MODIFICATIONS TO THE SLIDING-WINDOW KERNEL RLS ALGORITHM FOR TIME-VARYING NONLINEAR SYSTEMS: ONLINE RESIZING OF THE KERNEL MATRIX

Brian J. Julian

Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
Cambridge, Massachusetts, USA
Email: {bjulian}@mit.edu

ABSTRACT

A kernel-based recursive least-squares algorithm that implements a fixed size "sliding-window" technique has been recently proposed for fast adaptive nonlinear filtering applications. We propose a methodology of resizing the kernel matrix to assist in system identification of time-varying nonlinear systems. To be applicable in practice, the modified algorithm must preserve its ability to operate online. Given a bound on the maximum kernel matrix size, we define the set of all obtainable sizes as the resizing range. We then propose a simple online technique that resizes the kernel matrix within the resizing range. The modified algorithm is applied to the nonlinear system identification problem that was used to evaluate the original algorithm. Results show that an increase in performance is achieved without increasing the original algorithm's computation time.

***Index Terms*—** identification, learning systems, least squares methods, nonlinear filters, time-varying filters

1. INTRODUCTION

The design of fast adaptive nonlinear filters is a heavily researched topic in the field of signal processing. Traditionally, algorithms based on kernel methods were not capable of online operation since they inherently caused the kernel matrix to grow without bound [1]. To address this issue, the sliding-window kernel recursive least-squares (RLS) algorithm was proposed. This algorithm implements a "sliding-window" technique that discards all but the N most recent data inputs [2, 3]. Moreover, a downsizing/upsizing technique allows the algorithm to calculate the $N \times N$ regularized and inverse regularized kernel matrices[1] in $O(N^2)$ time instead of $O(N^3)$ time.

Consider an $N \times N$ "sliding-window" kernel matrix used to adaptively filter a nonlinear system. If the system is time-invariant, the N most recent data inputs accurately represent the current system. This condition is not the case following an abrupt system change; "remembering" irrelevant data inputs decreases the performance of the filter. This decrease in performance provides the motivation to resize the "sliding-window" kernel matrix depending on the system's behavior. However, operations to the kernel matrix are computationally expensive, especially for online algorithms.

In this paper we provide the groundwork for developing resizing techniques for the sliding-window kernel RLS algorithm. Given a maximum kernel matrix size $N \times N$, we can calculate the online computation time allocated for kernel matrix operations. This finite capacity limits how the modified algorithm can resize the "sliding-window" kernel matrix, forming the resizing range. We show through a simple technique that using the resizing range can increase filter performance for time-varying nonlinear systems.

[1] Henceforth referred to as the regularized kernel matrices

2. SLIDING-WINDOW KERNEL RLS ALGORITHM

2.1. Regularized Least-Squares

The regularized least-squares method is commonly used for system identification in offline machine learning problems [4, 5, 6]. Given a training matrix of n independent and identically distributed vector inputs

$$\mathbf{X} = (\vec{x}_1, \ldots, \vec{x}_n)^T \tag{1}$$

and their respective scalar outputs

$$\vec{Y} = (y_1, \ldots, y_n)^T \tag{2}$$

the regularized least-squares method learns a function in the reproducing kernel Hilbert space (RKHS) by solving

$$\min_{f \in \mathcal{H}} \left[\sum_{i=1}^{n} (f(\vec{x}_i) - y_i)^2 + \lambda \|f\|_{\mathcal{H}}^2 \right] \tag{3}$$

where $(f(\vec{x}_i) - y_i)$ is the empirical error on the ith data entry and $\|f\|_{\mathcal{H}}$ is a regularization term weighted by the constant λ. By defining a positive definite kernel κ with a feature map Φ, we can transform the training data from the input space into a high-dimensional feature space

$$\kappa(\vec{x}_i, \vec{x}_j) = < \Phi(\vec{x}_i), \Phi(\vec{x}_j) >_{\mathcal{H}} \tag{4}$$

