Boosted LMS-based Piecewise Linear Adaptive Filters

Dariush Kari and Iman Marivani
Department of Electrical and Electronics Engineering
Bilkent University, Ankara, Turkey
{kari, marivani}@ee.bilkent.edu.tr

Ibrahim Delibalta
Turk Telekom Communications Services Inc., Istanbul, Turkey
ibrahim.delibalta@turktelekom.com.tr

Suleyman Serdar Kozat
Department of Electrical and Electronics Engineering
Bilkent University, Ankara, Turkey
kozat@ee.bilkent.edu.tr

***Abstract*—We introduce the boosting notion extensively used in different machine learning applications to adaptive signal processing literature and implement several different adaptive filtering algorithms. In this framework, we have several adaptive constituent filters that run in parallel. For each newly received input vector and observation pair, each filter adapts itself based on the performance of the other adaptive filters in the mixture on this current data pair. These relative updates provide the boosting effect such that the filters in the mixture learn a different attribute of the data providing diversity. The outputs of these constituent filters are then combined using adaptive mixture approaches. We provide the computational complexity bounds for the boosted adaptive filters. The introduced methods demonstrate improvement in the performances of conventional adaptive filtering algorithms due to the boosting effect.**

I. Introduction

Boosting is considered as one of the most important ensemble learning methods in the machine learning literature [1]–[3]. As an ensemble learning method [4], boosting combines several parallel running "weakly" performing algorithms to build a final "strongly" performing algorithm, by finding a linear combination of weak learning algorithms. However, significantly less attention is given to the idea of boosting in the adaptive signal processing literature. To this end, our goal is (a) to use the boosting notion in adaptive filtering, (b) derive several different adaptive filtering algorithms based on the boosting approach (c) and demonstrate the intrinsic connections of boosting with the adaptive mixture methods [5] and data reuse algorithms [6] widely studied in the adaptive signal processing literature.

Although boosting is initially introduced in the batch setting [2], i.e., where algorithms boost themselves over a fixed set of training data, it is later extended to the online setting [7]. In the online setting, we neither need nor have a fixed set of training data, however, the data arrives one by one as a stream. Each newly arriving data is processed and then discarded without any storing. The online setting is naturally motivated by many real life applications especially for the ones involving big data, where there is not enough storage space available or the constraints of the problem require instant processing [8]. However, for our purposes, the online setting is especially important since it is directly akin to adaptive filtering framework where the streaming or sequentially arriving data is used to adapt the internal parameters of the filter, either to adaptively learn the underlying model or to track the nonstationary data statistics [9].

Specifically, we have m parallel running adaptive filters that receive the input vectors sequentially one by one. Each adaptive algorithm can use a different update, such as the recursive least squares (RLS) update or least-mean squares (LMS) update. After receiving the input vector, each algorithm produces its output and then calculates its instantaneous error after the observation is revealed. These updates are performed for all the m constituent filters in the mixture. However, in the online boosting approaches, these adaptations at each time proceed in rounds from top to bottom, starting from the first adaptive filter to the last one to achieve the "boosting" effect [10]. Furthermore, unlike the usual mixture approaches [5], the update of each adaptive filter depends on the previous adaptive filters in the mixture. Based on the performance of the filters from 1 to k on the current $(\mathbf{x}_t, d_t)$ pair, the $(k+1)$th filter may give more or less emphasize to $(\mathbf{x}_t, d_t)$ pair in its adaptation in order to rectify the mistake of the previous adaptive filters.

This idea is clearly related to the adaptive mixture algorithms widely used in the signal processing literature. However, unlike the mixture methods, the updates of the constituent filters are not independent in boosting methods.

We implement our boosting algorithms on piecewise linear filters, since such filters deliver a significantly superior performance than linear filters, with a comparable complexity [11]. To this end, we apply the boosting notion to several parallel running piecewise linear LMS-based filters, and introduce three different approaches to use the importance weights [10]. In the first approach, weighted updates, we use the importance weights directly to produce certain weighted LMS algorithms. In the second approach, data reuse, we use the importance weights to construct data reuse adaptive algorithms. The third approach, random updates, uses the importance weights to decide whether to update the constituent filters, based on a random number generated from a Bernoulli distribution with the parameter equal to the weight. The random updates method can be effectively used for big data processing [12], due to the reduced complexity. The output of the constituent filters is also combined using a linear filter to construct the final output of the algorithm. The final combination filter is also updated

978-0-9928-6265-7/16/$31.00 ©2016 IEEE

using the LMS algorithm [5].

