samples seen. This is, of course, a natural consequence of using the Representer theorem in (7), which parameterizes the current prediction in terms of a sum of kernel evaluations on all previously seen data. In the KLMS algorithm as well, the prediction function in (16) involves a summation over all previously seen training data, and this grows with the number of samples seen. This is quite disturbing for any practical, real time application.

There have been quite a few attempts at overcoming the growing complexity of online learning algorithms. In [5] it was proposed to use the regularization term of the NORMA algorithm as a 'forgetting' factor, which would downweight the effect of previously seen samples. A sliding window approach could then be used to truncate the samples that fall beyond this window.

For the particular case of the KLMS algorithm, a way of constraining the the growth of the summation was proposed in [7], in which new feature vectors that are linearly dependent on the previous training vectors, are not used to update the model. This helps slow down the rate of growth of the summation. A more recent attempt at curbing the growth proposes to quantize the input space, in an online manner [3]. Quantization of the input space into a small number of 'centers' reduces redundancy in among the input vectors. For each new input vector, only the center closest to it is used for updating the model.

All these methods have greatly improved over the plain vanilla online kernel algorithm(s). However, they still have some drawbacks:

1. The above approaches do not fully solve the growing sum problem. They curb the rate of growth from linear to sublinear. The algorithms, however, still have growing complexity. For real time applications that require running the learning algorithms for long periods of time under non-stationary conditions, these algorithms are eventually bound to grow beyond control.

2. All the above algorithms essentially aim to constrain the training samples, either by quantization [3] or sparsification [7], or simply a sliding window [5]. However, in a non-Bayesian , supervised learning scenario, the training data is the *only* source of information available for learning the model. Some information is bound to be lost if the training data is constrained using the quantization or sparsification step.

3. The data constraining strategies such as quantization or sparsification introduce additional computational burden, at each iteration.

5. PROPOSED SOLUTION

We take a fundamentally different approach in addressing the growing sum problem of online learning. Instead of trying to constrain the input vectors, and curb the rate of growth of the sum, we try to address the problem at its root. We observe that the problem of the growing sum arises due to the Representer theorem of (7). The Representer theorem, in turn, is required since the 'weight vector' representation of the function is impractical to work with in functional spaces due to their infinite dimensionality. We therefore, propose to use *finite* dimensional approximations of the function weights (and the feature vectors). In other words, we propose to project the input vectors into an appropriate *finite* dimensional space, where the inner products are close approximations to the original infinite dimensional inner products (which are simply kernel evaluations using the 'trick').

Therefore, we want to look for a *finite* dimensional mapping $\Psi : \mathbb{R}^d \rightarrow \mathbb{R}^D$ such that,

$$\langle \Psi(\mathbf{x}), \Psi(\mathbf{y}) \rangle \approx \kappa(\mathbf{x}, \mathbf{y}). \tag{17}$$

The feature vectors $\Psi(\mathbf{x})$ are obtained using randomized feature maps. Randomization is a relatively old principle used in learning. For instance, using random networks of nonlinear functions for regression problems has been known to be empirically quite successful [2, 10].

More recently, inner products of random features have been proposed for approximating commonly used kernel functions such as Gaussians [9], and this has been successfully used for large scale machine learning applications [8].

In this work, we propose to use these random feature approximations for kernels for the stochastic gradient descent based online learning algorithms (in particular, KLMS algorithm).

Consider a Mercer kernel $\kappa(\cdot, \cdot)$ that satisfies the following for all input vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$:

- A1: It is translation or shift invariant. That is, $\kappa(\mathbf{x}, \mathbf{y}) = \kappa(\mathbf{x} - \mathbf{y})$.

- A2: It is normalized. That is, $\kappa(0) = 1$ and $\kappa(x - y) \leq 1$.

A way of approximating kernels satisfying A1 and A2 using inner products of finite dimensional random mappings was proposed in [9]. The underlying idea is based on Bochner's theorem [11], which is a fundamental result in harmonic analysis which states that any kernel function κ satisfying A1 and A2 above is a Fourier transform of a uniquely defined probability measure on $\mathbb{R}^d$.