Since the representer theorem implies that the learned function f in RKHS can be expressed as

$$f(\vec{x}) = \sum_{i=1}^{n} c_i \kappa(\vec{x}_i, \vec{x}), \quad c_i \in \mathbb{R} \tag{5}$$

we can rewrite (3) as

$$\min_{\vec{c} \in \mathbb{R}^n} \left[\|\mathbf{K}c - \vec{Y}\|_2^2 + \lambda c^T \mathbf{K} c \right] \tag{6}$$

where the kernel matrix $\mathbf{K}$ is defined by

$$\mathbf{K} = \kappa(\mathbf{X}^T, \mathbf{X}^T) \tag{7}$$

Solving for (6) results in the learned function coefficients

$$\vec{c} = (\mathbf{K} + \lambda \mathbf{I})^{-1} \vec{Y} = \mathbf{G}^{-1} \vec{Y} \tag{8}$$

where $\mathbf{G}$ is the regularized kernel matrix.

Thus, for any future data input, we can use the trained coefficients in (5) to make a prediction for the output.

2.2. "Sliding-Window" Technique

As previously stated, kernel methods such as regularized least-squares are traditionally used for offline applications. Applied to an online application, the kernel matrix would grow infinitely large as n tends to ∞. Even if the size of the kernel matrix is limited to $N \times N$, the algorithm would require $O(N^3)$ time to recalculate the regularized kernel matrices after each data input.

To address the shortcomings in online use of the regularized least-squares method, Van Vaerenbergh et al. proposed the "sliding-window" technique [2, 3]. Suppose the $N \times N$ regularized kernel matrix $\mathbf{G}_n$ is calculated from data inputs $\{\vec{x}_{n-N}, \ldots, \vec{x}_n\}$. Instead of directly calculating $\mathbf{G}_{n+1}$ from data inputs $\{\vec{x}_{n-N+1}, \ldots, \vec{x}_{n+1}\}$, we first downsize $\mathbf{G}_n$ by extracting the contribution from $\vec{x}_{n-N}$ to obtain

$$\check{\mathbf{G}}_n = \begin{bmatrix} \mathbf{G}_n(2,2) & \cdots & \mathbf{G}_n(2,N) \\ \vdots & \ddots & \vdots \\ \mathbf{G}_n(N,2) & \cdots & \mathbf{G}_n(N,N) \end{bmatrix} \tag{9}$$

and then upsize $\check{\mathbf{G}}_n$ by importing the data input $\vec{x}_{n+1}$ to obtain

$$\mathbf{G}_{n+1} = \begin{bmatrix} \check{\mathbf{G}}_n & \kappa(\mathbf{X}_n, \vec{x}_{n+1}) \\ \kappa(\vec{x}_{n+1}, \mathbf{X}_n) & \kappa(\vec{x}_{n+1}, \vec{x}_{n+1}) + \lambda \end{bmatrix} \tag{10}$$

where

$$\mathbf{X}_n = (\vec{x}_{n-N+1}, \ldots, \vec{x}_n)^T \tag{11}$$

The inverse regularized kernel matrix $\mathbf{G}_n^{-1}$ can also be computed through an upsizing/downsizing technique. Using the definitions located in Appendix A, we first downsize $\mathbf{G}_n^{-1}$ to obtain

$$\check{\mathbf{G}}_n^{-1} = \mathbf{D} - \vec{f}\vec{f}^T / \mathbf{G}_n^{-1}(1,1) \tag{12}$$

and then upsize $\check{\mathbf{G}}_n^{-1}$ to obtain

$$\mathbf{G}_{n+1}^{-1} = \begin{bmatrix} \check{\mathbf{G}}_n^{-1}(\mathbf{I} + \vec{b}\,\vec{b}^T(\check{\mathbf{G}}_n^{-1})^H g) & -\check{\mathbf{G}}_n^{-1}\vec{b}g \\ -(\check{\mathbf{G}}_n^{-1}\vec{b})^T g & g \end{bmatrix} \tag{13}$$

By implementing the "sliding-window" technique for online applications, we can calculate the updated function coefficients for (5) in $O(N^2)$ time.

