SLIDING WINDOW GREEDY RLS FOR SPARSE FILTERS

Alexandru Onose, Bogdan Dumitrescu, Ioan Tăbus

Department of Signal Processing
Tampere University of Technology
PO BOX 553, 33101 Tampere, Finland
e-mail: alexandru.onose@tut.fi, bogdan.dumitrescu@tut.fi, ioan.tabus@tut.fi

ABSTRACT
We present a sliding window RLS for sparse filters, based on the greedy least squares algorithm. The algorithm adapts a partial QR factorization with pivoting, using a simplified search of the filter support that relies on a neighbor permutation technique. For relatively small window size, the proposed algorithm has a lower complexity than recent exponential window RLS algorithms. Time-varying FIR channel identification simulations show that the proposed algorithm can also give better mean squared coefficient errors.

Index Terms— adaptive algorithm, recursive least squares, sliding window, sparse filters

1. INTRODUCTION
The recursive least squares (RLS) algorithm is most often implemented using an exponential window (EW). Although sliding window (SW) algorithms exist [1, 2], they offer relatively few benefits, while having a higher complexity; hence, they are much less used than EW RLS. The recent interest in RLS algorithms for sparse filters [3, 4, 5] was focused on EW RLS. Our aim in this paper is to show that, opposite to the full filters case, a sliding window RLS can have lower complexity than EW RLS.

At each time \( t \in \mathbb{N} \), the RLS algorithm provides a least-squares solution \( x_t \in \mathbb{R}^N \) to the overdetermined linear system \( A_t x_t \approx b_t \), taking into account the new equation \( a_t^T x_t \approx b_t \) together with previous equations of the same form. In a time-varying context, more weight is given to recent equations, while the past needs to be forgotten. The SW RLS criterion for a window of length \( L \)

\[
J(t) = \sum_{\tau=0}^{L-1} \lambda^\tau |e(t - \tau)|^2, \tag{1}
\]

is the approximation error for the equation appeared at time \( t \). So, only the most recent \( L \) equations are considered for computing \( x_t \). For example, in an FIR channel identification problem, the error (2) has the form

\[
e(t) = y(t) - \sum_{i=0}^{N-1} h_i u(t - i), \tag{3}
\]

where \( u(t) \) is the input, \( y(t) \) is the output and \( h \in \mathbb{R}^N \) is the vector of filter coefficients; the correspondence with (2) is immediate.

We consider the SW RLS problem for sparse filters: we assume that the solution \( x_t \) has at most \( M \) nonzero elements, \( M \) being given; the typical case is \( M \ll N \). For full filters, it is necessary to take \( L > N \) in order to obtain a least-squares solution. However, for sparse filters, the required condition is \( L > M \), which leaves open the possibility to take \( L < N \). This is the key to low complexity and good performance is obtained for sufficiently fast time-varying channels, as we will show later.

For solving the sparse SW RLS problem, we adapt the greedy least-squares (GLS) algorithm [6] (known also, in a modified form, as optimized orthogonal matching pursuit [7]) to the QR factorization update algorithm from [1]. In particular, for the greedy selection of the nonzero elements of the solution, we employ the neighbor-restricted search technique introduced in [5].

2. SLIDING WINDOW GREEDY RLS
Using a sliding window, the least-squares approximation problem corresponding to the criterion (1) is to minimize \( \|b_t - A_t x_t\|_2 \), with \( A_t \in \mathbb{R}^{L \times N} \), under the constraint that \( x_t \) has \( M \) nonzero elements. (For simplicity, we assume that the forgetting factors are included in \( A_t \) and \( b_t \).) The algorithm we propose maintains the partial QR factorization with pivoting

\[
A_t P_t = Q_t R_t, \tag{4}
\]

where \( Q_t \in \mathbb{R}^{L \times L} \) is an orthogonal matrix, \( R_t \in \mathbb{R}^{L \times N} \) is upper triangular in its first \( M \) columns and \( P_t \) is a permu-
tation matrix responsible for bringing into the first \( M \) positions the columns corresponding to the nonzero elements of \( x_t \) (called active columns). These elements are computed by solving the triangular system with the matrix \( \mathbf{R}_t(1 : M, 1 : M) \) and the right hand side made by the first \( M \) elements of \( \mathbf{Q}_t^T \mathbf{b}_t \).