Theorem 1 *(Bochner's Theorem) A kernel $\kappa(\boldsymbol{x}, \boldsymbol{y}) = \kappa(\boldsymbol{x} - \boldsymbol{y})$ on $\mathbb{R}^d$ is positive definite if and only if κ is the Fourier transform of a non-negative measure.*

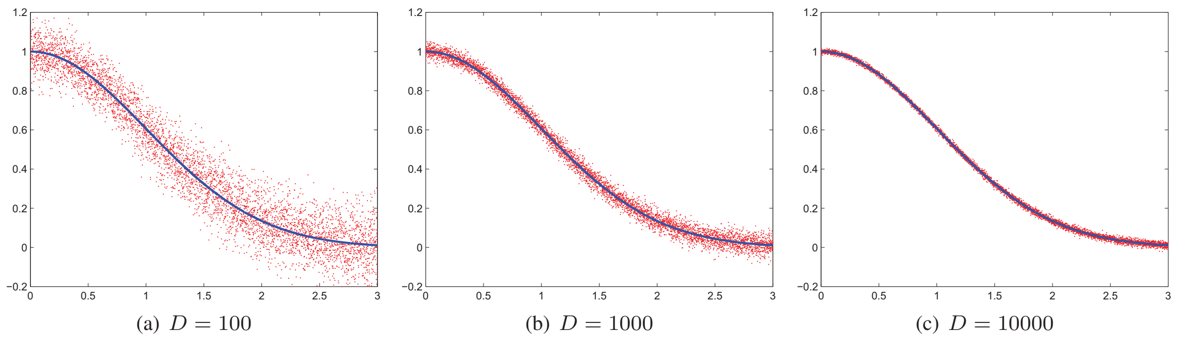


Fig. 1. Qualitative view of the random Fourier feature (RFF) inner product approximation of the Gaussian kernel. The blue curve shows the exact Gaussian function. The red scatter plot shows the approximation, for different values of D. (a) $D = 100$, (b) $D = 1000$, (c) $D = 10000$. Clearly, the variance in approximation falls with increasing D.

Bochner's theorem guarantees that the Fourier transform of an appropriately scaled, shift invariant kernel is a probability density. Defining $z_\omega(\mathbf{x}) = e^{j\omega^T\mathbf{x}}$, we get,

$$\kappa(\mathbf{x}-\mathbf{y}) = \int_{\mathcal{R}^d} p(\omega)e^{j\omega^T(\mathbf{x}-\mathbf{y})}d\omega = E_\omega\left[z_\omega(\mathbf{x})z_\omega(\mathbf{y})^*\right] \tag{18}$$

Therefore, $z_\omega(\mathbf{x})z_\omega(\mathbf{y})^*$ is an unbiased estimate of $\kappa(\mathbf{x},\mathbf{y})$ when ω is drawn from p. To reduce the variance of this estimate, we can take a sample average of D randomly chosen $z_\omega(.)$. Therefore, the D-dimensional inner product $\frac{1}{D}\sum_{j=1}^{D} z_{\omega_j}(\mathbf{x})z_{\omega_j}(\mathbf{y})^*$ is a low variance approximation to the kernel evaluation $\kappa(\mathbf{x},\mathbf{y})$. This approximation improves exponentially fast in D [9].

Note that in general, the features $z_\omega(\mathbf{x})$ are complex. However, we can exploit the symmetry property of the kernel κ, in which case it can be expressed using real valued cosine bases. Therefore the mapping,

$$\psi_\omega(\mathbf{x}) = \sqrt{2}\cos(\omega^T\mathbf{x}+b) \tag{19}$$

where ω is drawn from p and b is drawn uniformly from $[0,2\pi]$, also satisfies the condition $E_\omega\left[\psi_\omega(\mathbf{x})\psi_\omega(\mathbf{y})\right] = \kappa(\mathbf{x}-\mathbf{y})$ [9].

The vector $\Psi(\mathbf{x}) = [\psi_{\omega_1}(\mathbf{x}), \psi_{\omega_2}(\mathbf{x}), ..., \psi_{\omega_D}(\mathbf{x})]$ is therefore a D-dimensional *random Fourier feature* (RFF) of the input vector $\mathbf{x}$. This mapping satisfies the approximation in (17). For approximating the commonly used Gaussian kernel of width σ, we therefore draw ω_i, $i = 1, ..., D$, from the Normal$(0, I_d/\sigma^2)$ distribution. The quality of the approximation is controlled by the dimensionality of the mapping, D, which is a user defined parameter. Figure 1 shows a qualitative view of this approximation, for different values of D. We try to approximate $\kappa(\mathbf{x},0)$ using the D-dimensional inner product $\langle\Psi(\mathbf{x}),\Psi(\mathbf{0})\rangle$ as described above. The red scatterplot shows the variance in the approximation, and the blue curve shows the exact Gaussian kernel function.

Going back to the KLMS algorithm, instead of using exact Gaussian kernel evaluations for learning, we use the random Fourier feature (RFF) space to first explicitly map the input data vectors. Since this space is finite dimensional, it now becomes possible to directly work with the filter weights in this space. Therefore, (13) can be used directly, without invoking the Representer theorem. In fact, the update equations in this random Fourier feature space become exactly the same as the classical LMS algorithm.