II. Problem Description and Background

All vectors are column vectors and represented with bold lower case letters. Matrices are represented by bold upper case letters. For a vector $\mathbf{a}$ (or a matrix $\mathbf{A}$), $\mathbf{a}^T$ (or $\mathbf{A}^T$) is the transpose and Tr($\mathbf{A}$) is the trace of the matrix $\mathbf{A}$. The time index is given in the subscript, i.e., x_t is the sample at time t. We work with real data for notational simplicity. We denote the mean of a random variable x as $E[x]$.

We sequentially receive r-dimensional input (regressor) vectors $\{\mathbf{x}_t\}_{t\geq 1}$, $\mathbf{x}_t \in \mathbb{R}^r$, and desired data $\{d_t\}_{t\geq 1}$, and estimate d_t by

$$\hat{d}_t = f_t(\mathbf{x}_t), \tag{1}$$

in which, $f_t(.)$ is an adaptive filter. At each time t the estimation error is given by $e_t = d_t - \hat{d}_t$, and is used to update the parameters of the adaptive filter. For presentation purposes, we assume that $d_t \in [-1, 1]$, however, our derivations hold for any bounded but arbitrary desired data sequences. For example, in the prediction problem $d_t = x_{t+1}$ and in the channel equalization application $\{d_t\}$ are the transmitted bits, where $\mathbf{x}_t$ is the received data from the channel. In our framework, we do not use any statistical assumptions on the input vectors or on the desired data such that our results are guaranteed to hold in an individual sequence manner [13].

Note that although nonlinear filters can outperform linear filters, they usually undergo overfitting, stability, and convergence issues [11], [14]. Furthermore, nonlinear filters generally have higher computational complexities, which limits their use in most of the real-life applications [11], [14]. To overcome these problems, piecewise linear filters are proposed, which mitigate the overfitting and stability issues, while offering a comparable modeling performance to the nonlinear filters [11], [14]. Therefore, in this paper, we are particularly interested in piecewise linear filters, which serve as an elegant alternative to linear filters.

We use a piecewise linear adaptive filtering method, such that the desired signal is predicted as $\hat{d}_t = \sum_{i=1}^{N} s_{i,t}\mathbf{w}_{i,t}^T\mathbf{x}_t$, where $s_{i,t}$ is the indicator function of the ith region, i.e., $s_{i,t} = 1$ if $\mathbf{x}_t \in \mathcal{R}_i$, and $s_{i,t} = 0$ otherwise. Note that at each time t, only one of the $s_{i,t}$'s is nonzero, which indicates the region in which $\mathbf{x}_t$ lies. Thus, if $x_t \in \mathcal{R}_i$, we update only the ith linear filter. As an example, consider 2-dimensional input vectors $\mathbf{x}_t$, as depicted in Fig. 1. Here, we construct the piecewise linear filter f_t such that

$$\begin{aligned}\hat{d}_t = f_t(\mathbf{x}_t) &= s_{1,t}\mathbf{w}_{1,t}^T\mathbf{x}_t + s_{2,t}\mathbf{w}_{2,t}^T\mathbf{x}_t \\ &= s_t\mathbf{w}_{1,t}^T\mathbf{x}_t + (1-s_t)\mathbf{w}_{2,t}^T\mathbf{x}_t,\end{aligned} \tag{2}$$

Then, if $s_t = 1$ we shall update $\mathbf{w}_{1,t}$, otherwise we shall update $\mathbf{w}_{2,t}$, based on the amount of the error, e_t.

III. Boosted LMS Algorithms

As shown in Fig. 2, at each iteration t, we have m parallel running adaptive filters with estimating functions $f_t^{(k)}$, producing estimates $\hat{d}_t^{(k)} = f_t^{(k)}(\mathbf{x}_t)$ of d_t, $k = 1, \ldots, m$.

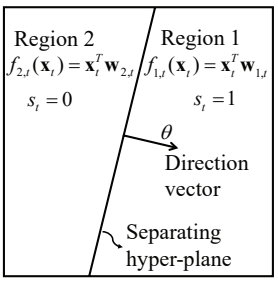

Fig. 1: A sample 2-region partition of the input vector (i.e., $\mathbf{x}_t$) space, which is 2-dimensional in this example. s_t determines whether $\mathbf{x}_t$ is in Region 1 or not.