At time \( t \), the algorithm has two main steps: downdating and updating [1]. Downdating has the purpose of eliminating the oldest equation from the previous window, i.e., starting from (4) at time \( t - 1 \), to compute

\[
\mathbf{A}_{t-1} \mathbf{P}_{t-1} = \begin{bmatrix}
1 & 0 \\
0 & \hat{\mathbf{Q}}_{t-1}^{-1}
\end{bmatrix}
\begin{bmatrix}
\mathbf{a}_{t-L}^T \\
\hat{\mathbf{R}}_{t-1}
\end{bmatrix},
\]

(5)

where \( \mathbf{a}_{t-L}^T = \mathbf{a}_{t-L}^T \mathbf{P}_{t-1}^{-1} \); hence, the first row of (5) reads

\[
\hat{\mathbf{a}}_{t-L}^T = \hat{\mathbf{a}}_{t-L}^T \mathbf{P}_{t-1}^{-1}
\]

and can be eliminated. The matrix \( \hat{\mathbf{R}}_{t-1} \in \mathbb{R}^{(L-1) \times N} \) is upper triangular in its first \( M \) columns, hence the last \( L - 1 \) rows of \( \mathbf{A}_{t-1} \mathbf{P}_{t-1} \) have the QR factorization \( \hat{\mathbf{Q}}_{t-1} \hat{\mathbf{R}}_{t-1} \).

Updating starts from adding the (permuted) current equation to the above factorization

\[
\mathbf{A}_{t} \mathbf{P}_{t} = \begin{bmatrix}
\hat{\mathbf{Q}}_{t-1} & 0 \\
0 & 1
\end{bmatrix}
\begin{bmatrix}
\hat{\mathbf{R}}_{t-1} \\
\hat{\mathbf{a}}_{t}^T
\end{bmatrix},
\]

(6)

where \( \mathbf{a}_{t}^T = \mathbf{a}_{t}^T \mathbf{P}_{t-1}^{-1} \), and obtains (4). It is in the updating stage that active columns are (re)selected. Following the strategy introduced in [5], we allow at most one inactive column to enter (and hence another column to leave) the active set at time \( t \).

All downdating and updating operations use orthogonal transformations and hence are numerically reliable. We give below details on the operations performed in the two steps. The matrix variables in the algorithm are \( \mathbf{R} \in \mathbb{R}^{L \times N} \) for storing \( \mathbf{R}_t \) and \( \mathbf{U} \in \mathbb{R}^{L \times L} \) for storing \( \mathbf{Q}_t^T \) (we work on the transposed for applying all transformations from the left); the column permutations will be described informally. The variable \( \mathbf{b} \) will store \( \mathbf{Q}_t^T \mathbf{b}_t \).

2.1. Downdating

Downdating consists of computing the elementary orthogonal transformations that bring \( \mathbf{Q}_{t-1} \) to the form from (5), i.e., force zeros in its first row (and column), in an order chosen to damage the least the triangular form of \( \mathbf{R}_{t-1} \). The initial and final form of the matrices \( \mathbf{U} \) and \( \mathbf{R} \) are shown in Figure 1, where \( M = 3, L = 6 \) and only the first 5 of the \( N \) columns of \( \mathbf{R} \) are depicted. The downdating process is similar with that from [1], excepting that here a Householder reflector can be used for zeroing the last \( L - M + 1 \) elements of the first column of \( \mathbf{U} \), instead of several Givens rotations. The main operations are the following.