The proposed RFF-KLMS algorithm is described below:

```
**Algorithm** *RFF-KLMS*
**Input:** Sequential training data $\{\mathbf{x}_t, y_t\}_{t=1}^{T}$, Gaussian kernel
    width $\sigma$, step size $\mu$, RFF dimension $D$.
1.  Draw i.i.d. $\{\omega_i\}_{i=1}^{D}$ from $\mathcal{N}(0, I_d/\sigma^2)$, where $d$ is the
    input space dimension.
2.  Draw i.i.d. $\{b_i\}_{i=1}^{D}$ from $Uniform[0, 2\pi]$.
3.  Initialize $\Omega_1 = 0 \in \mathbb{R}^D$
4.  for $t \leftarrow 1$ to $T$
5.          Compute Random Fourier Feature (RFF) vector $\Psi(\mathbf{x}_t) = [\psi_{\omega_1}(\mathbf{x}_t), ..., \psi_{\omega_D}(\mathbf{x}_t)]$, where each
            $\psi_{\omega_i}(\mathbf{x}_t) = \cos(\omega_i^T\mathbf{x}_t + b_i)$;
6.          Compute prediction $\hat{y}_t = \langle\Omega_t, \Psi(\mathbf{x}_t)\rangle$;
7.          Compute error $e_t = y_t - \hat{y}_t$;
8.          Update $\Omega_{t+1} = \Omega_t + \mu e_t\Psi(\mathbf{x}_t)$;
```

6. SIMULATION RESULTS

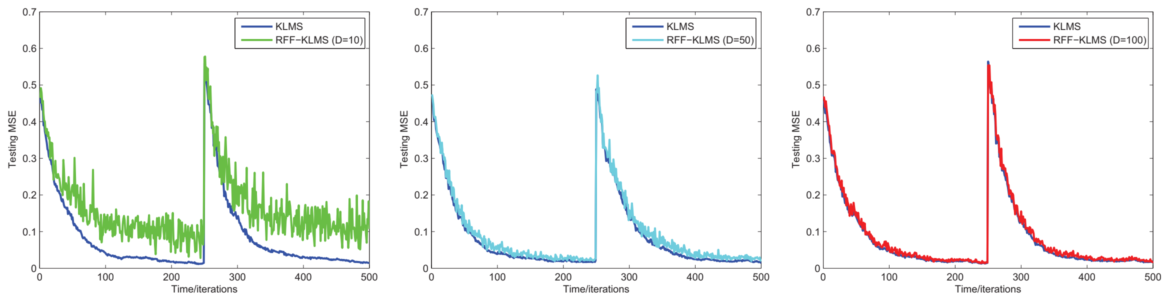
6.1. Non-Stationary Regression

In our first simulation we consider a time varying regression function as follows:

$$X_t \sim Uniform[0,1], \qquad t = 1, 2, ..., 500. \tag{20}$$

$$Y_t = \begin{cases} \sin(10X_t) + W_t, & \text{if } t \leq 250 \\ \sin(12X_t) + W_t, & \text{if } t > 250 \end{cases} \tag{21}$$

where $W_t \sim \mathcal{N}(0, 0.5)$ is i.i.d. observation noise, independent of X_t. Fig. 3 shows a realization from the above model. Clearly, the function is highly non-linear and there is a significant change in function at $t = 250$.



(a) $D = 10$ (b) $D = 50$ (c) $D = 100$

Fig. 2. Plots of MSE on test set vs. number of training iterations/samples, for the KLMS algorithm, and the RFF-KLMS algorithm for different values of D, for the time-varying regression problem. (a) $D = 10$, (b) $D = 50$, (c) $D = 100$. For $D = 100$, there is no observable difference in performance between the KLMS and RFF KLMS algorithms.

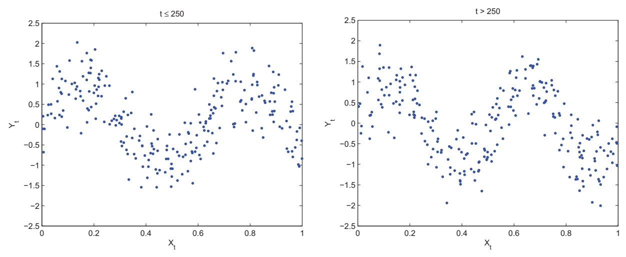


Fig. 3. Realization from the time varying regression function described in Eqns. 20 and 21. *Left:* $t \leq 250$. *Right:* $t > 250$. Clearly the function is highly non-linear, and cannot be learnt by a linear model. Non linear methods in batch mode are also not suited for tracking such a time varying function.