3. RESIZING THE KERNEL MATRIX

3.1. Online Computation Time Capacity

In practice, the kernel matrix size is bounded above due to the filter's finite computational resources. We will assume that computation time is the limiting resource responsible for the bounded size $N \times N$. For the fast adaptive filter to operate online, the regularized kernel matrices need to be downsized and upsized within the time-span of a single iteration. The following propositions describe the computation time required to upsize and downsize. Refer to Appendix B and C for proofs.

Proposition 1 (Computation Time to Downsize)**.** *The computation time required to downsize the regularized kernel matrices is*

$$T_D(m) = m^2 + m + O(1) \tag{14}$$

where $m \times m$ is the kernel matrix size.

Proposition 2 (Computation Time to Upsize)**.** *The computation time required to upsize the regularized kernel matrices is*

$$T_U(m) = 5m^2 + 2mT_\kappa + 3m + O(1) \tag{15}$$

where $m \times m$ is the kernel matrix size and T_κ is the computation cost to calculate the kernel in (4)*.*

Corollary 1 (Computation Time to Downsize/Upsize)**.** *The computation time required for the downsizing/upsizing technique is*

$$T_{D/U}(m) = 6m^2 + 2mT_\kappa + 4m + O(1) \tag{16}$$

where $m \times m$ is the kernel matrix size and T_κ is the computation cost to calculate the kernel in (4)*.*

Corollary 1 implies that the fast adapting filter needs to process at least $T_{D/U}(N)$ operations online. More specifically, this capacity refers to the time allotted for the algorithm to downsize and upsize the regularized kernel matrices. If the kernel matrix size is smaller than $N \times N$ at any given time, the downsizing/upsizing technique will not fully utilize the computation time allocated by the filter. It is this "residual" computation time that gives the algorithm flexibility to resize the kernel matrix online.

3.2. The Resizing Range

Consider a kernel matrix of size $m \times m$, where $1 < m < N$. Prior to upsizing, we have the option to bypass downsizing the regularized kernel matrices. This sequence will cause the kernel matrix size to grow by one. We may also be able to recursively upsize r more times, where data inputs would be recalled from the r most recent iterations not being used to calculate the current kernel matrix. From (15) and (16), the upsizing range $\mathcal{R}_m^U$ of computationally obtainable kernel matrix sizes given m is

$$\mathcal{R}_m^U = \left\{ \bar{m} \leq N : \sum_{i=m}^{\bar{m}} T_U(i) \leq T_{D/U}(N) \right\} \tag{17}$$

There is also the option to recursively downsize the regularized kernel matrices. However, enough computation time must be reserved to upsize with the most recent data input. From (14),(15), and (16), the downsizing range $\mathcal{R}_m^D$ of computationally obtainable kernel matrix sizes given m is

$$\mathcal{R}_m^D = \left\{ \bar{m} \geq 1 : \sum_{i=\bar{m}}^{m} T_D(i) + T_U(\bar{m}) \leq T_{D/U}(N) \right\} \tag{18}$$

Combining (17) and (18), the resizing range $\mathcal{R}_m$ for any kernel matrix of size $m \times m$ is

$$\mathcal{R}_m = \mathcal{R}_m^U \cup \mathcal{R}_m^D \cup \mathcal{R}_1^U \tag{19}$$

where the union with $\mathcal{R}_1^U$ represents the option of discarding the regularized kernel matrices prior to upsizing, an operation that takes $O(1)$ time.