1. Compute reflector \( \mathbf{H} \) that zeroes \( \mathbf{U}(M + 2 : L, 1) \) and apply it: \( \mathbf{U} \leftarrow \mathbf{H} \mathbf{U}, \mathbf{R} \leftarrow \mathbf{H} \mathbf{R}, \mathbf{b} \leftarrow \mathbf{H} \mathbf{b} \).
2. For \( k = M : -1 : 1 \)

\[
\begin{array}{c|c|c}
\mathbf{U} & \mathbf{R} & 1 \ 0 \ 0 \ 0 \ 0 \ 0 \\
\mathbf{U} & \mathbf{R} & \mathbf{x} \ \mathbf{x} \ \mathbf{x} \ \mathbf{x} \ \mathbf{x} \ \mathbf{x}
\end{array}
\]

Fig. 1. \( \mathbf{U} \) and \( \mathbf{R} \) before and after downdating.

- \( \mathbf{x} \) denotes a zero column.

2.1. Compute rotation \( \mathbf{G} \) that zeroes \( \mathbf{U}(k+1, 1) \), based on \( \mathbf{U}(k, 1) \), and apply it: \( \mathbf{U} \leftarrow \mathbf{G} \mathbf{U}, \mathbf{R} \leftarrow \mathbf{G} \mathbf{R}, \mathbf{b} \leftarrow \mathbf{G} \mathbf{b} \).

Of course, the above operations are performed efficiently:

- in step 1, only rows \( M + 1 : L \) of the matrices are modified;
- in step 2.1, only rows \( k \) and \( k+1 \) are affected. This operation order ensures that at the end \( \mathbf{R} \) is upper Hessenberg in the first \( M \) columns and, after eliminating its first row, it becomes upper triangular in the first \( M \) columns.

2.2. Updating

Greedy LS belongs to the family of matching pursuit algorithms and finds the nonzero (active) positions of \( x_t \) (or the corresponding columns of \( \mathbf{A}_t \)) one by one. Each position is chosen such that, when adding it to the active set, the LS residual corresponding to the solution obtained using the current active positions is minimized. In the original batch algorithm, all positions are allowed to compete for the active set. In a recursive context, it is unlikely that radical changes of the active set are possible to detect immediately. So, it is enough to allow gradual changes. We have shown in [5] that, with an exponential window, the following strategy for changing the active set is successful in tracking a time-varying channel:

- For position \( k \leq M - 1 \), choose only between columns \( k \) and \( k+1 \) of the current active set, process called neighbor permutation.
- For position \( M \), choose between all remaining columns.

So, at most one new column enters the active set at time \( t \). We adopt the same strategy for the SW algorithm and give next a formal description of the update step, then explain it in more detail. Remind that the update starts with \( \mathbf{R} \) as in (6), i.e., upper triangular in the first \( M \) columns, with the exception of the last row, as in the leftmost diagram from Figure 2.

A. Neighbor permutation

1. For \( k = 1 : M - 1 \)

1.1. Find the index of the best column in \( S = \{ k, k+1 \} \):

\[
\tilde{k} = \arg \max_{i \in S} \frac{\mathbf{R}(k : L, i)^T \cdot \mathbf{b}(k : L)}{\| \mathbf{R}(k : L, i) \|_2^2}
\]

(7)

1.2. If \( \tilde{k} = k + 1 \) (permutation is needed)

1.2.1. Swap columns \( k \) and \( k+1 \) of \( \mathbf{R} \)
Table 1, we 1000 runs with different channel filters at each run, all with the same variation speed \( f_d T_s \). In Table 1, we

Fig. 3. Evolution of \( R \) after selection of last active column.

The input \( u(t) \) is Gaussian from \( \mathcal{N}(0, 1) \) and the output \( y(t) \) is affected by additive white noise with variance \( \sigma^2 = 0.01 \). Denoting \( \hat{h}(t) \in \mathbb{R}^{N+1} \) the true vector of coefficients and \( \hat{h}(t) \) the estimated one at time \( t \) we have measured the mean squared coefficient error

\[
\text{MSE}(t) = \frac{E[\| \hat{h}(t) - h(t) \|_2^2]}{E[\| h(t) \|_2^2]} 
\]

by averaging 1000 runs with different channel filters at each run, all with the same variation speed \( f_d T_s \). In Table 1, we

2.3. Complexity issues

The active set needs not be changed at each moment, but only when \( t \) is a multiple of a small integer \( \tau_0 \). When the active set is not changed, the updating algorithm becomes much simpler; column permutations are not needed and the triangular structure is restored plainly with Givens rotations.