We train the KLMS algorithm and the proposed RFF-KLMS algorithm with the 500 samples from the above model. We also generate 100 different i.i.d. samples from this model as a test dataset. These test samples are not corrupted by the additive noise W. Cross validation using such a test set allows us to see how well the model is able to learn the true underlying function, using noisy training samples.

After each update step in the KLMS and the RFF-KLMS algorithm, we cross validate the model learnt thus far using the 100 test samples. We compute the mean squared error (MSE) as the performance statistic. We plot the MSE as a function of the training iterations.

We have used step size $\mu = 0.1$ for both algorithms, and Gaussian kernel width parameter $\sigma = 0.1$ for this experiment.

Fig. 2 shows our results. We compare the performance of the KLMS algorithm with the proposed RFF-KLMS algorithm, for different values of D. Clearly, having a very small value of D creates high variance in the approximation of the Gaussian kernel, and performance suffers. For $D = 100$, the RFF-KLMS performs very similar to the KLMS algorithm. Note that each plot is obtained after averaging over 10 trials with different i.i.d. training and testing data, and different i.i.d. sampling of ω_i and b_i for the RFF-KLMS algorithm.

The more interesting result is shown in Fig. 4, which

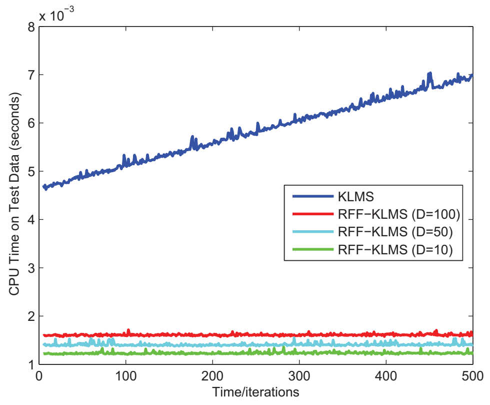


Fig. 4. CPU time required for testing, as a function of the number of training iterations/samples, for the KLMS algorithm and the RFF-KLMS algorithm with $D = 50, 100, 500$, for the time-varying regression problem. The KLMS algorithm has a linearly growing CPU time requirement due to the growing summation. The proposed RFF-KLMS algorithm has a constant complexity.

shows the CPU time required for computing predictions on the test data, after each training iteration/sample. Due to the growing sum problem in the KLMS algorithm, the CPU time required for testing/prediction grows linearly with the number of training samples seen. On the other hand, the proposed RFF-KLMS algorithm runs in constant time, which is also several times faster than the KLMS algorithm.

6.2. Classification of Diabetes Data

The Pima-Indians Diabetes dataset [1] consists of 8 physiological measurements (features) from 768 subjects. The objective is to classify a test subject into either the diabetic or non-diabetic group, based on these physiological measurements. We use 500 samples as the training data and the rest as a test set. We perform a similar experiment as before, where both KLMS and RFF-KLMS are trained with the training data, one sample at a time. We compute the generalization

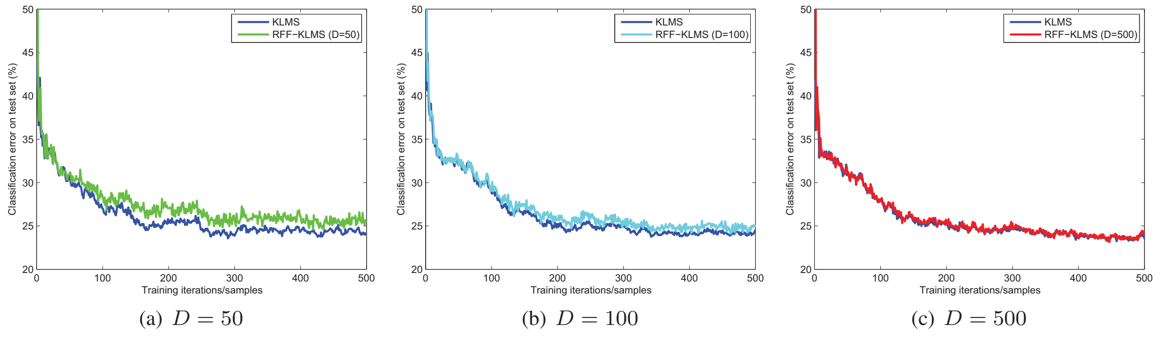


Fig. 5. Plots of Diabetes dataset classification error vs. number of training iterations/samples, for the KLMS algorithm, and the RFF-KLMS algorithm for different values of D. (a) $D = 50$, (b) $D = 100$, (c) $D = 500$. For $D = 500$, there is no observable difference in performance between the KLMS and RFF KLMS algorithms.

