Boosted Adaptive Filters

Dariush Kari, Ibrahim Delibalta and Suleyman S. Kozat, *Senior Member, IEEE*

***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 filtering algorithms, i.e., the weak learners, that run in parallel on a common task such as equalization, classification, regression or filtering. For each newly received input vector and observation pair, each algorithm adapts itself based on the performance of the other adaptive algorithms in the mixture on this current data pair. These relative updates provide the boosting effect such that the algorithms in the mixture learn a different attribute of the data providing diversity. The outputs of these parallel running algorithms are then combined using adaptive mixture approaches. We demonstrate the intrinsic connections of boosting with the adaptive mixture methods and data reuse algorithms. Specifically, we study parallel running recursive least squares and least mean squares algorithms and provide different boosted versions of these well known adaptation methods. We provide the MSE performance results as well as computational complexity bounds for the boosted adaptive filters. We demonstrate over widely used real life data sets in the machine learning and adaptive signal processing literatures that we can substantially improve the performances of these algorithms due to the boosting effect with a relatively small computational complexity increase.**

***Index Terms*—Boosting, Online boosting, Adaptive filtering, RLS, LMS, Ensemble method.**

I. Introduction

Boosting is considered as one of the most important ensemble learning methods in the machine learning literature extensively used in several different real life applications from classification to regression [1]–[3]. As an ensemble learning method [4], boosting combines several parallel running "weakly" performing algorithms to build a final "strongly" performing algorithm [1]. This is accomplished by finding a linear combination of weak learning algorithms in order to minimize the total loss over a set of training data, usually using a functional gradient descent [2]. Boosting is successfully applied to several different problems in the machine learning literature including classification [2], regression [3], prediction [5], [6] and financial forecasting [7]. 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 [8] and data reuse algorithms [9] widely studied in the adaptive signal processing literature.

This work is supported in part by Turkish Academy of Sciences Outstanding Researcher Programme, TUBITAK Contract No. 113E517, and Turk Telekom Communications Services Incorporated.

Dariush Kari and Suleyman S. Kozat are with the EE Department of Bilkent University, Ankara, Turkey, {kari, kozat}@ee.bilkent.edu.tr, tel: 0312 2902336. Ibrahim Delibalta is with the Turk Telekom Communications Services Inc., Istanbul, Turkey and Graduate School of Social Sciences and Humanities, Koc University, Istanbul, Turkey, ibrahim.delibalta@turktelekom.com.tr

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 [10]. In the online setting, we do not need or 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 [11]. 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 [12]. In this sense, we mainly concentrate on the online boosting framework, which naturally conforms to the widely studied adaptive filtering framework [12] that we are interested in.

Specifically, we have m parallel running adaptive filters (or weak learners [1]) that recieve 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, depending on the target of the applications or problem constraints [12]. After receiving the input vector, each algorithm produces its output and then calculates its instantaneous error after the observation is revealed. In the most generic setting, this estimation/prediction error and the corresponding input vector are then used to adapt the internal parameters of the algorithm to minimize a priori defined loss function, e.g., instantaneous error for the LMS algorithm. 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 [13]. Furthermore, unlike the usual mixture approaches [8], the update of each adaptive filter depends on the previous adaptive filters in the mixture. In particular, at each time t, after the kth filter, calculates its error over $(\mathbf{x}_t, d_t)$ pair, it passes a certain weight to the next adaptive filter, the $(k+1)$th filter, quantifying how much error the constituent filters from 1st to kth made on the current $(\mathbf{x}_t, d_t)$ pair. 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, where several parallel running adaptive algorithms are combined to improve the performance. In the mixture methods, the performance improvement is achieved due to the diversity provided by using several different adaptive algorithm each having either a different view or advantage [8]. This diversity is exploited to yield a final combined algorithm, which achieves a performance better than any of the algorithms in the mixture. Although the online boosting approach is similar to mixture approaches in the adaptive filtering literature [8], however, there are significant differences. In the boosting notion, the parallel running adaptive algorithms are not independent, where one deliberately introduces the diversity by adapting the filters one by one from the first filter to the last mth filter for each new sample based on the performance of all the previous filters on this sample. In this sense, each adaptive algorithm, say the $(k+1)$th filter, receives feedback from the previous filters, i.e., 1st to kth, and updates its inner parameters accordingly. As an example, if the current $(\mathbf{x}_t, d_t)$ is well modeled by the previous filters, then the $(k+1)$th filter performs minor update using $(\mathbf{x}_t, d_t)$ and may put more emphasize to the later arriving samples which may be worse modeled by the previous filters. Therefore, by boosting, each adaptive algorithm in the mixture can concentrate on different parts of the input and output signal pairs achieving diversity and significantly improve the gain. Moreover, we introduce the "random updates" method for boosting, which significantly reduces the computational complexity, while achieving the performance of the mixture methods. This is because in this scenario, the kth filter is updated with probability $\lambda_t^{(k)}$, which depends on the performance of other filters.