As an example, if we use m "linear" filters, $\hat{d}_t^{(k)} = \mathbf{x}_t^T\mathbf{w}_t^{(k)}$ is the estimate generated by the kth constituent filter, and if we use piecewise linear filters (each of which with N different regions), $\hat{d}_t^{(k)} = \sum_{i=1}^{N} s_{i,t}\mathbf{x}_t^T\mathbf{w}_{i,t}$. The outputs of these m filters are then combined using the linear weights $\mathbf{z}_t$ to produce the final estimate as $\hat{d}_t = \mathbf{z}_t^T\mathbf{y}_t$ [5], where $\mathbf{y}_t \triangleq [\hat{d}_t^{(1)}, \ldots, \hat{d}_t^{(m)}]^T$ is the vector of outputs. After the desired signal d_t is revealed, the m parallel running filters will be updated for the next iteration. Moreover, the linear combination coefficients $\mathbf{z}_t$ are also updated using ordinary LMS method, as detailed later in Section III-D.

After d_t is revealed, the constituent filters, $f_t^{(k)}$, $k = 1, \ldots, m$, are consecutively updated as shown in Fig. 2 from top to bottom, i.e., first $k = 1$ is updated, then, $k = 2$ and finally $k = m$ is updated. However, to enhance the performance, we use a boosted updating approach [2], such that, the $(k+1)$th filter receives a "total loss" parameter, $l_t^{(k+1)}$, from the filter $f_t^{(k)}$.

$$l_t^{(k+1)} = l_t^{(k)} + \left[\sigma^2 - \left(d_t - f_t^{(k)}(\mathbf{x}_t)\right)^2\right], \tag{3}$$

to compute a weight $\lambda_t^{(k)}$. The total loss parameter $l_t^{(k)}$, indicates the sum of the differences between the desired Mean Squared Error (MSE), σ^2, and the squared error of the first $k-1$ filters at time t. Then, the difference $\sigma^2 - (e_t^{(k)})^2$ is added to $l_t^{(k)}$, to generate $l_t^{(k+1)}$, and $l_t^{(k+1)}$ is passed to the next constituent filter as shown in Fig. 2. Here, $\left[\sigma^2 - \left(d_t - f_t^{(k)}(\mathbf{x}_t)\right)^2\right]$ measures how much the kth constituent filter is off with respect to the final MSE performance goal. For example, if $d_t = f(\mathbf{x}_t) + \nu_t$ for some deterministic nonlinear function $f(\cdot)$ and ν_t is the observation noise, then σ^2 can be selected as an upper bound on the variance of the noise process ν_t. In this sense, $l_t^{(k)}$ measures how the constituent filters $j = 1, \ldots, k$ are cumulatively performing on $(d_t, \mathbf{x}_t)$ pair with respect to the final performance goal.

We then use the weight $\lambda_t^{(k)}$ to update the kth constituent filter with one of the methods "weighted updates", "data reuse", or "random updates", which will be explained later in the subsections of this section. Our aim is to make $\lambda_t^{(k)}$ large if the first $k-1$ constituent filters made large errors on d_t, so that the kth filter gives more importance to $(d_t, \mathbf{x}_t)$