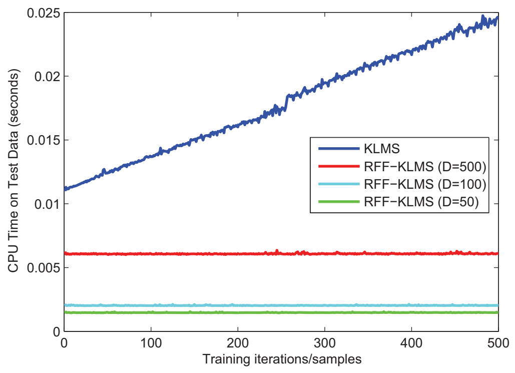


Fig. 6. CPU time required for testing, as a function of the number of training iterations/samples, for the KLMS algorithm and the RFF-KLMS algorithm with $D = 50, 100, 500$, for the Diabetes classification problem. The KLMS algorithm has a linearly growing CPU time requirement due to the growing summation. The proposed RFF-KLMS algorithm has a constant complexity.

performance in terms of classification error on the test set, after each training iteration.

We use the same stepsize $\mu = 0.1$ for both algorithms. The kernel width was chosen to be $\sigma = 3$ for the 8-dimensional isometric Gaussian kernel in this problem.

Fig. 5 shows classification error (on test set) vs. training samples/iterations, for the KLMS algorithm and the RFF-KLMS algorithm with $D = 50, 100, 500$. In the case when $D = 500$, there is no observable difference between the RFF-KLMS and the KLMS algorithm performance. From Fig. 6, we see that RFF-KLMS is much faster than KLMS, and more importantly, the computation time remains constant.

7. CONCLUSION

We have presented a way of overcoming the growing sum problem of the KLMS algorithm. We do so by approximating the kernel function using *finite* dimensional inner products in an appropriately computed randomized feature space. Our simulations have shown that we are able to achieve the same performance as the conventional KLMS algorithm, with the simple computational complexity of training linear filters.

We have restricted ourselves to the KLMS algorithm in this work, but it would be interesting to apply the same ideas to other online algorithms such as PEGASOS or NORMA. We would also be looking at exhaustively comparing the proposed algorithm with other existing approaches for curbing growth.

Since the use of kernel methods for signal processing tasks has become an important research area of late, we believe the the ideas presented in this work can help bridge the gap between some well grounded theory, and practical implementation for real time, large scale signal processing systems.

8. REFERENCES

[1] A. Asuncion and D. Newman. UCI machine learning repository, 2007.

[2] H. D. Block. The perceptron: A model for brain functioning. i. *Rev. Mod. Phys.*, 34:123–135, Jan 1962.

[3] B. Chen, S. Zhao, P. Zhu, and J. Principe. Quantized kernel least mean square algorithm. *Neural Networks and Learning Systems, IEEE Transactions on*, 23(1):22 –32, jan. 2012.

[4] S. Haykin. *Adaptive filter theory*. Prentice-Hall, Inc., 1991.

[5] J. Kivinen, A. Smola, and R. Williamson. Online learning with kernels. *Signal Processing, IEEE Transactions on*, 52(8):2165 – 2176, aug. 2004.

[6] W. Liu, P. Pokharel, and J. Principe. The kernel least-mean-square algorithm. *Signal Processing, IEEE Transactions on*, (2):543 –554, 2008.

[7] P. P. Pokharel, W. Liu, and J. C. Principe. Kernel least mean square algorithm with constrained growth. *Signal Processing*, 89(3):257 – 265, 2009.

[8] M. Raginsky and S. Lazebnik. Locality-sensitive binary codes from shift-invariant kernels. In *NIPS*, pages 1509–1517, 2009.

[9] A. Rahimi and B. Recht. Random features for large-scale kernel machines. In *In NIPS*, 2007.

[10] A. Rahimi and B. Recht. Uniform approximation of functions with random bases. In *Communication, Control, and Computing, 2008 46th Annual Allerton Conference on*, pages 555 –561, sept. 2008.

[11] M. Reed and B. Simon. *Methods of Modern Mathematical Physics II: Fourier Analysis, Self-Adjointness*. Academic Press, 1975.

[12] Y. Singer and N. Srebro. Pegasos: Primal estimated sub-gradient solver for svm. In *In ICML*, pages 807–814, 2007.