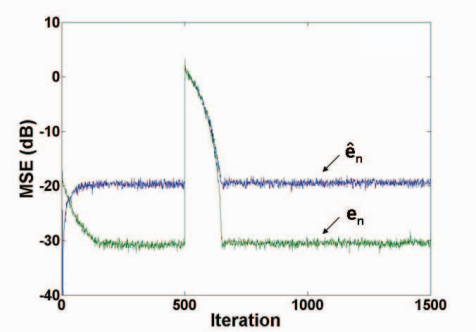

Fig. 1. The original sliding-window kernel RLS algorithm adaptively filters a nonlinear Wiener system. The linear channel is initialized with H_1 and then is abruptly changed to H_2 at $n = 500$. The overall performance e_n and system behavior $\hat{e}_n$ parameters are shown for a kernel matrix of fixed size 150×150.

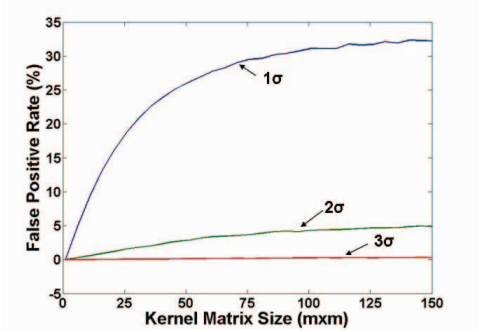

Fig. 2. False positive rates for threshold detection are dependent on the kernel matrix size. Error percentages are averaged over 10,000 simulations of the time-invariant nonlinear Wiener system.

4. APPLICATIONS TO THE NONLINEAR WIENER SYSTEM

4.1. System Identification Problem

The following summarizes the nonlinear Wiener system described by Van Vaerenbergh et al [2, 3]. A binary signal $x_n \in \{+1, -1\}$ is sent through a communication channel composed of a linear filter $H(z)$ of length L convolved with a nonlinear transformation $f(u)$. The channel is located within a black box such that the noisefree output signal v_n is inaccessible. Instead, a summation of v_n and additive white Gaussian noise (AWGN) is outputted as a noisy signal y_n.

The sliding-window kernel RLS algorithm acts as a fast adaptive filter for the nonlinear Wiener system. Given x_n and y_n, the filter predicts v_n based on the function coefficients learned over the last m iterations, where $m \times m$ is the current kernel matrix size. Overall performance e_n is measured in mean squared error (MSE) between the predicted and noiseless signals $\hat{v}_n$ and v_n averaged over 250 Monte-Carlo simulations.

For all simulations the following parameters are used: AWGN level is SNR $= 20dB$; channel length is $L = 4$; regularization constant is $\lambda = 1$; memoryless nonlinearity is $f(u) = \tanh(u)$; polynomial kernel is $\kappa(\vec{x}_i, \vec{x}_j) = (1 + \vec{x}_i'\vec{x}_j)^3$; linear channels are

$$H_1(z) = 1 - 0.3668z^{-1} - 0.4764z^{-2} + 0.8070z^{-3} \quad (20)$$

$$H_2(z) = 1 - 0.8326z^{-1} + 0.6656z^{-2} + 0.7153z^{-3} \quad (21)$$

4.2. Estimating System Behavior

Figure 1 shows the overall performance e_n for a fixed size kernel matrix. In addition, the MSE between the predicted and noisy signals $\hat{v}_n$ and y_n is plotted. This parameter, labeled $\hat{e}_n$, describes how different the filter's and system's output signals are. Unlike e_n, the components of $\hat{e}_n$, labeled $|\hat{v}_n - y_n|$, can be calculated online by the fast adaptive filter.

During filter initialization when only $n < N$ data inputs exist, the kernel matrix is relatively small, so the predicted signal $\hat{v}_n$ does not stray far from the noisy signal y_n. As the size of the kernel matrix increases, the function coefficients in (5) are further trained and assumed to minimize $|\hat{v}_n - v_n|$. Thus, the metric $|\hat{v}_n - y_n|$ becomes an increasingly better estimation of the noise signal of a time-invariant system.

If the system's noise level is bounded, we can use $|\hat{v}_n - y_n|$ to detect system changes. For example, the abrupt linear channel change in Figure 1 causes $\hat{e}_n$ to spike above its steady state level at the SNR of $20dB$. By setting a threshold, we can declare a system change when $|\hat{v}_n - y_n|$ exceeds this value. However, we would like to know the false positive rate for a given threshold.