We assume that \( M \ll N \), i.e. the solution is indeed sparse. With a careful implementation, the average number of operations (multiplications or additions) required by downdating and updating is

\[
\nu \approx \left( 4 + \frac{4}{\tau_0} \right) L(N + L). 
\]
Table 1. MSE for the studied algorithms.

<table>
<thead>
<tr>
<th>( f_d T_s )</th>
<th>MSE ((L, \lambda))</th>
<th>MSE ((\lambda))</th>
<th>MSE ((\lambda))</th>
<th>MSE ((L, \lambda))</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0002</td>
<td>0.00352</td>
<td>0.00333</td>
<td>0.051</td>
<td>0.00182</td>
</tr>
<tr>
<td></td>
<td>(55, 0.98)</td>
<td>(0.96)</td>
<td>(0.98)</td>
<td>(55, 0.98)</td>
</tr>
<tr>
<td>0.0005</td>
<td>0.00606</td>
<td>0.00621</td>
<td>0.176</td>
<td>0.00328</td>
</tr>
<tr>
<td></td>
<td>(45, 0.96)</td>
<td>(0.94)</td>
<td>(0.96)</td>
<td>(45, 0.96)</td>
</tr>
<tr>
<td>0.001</td>
<td>0.01202</td>
<td>0.01233</td>
<td>0.568</td>
<td>0.00599</td>
</tr>
<tr>
<td></td>
<td>(40, 0.92)</td>
<td>(0.92)</td>
<td>(0.96)</td>
<td>(40, 0.92)</td>
</tr>
<tr>
<td>0.002</td>
<td>0.02684</td>
<td>0.02883</td>
<td>1.333</td>
<td>0.01363</td>
</tr>
<tr>
<td></td>
<td>(40, 0.90)</td>
<td>(0.90)</td>
<td>(0.98)</td>
<td>(40, 0.90)</td>
</tr>
<tr>
<td>0.005</td>
<td>0.11973</td>
<td>0.13187</td>
<td>1.391</td>
<td>0.05128</td>
</tr>
<tr>
<td></td>
<td>(25, 0.90)</td>
<td>(0.86)</td>
<td>(0.995)</td>
<td>(25, 0.90)</td>
</tr>
</tbody>
</table>

It can be noticed from Table 1 that, as the channel variation speed increases, the best window length \( L \) decreases and also SW-GRLS becomes increasingly better compared to its EW counterpart. Since \( L \) is about or less than \( N/4 \), the complexity of SW-GRLS is lower than that of EW-GRLS. While SW-GRLS largely (and expectedly) outperforms the full RLS algorithm (which is unable to give meaningful results for higher variation speeds), it gives a MSE only about twice larger than the sparsity informed RLS, which is the best attainable by an RLS algorithm. If \( M > M' \), the performance degrades, but is still acceptable if \( M' \) is only slightly larger than \( M' \). For example, for \( M = 10 \), MSE is about twice larger than the values from Table 1.

Comparisons with SPARLS [4] appear to be favorable, but not included here since SPARLS adapts also to the true number of coefficients \( M' \), while our algorithm does not yet adapt.

4. CONCLUSIONS

We have presented a sliding window RLS that uses a greedy approach for finding the nonzero coefficients of the sparse solution. The algorithm is based on orthogonal operations and hence is numerically stable. Theoretical and simulation study show that, for small window length \((L < N/4)\) and for sufficiently fast time-varying channels, the algorithm is less complex and gives better coefficient errors than exponential window greedy RLS. So, for sparse filters, the sliding window can give benefits in terms of both complexity and estimation quality, a property that does not hold for nonsparse filters.

5. REFERENCES