The linear filters, such as LMS or RLS, are among the simplest as well as the most widely used adaptive filters in the real-life applications [12]. However, note that although linear filters have a low complexity, piecewise linear filters deliver a significantly superior performance in real life applications [14], with a comparable complexity. These filters mitigate the overfitting, stability and convergence issues tied to nonlinear models [12], [15]–[17], while effectively improving the modeling power relative to linear filters [14]. Therefore, we implement our boosting algorithms on piecewise linear filters. To this end, we first apply the boosting notion to several parallel running piecewise linear RLS-based filters, and introduce three different approaches to use the importance weights [13]. In the first approach, we use the importance weights directly to produce certain weighted RLS algorithms. In the second approach, we use the importance weights to construct data reuse adaptive algorithms. However, we need to emphasize that the data reuse in boosting, such as [10], is significantly different from the usual data reusing approaches in adaptive filtering [9]. As an example, in boosting, the importance weight coming from the kth filter determines the data reuse amount in the $(k+1)$st filter, i.e., it is not used for the kth filter, hence, achieving the diversity. The third approach, 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 latter method can be effectively used for big data processing [18], due to the reduced complexity. The output of the mixture filters, before the boosting step, is also combined using a linear filter to construct the final output of the algorithm. The final combination filter is also updated using the RLS algorithm [8]. The boosting idea is then extended to parallel running piecewise linear LMS-based algorithm similar to the RLS case. For this case, we combine the outputs of the constituent filters using a linear filter, which is trained using the LMS algorithm. For all these different cases, we derive the corresponding mean squared error (MSE) results and provide performance bounds in an individual sequence manner [19], [20].

We start our discussions by introducing the problem setup and background in Section II, where we provide individual sequence as well as MSE convergence results for the RLS and LMS algorithms. We introduce our first boosted adaptive filters using the RLS update in Section III. Three different updates are introduced in this section. We then continue with the boosted LMS algorithms in Section IV. Then, in Section V we provide the mathematical analysis for the MSE performance and computational complexity of the proposed algorithms. The paper concludes with extensive sets of experiments over the well known benchmark data sets and simulation models widely used in the machine learning and adaptive signal processing literatures to demonstrate the significant gains achieved by the boosting notion.

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 $\text{Tr}(\mathbf{A})$ is the trace of the matrix $\mathbf{A}$. Here, $\mathbf{I}_m$ and $\mathbf{0}_m$ represent an identity matrix of dimension $m \times m$ and a vector of all zeros of length m, respectively. Except $\mathbf{I}_m$ and $\mathbf{0}_m$, 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 [21]. However, we also provide steady-state, tracking and transient MSE analysis of the algorithms under widely used statistical models in the signal processing literature [12] for completeness.

Note that although nonlinear filters can outperform linear filters, they usually undergo overfitting, stability, and convergence issues [14], [22]. Furthermore, nonlinear filters generally have higher computational complexities, which limits their use in most of the real-life applications [14], [22]. 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 [14], [22]. Therefore, in this paper, we are particularly interested in piecewise linear filters, which serve as an elegant alternative to linear filters. Nevertheless, for illustration, we first explain the basic principles of linear filters, and their extension to the piecewise linear case. Then, in Sections III and IV, we introduce our algorithm in a piecewise linear model.