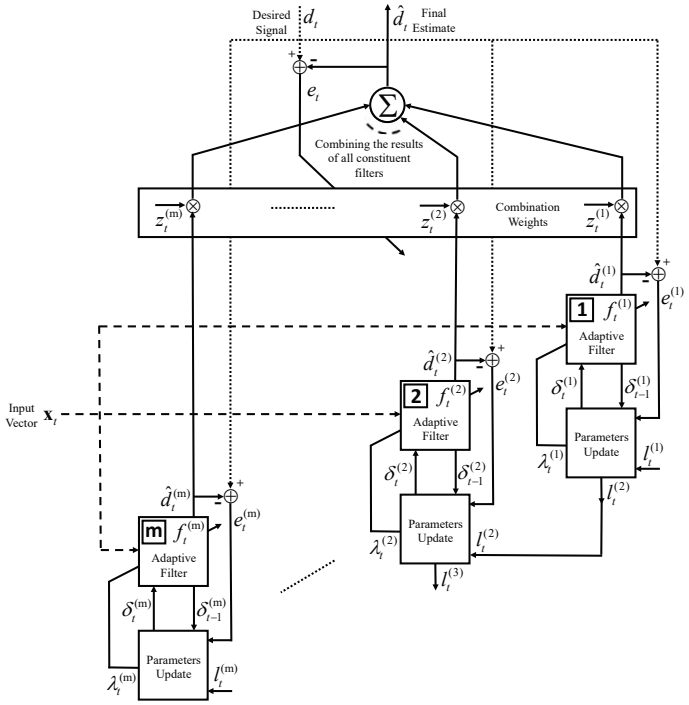

Fig. 2: The block diagram of a boosted adaptive filtering system that uses the input vector $\mathbf{x}_t$ to produce the final estimate $\hat{d}_t$. There are m constituent filters $f_t^{(1)}, \ldots, f_t^{(m)}$, each of which is an adaptive piecewise linear filter that generates its own estimate $\hat{d}_t^{(k)}$. The final estimate $\hat{d}_t$ is a linear combination of the estimates generated by all these constituent filters, with the combination weights $z_t^{(k)}$'s corresponding to $\hat{d}_t^{(k)}$'s. The combination weights are stored in a vector which is updated after each iteration t. At time t the kth filter is updated based on the values of $\lambda_t^{(k)}$ and $e_t^{(k)}$, and provides the $(k+1)$th filter with $l_t^{(k+1)}$ that is used to compute $\lambda_t^{(k+1)}$. The parameter $\delta_t^{(k)}$ indicates the average Mean Squared Error (MSE) of the kth filter over the first t estimations, and is used in computing $\lambda_t^{(k)}$.

in order to rectify the performance of the overall system. We now explain how to construct these weights, such that $0 < \lambda_t^{(k)} \leq 1$. To this end, we set $\lambda_t^{(1)} = 1$, for all t, and introduce a weighting similar to [10], [15]. We define the weights as $\lambda_t^{(k)} \triangleq \min\left\{1, \left(\delta_{t-1}^{(k)}\right)^{c\, l_t^{(k)}}\right\}$, where $\delta_{t-1}^{(k)}$ indicates an estimate of the kth filter's MSE, and $c \geq 0$ is a design parameter, which determines the "dependence" of each filter update on the performance of the previous filters, i.e., $c = 0$ corresponds to "independent" updates, like the ordinary combination of the filters [5], while a greater c indicates the greater effect of the previous filters performance on the weight $\lambda_t^{(k)}$ of the current filter. Here, $\delta_{t-1}^{(k)}$ is an estimate of the "Weighted Mean Squared Error" (WMSE) of the kth constituent filter over $\{\mathbf{x}_t\}_{t\geq 1}$ and $\{d_t\}_{t\geq 1}$. In the basic implementation of online boosting [10], [15], $\left(1 - \delta_{t-1}^{(k)}\right)$ is set to the classification advantage of the weak learners [15], where this advantage is assumed to be the same for all weak learners from $k = 1, \ldots, m$. In this paper, to avoid using any a priori knowledge and to be completely adaptive, we choose $\delta_{t-1}^{(k)}$ as the weighted and thresholded MSE of the kth filter up to time $t-1$ as

$$\delta_t^{(k)} = \frac{\sum_{\tau=1}^{t} \frac{\lambda_\tau^{(k)}}{4} \left(d_\tau - \left[f_\tau^{(k)}(\mathbf{x}_\tau)\right]^+\right)^2}{\sum_{\tau=1}^{t} \lambda_\tau^{(k)}}, \tag{4}$$

where $\left[f_\tau^{(k)}(\mathbf{x}_\tau)\right]^+$ thresholds $f_\tau^{(k)}(\mathbf{x}_\tau)$ into the range $[-1, 1]$. This thresholding is necessary to assure that $0 < \delta_t^{(k)} \leq 1$, which guarantees $0 < \lambda_t^{(k)} \leq 1$ for all $k = 1, \ldots, m$ and t. We point out that $\delta_t^{(k)}$ can be calculated recursively.