Let σ be the standard deviation of a system's AWGN. For the given thresholds, Figure 2 shows the relationship between the kernel matrix size and the false positive rate of a time-invariant system. For small kernel matrix sizes, the filter has lower false positive rates, a direct result from their tendency to predict a signal $\hat{v}_n$ similar to the noisy signal y_n. As m increases, the confidence levels of the $1\sigma, 2\sigma,$ and 3σ thresholds tend to the Gaussian empirical confidence levels of $68, 95,$ and 99.7 percent.

4.3. A Simple Technique Using the Resizing Range

We propose a simple resizing technique to demonstrate the potential performance increase of using the resizing range. When a given a threshold is exceeded by $|\hat{v}_n - y_n|$, the modified algorithm resizes the $m \times m$ kernel matrix to $\max(\bar{m} \in \mathcal{R}_1(m))$, where

$$\mathcal{R}_1(m) = \left\{ \bar{m} \leq N : \sum_{i=1}^{\bar{m}} T_U(i) \leq T_{D/U}(m) \right\} \quad (22)$$

In other words, a resizing range subset is created assuming the current value $m \times m$ is the maximum kernel matrix size for the filter.

To gather intuition on how this technique works, consider both a true and false positive occurrence for a kernel matrix of $m = N$. Both occurrences resize the kernel matrix to $\max(\bar{m} \in \mathcal{R}_1)$. For the true positive occurrence, the modified algorithm will iteratively decrease the kernel matrix to $\max(\bar{m} \in \mathcal{R}_1(m))$ until the threshold is no longer triggered. This allows the kernel matrix size to remain around some $m \times m$ that achieves acceptable system behavior with respect to $|\hat{v}_n - y_n|$. For the false positive occurrence, the modified algorithm of $m < N$ is now less likely to trigger another false positive (see Figure 2). This property allows the kernel matrix size to be iteratively increased back to its optimal size of $m = N$.

Figure 3 shows the original and modified algorithms adaptively filtering a nonlinear Wiener system whose linear channel undergoes an abrupt change. As expected, the resizing technique allows the filter to adapt much faster to the H_2 linear channel by essentially discarding the data inputs prior to the change. We also see the effect

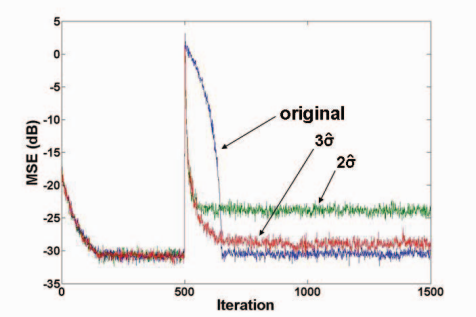

Fig. 3. A modified algorithm using the simple resizing technique adaptively filters the same system described in Figure 1. An estimated system noise $\hat{\sigma}$ is calculated from $|\hat{v}_n - y_n|$ averaged over the iterations $150 < n < 500$. Thresholds of $2\hat{\sigma}$ and $3\hat{\sigma}$ are implemented beginning at $n = 500$. The overall performance parameter e_n is shown for the kernel matrices of $N = 150$.

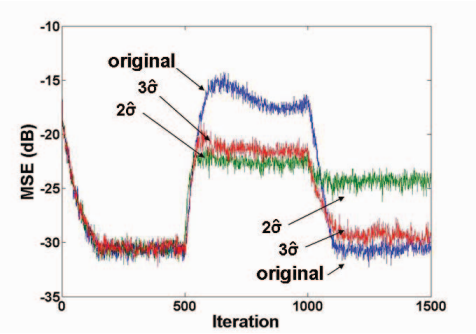

Fig. 4. The modified algorithm described in Figure 3 adaptively filters a nonlinear Wiener system whose linear channel varies linearly from H_1 at $n = 500$ to H_2 at $n = 1000$.

that the threshold has on steady state filter performance; low thresholds, which correspond to higher false positive rates, yield worse performance when the system is time-invariant.