The linear filters are considered as the simplest adaptive filters, which estimate the desired data d_t by a linear model as $\hat{d}_t = \mathbf{w}_t^T\mathbf{x}_t$,, where $\mathbf{w}_t$ is the linear adaptive filter coefficients at time t. We emphasize that (II) also covers the affine model if one includes a constant term in $\mathbf{x}_t$, hence we use the purely linear form for notational simplicity. When the true d_t is revealed, the algorithm updates its linear filter coefficients $\mathbf{w}_t$ based on the error e_t. As an example, in the basic implementation of the RLS algorithm, the linear filter coefficients are selected to minimize the accumulated squared regression error up to time $t-1$ as

$$\mathbf{w}_t = \arg\min_{\mathbf{w}} \sum_{l=1}^{t-1}(d_l - \mathbf{x}_l^T\mathbf{w})^2, \\ = \left(\sum_{l=1}^{t-1}\mathbf{x}_l\mathbf{x}_l^T\right)^{-1}\left(\sum_{l=1}^{t-1}\mathbf{x}_l d_l\right). \quad (2)$$

where $\mathbf{w}$ is a fixed vector of coefficients. The RLS algorithm is shown to enjoy several optimality properties under different statistical settings [12]. Apart from these results and more related to the framework of this paper, the RLS algorithm is also shown to be rate optimal in an individual sequence manner [23]. As shown in [23] (Section V), when applied to any sequences $\{\mathbf{x}_t\}_{t\geq 1}$ and $\{d_t\}_{t\geq 1}$, the accumulated squared error of the RLS algorithm is as small as the accumulated squared error of the best batch LS filter that is directly optimized for these realizations of the sequences, i.e., for all n, $\{\mathbf{x}_t\}_{t\geq 1}$ and $\{d_t\}_{t\geq 1}$, the RLS filter achieves

$$\sum_{l=1}^{n}(d_l - \mathbf{x}_l^T\mathbf{w}_l)^2 - \min_{\mathbf{w}}\sum_{l=1}^{n}(d_l - \mathbf{x}_l^T\mathbf{w})^2 \leq O(\ln n). \quad (3)$$

The RLS algorithm is a member of the Follow-the-Leader type algorithms [20] (Section 3), where one uses the best performing linear model up to time $t-1$ to predict d_t. Hence, (3) follows by direct application of the online convex optimization results [24] after regularization. The convergence rate (or the rate of the regret) of the RLS algorithm is also shown to be optimal so that $O(\ln n)$ in the upper bound cannot be improved [19]. It is also shown in [19] that one can reach the optimal upper bound (with exact scaling terms) by using a slightly modified version of (3)

$$\mathbf{w}_t = \left(\sum_{l=1}^{t}\mathbf{x}_l\mathbf{x}_l^T\right)^{-1}\left(\sum_{l=1}^{t-1}\mathbf{x}_l d_l\right). \quad (4)$$

Note that the extension (4) of (3) is a forward algorithm (Section 5 of [25]) and one can show that, in the scalar case, the predictions of (4) are always bounded (which is not the case for (3)) [19].

We emphasize that in the basic application of the RLS algorithm all data pairs $(d_l, \mathbf{x}_l)$, $l = 1, \ldots, t$, receive the same "importance" or weight in (2). Although there exists exponentially weighted or windowed versions of the basic RLS algorithm [12], these methods weight (or concentrate on) the most recent samples to model the nonstationary better [12]. However, in boosting framework [2], each sample pair receives a different weight based on not only those weighting schemes, but also the performance of the boosted algorithms on this pair. As an example, if an algorithm performs worse on a sample, the next iteration concentrates more on this example to better rectify this mistake. In the following sections, we use this notion to derive different boosted adaptive filters.

We use a piecewise linear adaptive filtering method, such that the desired signal is predicted as

$$\hat{d}_t = \sum_{i=1}^{N} s_{i,t}\mathbf{w}_{i,t}^T\mathbf{x}_t, \quad (5)$$

where $s_{i,t}$ is the indicator function of the ith region, i.e.,

$$s_{i,t} = \begin{cases} 1 & \text{if } \mathbf{x}_t \in \mathcal{R}_i \\ 0 & \text{if } \mathbf{x}_t \notin \mathcal{R}_i. \end{cases} \quad (6)$$

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

$$\hat{d}_t = f_t(\mathbf{x}_t) = s_{1,t}\mathbf{w}_{1,t}^T\mathbf{x}_t + s_{2,t}\mathbf{w}_{2,t}^T\mathbf{x}_t \\ = s_t\mathbf{w}_{1,t}^T\mathbf{x}_t + (1-s_t)\mathbf{w}_{2,t}^T\mathbf{x}_t, \quad (7)$$

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.