Regarding the definition of $\delta_t^{(k)}$ and $\lambda_t^{(k)}$, if the kth filter is "good", i.e., if $\delta_t^{(k)}$ is small enough, we will pass less weight to the next filters, such that those filters can concentrate more on the other samples. Hence, the filters can increase the diversity by concentrating on different parts of the data [5]. Furthermore, the weights $\lambda_t^{(k)}$'s are larger, i.e., close to 1, if most of the constituent filters, $j = 1, \ldots, k$, have errors larger than σ^2 on $(d_t, \mathbf{x}_t)$, and smaller, i.e., close to 0, if the pair $(d_t, \mathbf{x}_t)$ is easily modeled by the previous constituent filters such that the filters $k+1, \ldots, m$ do not need to concentrate more on this pair. Based on these weights, we next introduce three approaches to update the constituent filters, which are piecewise linear filters explained in Section II updated using LMS algorithm.

A. Directly Using λ's to Scale the Learning Rates

Since $0 < \lambda_t^{(k)} \leq 1$, these weights can be directly used to scale the learning rates for the LMS updates. When the kth filter receives the weight $\lambda_t^{(k)}$, it updates its filter coefficients $\mathbf{w}_{i,t}^{(k)}, i = 1, \ldots, N$, as

$$\mathbf{w}_{i,t+1}^{(k)} = \left(\mathbf{I} - \mu_i^{(k)} \lambda_t^{(k)} \mathbf{x}_t \mathbf{x}_t^T\right) \mathbf{w}_{i,t}^{(k)} + \mu_i^{(k)} \lambda_t^{(k)} \mathbf{x}_t d_t, \tag{5}$$

where $0 < \mu_i^{(k)} \lambda_t^{(k)} \leq \mu_i^{(k)}$. Note that we can choose $\mu_i^{(k)} = \mu_i$ for all k, since the adaptive algorithms work consecutively from top to bottom, and the ith linear filter of each different constituent filter will have a different learning rate $\mu_i \lambda_t^{(k)}$.

B. A Data Reuse Approach Based on the Weights

In this scenario, for updating $\mathbf{w}_{i,t}^{(k)}$, we use the LMS update $n_t^{(k)} = \text{ceil}(K \lambda_t^{(k)})$ times, where K is a fixed integer number, to obtain the $\mathbf{w}_{t+1}^{(k)}$ as

$$\begin{aligned} &\mathbf{q}^{(0)} = \mathbf{w}_{i,t}^{(k)}, \\ &\mathbf{q}^{(a)} = \left(\mathbf{I} - \mu_i^{(k)} \mathbf{x}_t \mathbf{x}_t^T\right) \mathbf{q}^{(a-1)} + \mu_i^{(k)} \mathbf{x}_t d_t, \; a = 1, \ldots, n_t^{(k)}, \\ &\mathbf{w}_{t+1}^{(k)} = \mathbf{q}^{\left(n_t^{(k)}\right)}. \end{aligned} \tag{6}$$

C. Random Updates Based on the Weights

In this scenario, we use the weight $\lambda_t^{(k)}$ to generate random number from a Bernoulli distribution, which equals 1 with probability $\lambda_t^{(k)}$, or equals zero with probability $1 - \lambda_t^{(k)}$. Then, if this number is 1, we do the ordinary LMS update on $\mathbf{w}_{i,t}^{(k)}$, otherwise we do not.