Figure 4 shows a more gradual linear channel transition from H_1 to H_2. When the system is linearly varying, the modified algorithms achieve better performance than the original. Here the resizing technique retains a fraction of the data inputs to construct the kernel matrix; high thresholds result in larger fractions on average.

5. CONCLUSION

The resizing methodology gives the sliding-window kernel RLS algorithm additional flexibility for fast adaptive nonlinear filtering applications. Through simple modifications to the original algorithm, we showed increased performance for time-varying systems.

We also examined the system behavior metric $|\hat{v}_n - y_n|$ and its application of detecting system changes. Furthermore, confidence levels in select thresholds for this metric are shown to depend on the system noise and kernel matrix size.

Our goal is to provide the groundwork for more elaborate algorithms to adaptively modify the kernel matrix size within the resizing range. We conclude that the techniques presented are applicable in practice since they preserve the original algorithm's $O(N^2)$ time.

6. REFERENCES

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

[2] S. Van Vaerenbergh, J. Via, and I. Santamana, "A sliding-window kernel rls algorithm and its application to nonlinear channel identification," *Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings. 2006 IEEE International Conference on*, vol. 5, May 2006.

[3] S. Van Vaerenbergh, J. Via, and I. Santamana, "Nonlinear system identification using a new sliding-window kernel rls algorithm," *Journal of Communications*, vol. 2, no. 3, pp. 1–8, May 2007.

[4] V. N. Vapnik, *The Nature of Statistical Learning Theory*, Springer-Verlag New York, Inc., New York, USA, 1995.

[5] N. Aronszajn, "Theory of reproducing kernels," *Transactions of the American Mathematical Society*, vol. 68, pp. 337–404, 1950.

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

A. INVERSE REGULARIZED KERNEL MATRIX

Definitions for downsizing calculations

$$\mathbf{D} = \begin{bmatrix} \mathbf{G}_n^{-1}(2,2) & \cdots & \mathbf{G}_n^{-1}(2,N) \\ \vdots & \ddots & \vdots \\ \mathbf{G}_n^{-1}(N,2) & \cdots & \mathbf{G}_n^{-1}(N,N) \end{bmatrix} \tag{23}$$

$$\vec{f} = \left(\mathbf{G}_n^{-1}(2,1), \ldots, \mathbf{G}_n^{-1}(N,1)\right)^T \tag{24}$$

Definitions for upsizing calculations

$$\vec{b} = (\mathrm{G}_n(1,N), \ldots, \mathrm{G}_n(N-1,N))^T \tag{25}$$

$$g = (\mathrm{G}_{n+1}(N,N) - \vec{b}^T \check{\mathbf{G}}_n^{-1} \vec{b})^{-1} \tag{26}$$

B. PROOF OF PROPOSITION 1

Proof. From (9), the computation time required to downsize $\mathbf{G}_n$ to $\check{\mathbf{G}}_n$ is constant time for all m. From (12), the computation time required to downsize $\mathbf{G}_n^{-1}$ to $\check{\mathbf{G}}_n^{-1}$ is $m^2 + m + O(1)$. Summing these results we have

$$T_D(m) = m^2 + m + O(1) \tag{27}$$

□

C. PROOF OF PROPOSITION 2

Proof. From (10), the computation time required to upsize $\check{\mathbf{G}}_n$ to $\mathbf{G}_{n+1}$ is $2mT_\kappa + O(1)$ where T_κ is the computation cost to calculate the kernel in (4). From (13), the computation time required to upsize $\check{\mathbf{G}}_n^{-1}$ to $\mathbf{G}_{n+1}^{-1}$ is $T_g(m) + 4m^2 + 2m + O(1)$ where $T_g(m) = m^2 + m + O(1)$ is calculated from (26). Summing these results we have

$$T_U(m) = 5m^2 + 2mT_\kappa + 3m + O(1) \tag{28}$$

□