Algorithm 1 Boosted LMS with the proposed methods

Input: $(\mathbf{x}_t, d_t)$ (data stream), m (number of LMS piecewise linear constituent filters running in parallel) and σ^2 (the desired MSE, upper bound on the error variance).
Initialize the regression coefficients $\mathbf{w}_{i,1}^{(k)}$ for each LMS filter; and the combination coefficients as $\mathbf{z}_1 = \frac{1}{m}[1, 1, \ldots, 1]^T$; and for all k set $\delta_0^{(k)} = 0$.
for $t = 1$ **to** T **do**
  Receive the regressor data instance $\mathbf{x}_t$;
  Compute the indicator functions $s_{i,t}^{(k)}$ for all k's
  Compute the constituent filter outputs $\hat{d}_t^{(k)} = \sum_{i=1}^{N} s_{i,t}^{(k)} \mathbf{x}_t^T \mathbf{w}_{i,t}^{(k)}$;
  Produce the final estimate $\hat{d}_t = \mathbf{z}_t^T [\hat{d}_t^{(1)}, \ldots, \hat{d}_t^{(m)}]^T$;
  Receive the true output d_t (desired data);
  $\lambda_t^{(1)} = 1$; $l_t^{(1)} = 0$;
  for $k = 1$ **to** m **do**
    Update the regression coefficients $\mathbf{w}_{i,t}^{(k)}$ by using LMS and the weight $\lambda_t^{(k)}$ based on one of the introduced algorithms in Section III;
    $e_t^{(k)} = d_t - \hat{d}_t^{(k)}$;
    $\lambda_t^{(k)} = \min\left\{1, \left(\delta_{t-1}^{(k)}\right)^{c\, l_t^{(k)}}\right\}$;
    $\delta_t^{(k)} = \frac{\Lambda_{t-1}^{(k)} \delta_{t-1}^{(k)} + \frac{\lambda_t^{(k)}}{4}\left(d_t - \left[f_t^{(k)}(\mathbf{x}_t)\right]^+\right)^2}{\Lambda_{t-1}^{(k)} + \lambda_t^{(k)}}$;
    $\Lambda_t^{(k)} = \Lambda_{t-1}^{(k)} + \lambda_t^{(k)}$
    $l_t^{(k+1)} = l_t^{(k)} + \left[\sigma^2 - \left(e_t^{(k)}\right)^2\right]$;
  end for
  $\mathbf{z}_{t+1} = \left(\mathbf{I} - \mu \mathbf{y}_t \mathbf{y}_t^T\right) \mathbf{z}_t + \mu \mathbf{y}_t d_t$;
end for

D. The Final Algorithm

After the desired data d_t is revealed, we update the constituent filters as well as the combination weights $\mathbf{z}_t$. To update the combination weights, we again employ an LMS algorithm yielding

$$\mathbf{z}_{t+1} = \left(\mathbf{I} - \mu \mathbf{y}_t \mathbf{y}_t^T\right) \mathbf{z}_t + \mu \mathbf{y}_t d_t, \tag{7}$$

where $\mu > 0$ and $\mathbf{y}_t = [\hat{d}_t^{(1)}, \ldots, \hat{d}_t^{(m)}]^T$. The complete final algorithm is given in Algorithm 1.

IV. Complexity Analysis

In this section we compare the complexity of the proposed algorithms and find an upper bound for the weights $\lambda_t^{(k)}$. Suppose that the input vector has a length of r, i.e., $\mathbf{x}_t \in \mathbb{R}^r$. Each constituent filter performs $O(r)$ computations to generates its estimate, and requires $O(r)$ computations due to updating the linear filters using the LMS method (in their most basic implementations).

We derive the computational complexity of using the LMS updates in different boosting scenarios. Since there are a total of m constituent filters, all of which are updated in "weighted samples" method, this method has a computational cost of order $O(mr)$ per each iteration t. However, in "random updates", at iteration t, the kth filter will or will not be updated with probabilities $\lambda_t^{(k)}$ and $1 - \lambda_t^{(k)}$ respectively. Hence, if $E\left[\lambda_t^{(k)}\right]$ is upper bounded by $\tilde{\lambda}^{(k)} < 1$, the average computational complexity of the random updates method, will be $\sum_{k=1}^{m} O(\tilde{\lambda}^{(k)} r)$. In the Theorem, we provide sufficient constraints to have such an upper bound.

Furthermore, we can use such a bound for the "data reuse" mode as well. In this case, for each filter $f_t^{(k)}$, we perform the LMS update $\lambda_t^{(k)} K$ times, resulting a computational complexity of order $\sum_{k=1}^{m} K\, \tilde{\lambda}^{(k)} (O(r))$.

The following theorem determines the upper bound $\tilde{\lambda}^{(k)}$ for $E\left[\lambda_t^{(k)}\right]$.

Theorem: *If the adaptive filters converge and achieve a sufficiently small MSE (according to the proof following this Theorem), the following upper bound is obtained for $\lambda_t^{(k)}$, given that σ^2 is chosen properly,*

$$E\left[\lambda_t^{(k)}\right] \le \tilde{\lambda}^{(k)} = \left(\gamma^{-2\sigma^2}(1 + 2\zeta^2 \ln \gamma)\right)^{\frac{1-k}{2}}, \tag{8}$$

where $\gamma \triangleq E\left[\delta_{t-1}^{(k)}\right]$ and $\zeta^2 \triangleq E\left[\left(e_t^{(k)}\right)^2\right]$.

It can be straightforwardly shown that, this bound is less than 1 for appropriate choices of σ^2, and reasonable values for the MSE according to the proof. This theorem states that if we adjust σ^2 such that it is achievable, i.e., the adaptive filters can provide a slightly lower MSE than σ^2, the probability of updating the filters in the random updates scenario will decrease. This is of course our desired result, since if the filters are performing sufficiently well, there is no need for additional updates. Moreover, if σ^2 is opted such that the filters cannot achieve a MSE equal to σ^2, the filters have to be updated at each iteration, which increases the complexity.

Outline of the proof: For simplicity, in this proof, we have assumed that $c = 1$, however, the results are readily extended to the general values of c. Assume that $e_t^{(k)}$'s are independent and identically distributed (i.i.d) zero-mean Gaussian random variables with variance ζ^2. It can be shown that we achieve the stated upper bound in the Theorem, under the following necessary and sufficient conditions:

$$\frac{\left(\delta_{t-1}^{(k)}\right)^{2\sigma^2}}{\left(1 + 2\zeta^2 \ln\left(\delta_{t-1}^{(k)}\right)\right)^2} < \frac{\left(1 + 2\sigma^2\right)^2}{4(k+1)}, \tag{9}$$

and

$$\frac{(1-\xi_1)\sigma^2}{1 - 2\sigma^2 \ln\left(\delta_{t-1}^{(k)}\right)} < \zeta^2 < \frac{(1-\xi_2)\sigma^2}{1 - 2\sigma^2 \ln\left(\delta_{t-1}^{(k)}\right)}, \tag{10}$$

where

$$\xi_1 = \frac{\alpha^2(1 + 2\sigma^2) + \alpha\sqrt{(1 + 2\sigma^2)^2\alpha^2 - 4(k+1)(\delta_{t-1}^{(k)})^{2\sigma^2}}}{2(k+1)(\delta_{t-1}^{(k)})^{2\sigma^2}},$$

$$\xi_2 = \frac{\alpha^2(1+2\sigma^2) - \alpha\sqrt{(1+2\sigma^2)^2\alpha^2 - 4(k+1)(\delta_{t-1}^{(k)})^{2\sigma^2}}}{2(k+1)(\delta_{t-1}^{(k)})^{2\sigma^2}},$$

and

$$\alpha \triangleq 1 + 2\zeta^2 \ln\left(\delta_{t-1}^{(k)}\right).$$

V. Experiments

In this section, we demonstrate the efficiency of the introduced methods in a nonstationary environment. These experiments show that our algorithms can successfully improve the performance of single piecewise linear filters, and in some cases, even outperform the conventional mixture method.

We have considered the case where the desired data is generated by a nonstationary piecewise linear model with 3 regions. $\mathbf{x}_t = [x_1 \ x_2]^T$ is drawn from a jointly Gaussian random process, and then scaled such that $\mathbf{x}_t \in [0 \ 1]^2$. However, in this experiment, we have divided the total data interval $[0 \ T]$ into 4 disjoint intervals, each of length $T/4$, and used a different 3-region model in each region.

In this experiment, each boosting algorithm uses 5 constituent filters, each of which uses a piecewise linear filter over a 2-region partition. The Accumulated Squared Error (ASE) performance of different methods are compared in Fig. 3. In the Fig. 3, "PLMS", "MIX", and "BPLMS", respectively show a single piecewise linear LMS filter, the ordinary mixture method, and the boosted filters methods. In addition, the suffixes "WU", "RU", and "DR" indicate the weighted updates, random updates, and data reuse methods, respectively. The learning rates for the LMS-based algorithms are set to 0.02, and the desired MSE parameter σ^2 is set to 0.01. Also, the direction vector for the separating hyperplane is set to $\boldsymbol{\theta} = [\theta_1 \ \theta_2 \ -\theta_3]^T$. $\boldsymbol{\theta}$ is consisted of three random variables, each with mean 1, to construct random constituent filters. The results show the superior performance of our algorithms over the single piecewise linear filters, as well as the mixture method, in this highly nonstationary environment. Moreover, as shown in Fig. 3 the data reuse method shows a better performance relative to the other boosting methods. However, according to Table I the random updates method has a significantly lower time consumption, which makes it more desirable for big data applications.

VI. Conclusion

We introduce the boosting concept, extensively studied in machine learning literature, to adaptive filtering context, and propose three different boosting approaches, "weighted updates","data reuse", and "random updates" which are applicable to different adaptive filtering algorithms. We show that by these approaches we can improve the MSE performance of the conventional LMS filters in piecewise linear models, and we provide an upper bound for the weights generated during the algorithm, which lead us to a thorough analysis of the complexity of these methods. We show that the complexity of random updates method is remarkably lower than other two approaches, while the MSE performance does not degrade.

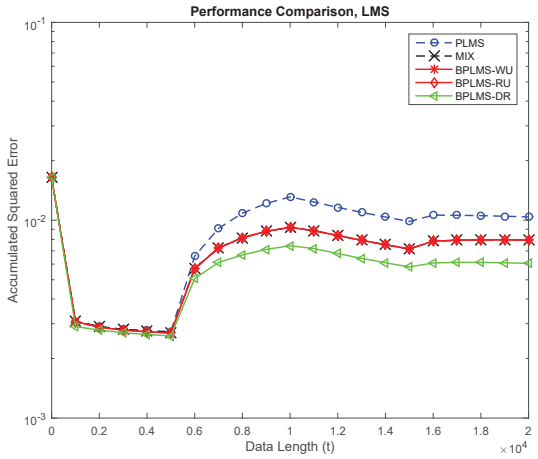

Fig. 3: ASE performance

TABLE I: Time comparison of different methods (seconds)

LMS-MIX	BPLMS-RU	BPLMS-WU	BPLMS-DR
1.576	1.319	1.588	2.564

Therefore, the boosting using random updates approach can be efficiently applied to real life large scale problems.

References

[1] R. E. Schapire and Y. Freund, *Boosting: Foundations and Algorithms*, MIT Press, 2012.

[2] Y. Freund and R. E.Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of Computer and System Sciences*, vol. 55, pp. 119–139, 1997.

[3] D. L. Shrestha and D. P. Solomatine, "Experiments with adaboost.rt, an improved boosting scheme for regression," in *Experiments with AdaBoost.RT, an improved boosting scheme for regression*, 2006.

[4] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*, John Willey and Sons, 2001.

[5] S. S. Kozat, A. T. Erdogan, A. C. Singer, and A. H. Sayed, "Steady state MSE performance analysis of mixture approaches to adaptive filtering," *IEEE Transactions on Signal Processing*, 2010.

[6] S. Shaffer and C. S. Williams, "Comparison of lms, alpha-lms, and data reusing lms algorithms," in *Conference Record of the Seventeenth Asilomar Conference on Circuits, Systems and Computers*, 1983.

[7] N. C. Oza and S. Russell, "Online bagging and boosting," in *Proceedings of AISTATS*, 2001.

[8] L. Bottou and O. Bousquet, "The tradeoffs of large scale learning," in *NIPS*, 2008.

[9] A. H. Sayed, *Fundamentals of Adaptive Filtering*, John Wiley and Sons, 2003.

[10] Shang-Tse Chen, Hsuan-Tien Lin, and Chi-Jen Lu, "An online boosting algorithm with theoretical justifications," in *ICML*, 2012.

[11] N. D. Vanli and S. S. Kozat, "A comprehensive approach to universal piecewise nonlinear regression based on trees," *IEEE Transactions on Signal Processing*, vol. 62, no. 20, pp. 5471–5486, Oct 2014.

[12] P. Malik, "Governing big data: Principles and practices," *IBM J. Res. Dev.*, vol. 57, no. 3-4, pp. 1:1–1:1, May 2013.

[13] S. S Kozat and A. C. Singer, "Universal switching linear least squares prediction," *IEEE Transactions on Signal Processing*, vol. 56, pp. 189–204, Jan. 2008.

[14] Suleyman Serdar Kozat, Andrew C. Singer, and Georg Zeitler, "Universal piecewise linear prediction via context trees.," *IEEE Transactions on Signal Processing*, vol. 55, no. 7-2, pp. 3730–3745, 2007.

[15] R. A. Servedio, "Smooth boosting and learning with malicious noise," *Journal of Machine Learning Research*, vol. 4, pp. 633–648, 2003.

[16] A. Papoulis and S. U. Pillai, *Probability, Random Variables, and Stochastic Processes*, McGraw-Hill Higher Education, 4 edition, 2002.