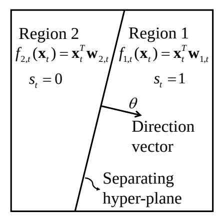


Fig. 1: A sample 2-region partition of the input vector (i.e., $\mathbf{x}_t$) space, which is 2-dimensional in this example. s_t determines whether $\mathbf{x}_t$ is in Region 1 or not, hence, can be used as the indicator function for this region. Similarly, $1-s_t$ serves as the indicator function of Region 2.

III. Boosted RLS 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, $k = 1, \ldots, m$. 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 $\hat{d}_t = \mathbf{z}_t^T\mathbf{y}_t$ [8], 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 RLS 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

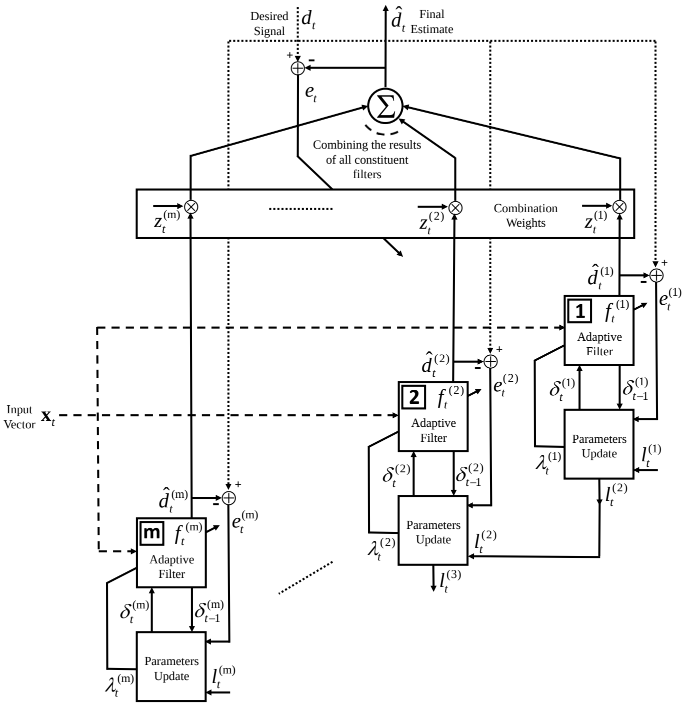


Fig. 2: The block diagram of a boosted adaptive filtering system that uses the input vector $\mathbf{x}_t$ to produce the final estimate $\hat{d}_t$. There are m constituent filters $f_t^{(1)}, \ldots, f_t^{(m)}$, each of which is an adaptive piecewise linear filter that generates its own estimate $\hat{d}_t^{(k)}$. 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 MSE of the kth filter over the first t estimations, and is used in computing $\lambda_t^{(k)}$, as detailed in Fig. 3.

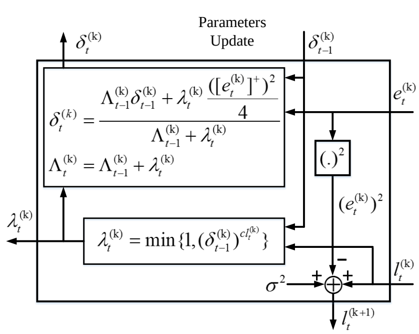


Fig. 3: Parameters update block of the kth constituent filter, which is embedded in the kth filter block as depicted in Fig. 2. This block receives the parameter $l_t^{(k)}$ provided by the $(k-1)$th filter, and uses that in computing $\lambda_t^{(k)}$. It also computes $l_t^{(k+1)}$ and passes it to the $(k+1)$th filter. The parameter $[e_t^{(k)}]^+$ represents the error of the thresholded estimate as explained in (10), and $\Lambda_t^{(k)}$ shows the sum of the weights $\lambda_1^{(k)}, \ldots, \lambda_t^{(k)}$. The WMSE parameter $\delta_{t-1}^{(k)}$ represents the time averaged weighted square error made by the kth filter up to time $t-1$.

that, the $(k+1)$th filter receives a "total loss" parameter, $l_t^{(k+1)}$, from the filter $f_t^{(k)}$, as (Refer to Fig. 3)

$$l_t^{(k+1)} = l_t^{(k)} + \left[\sigma^2 - \left(d_t - f_t^{(k)}(\mathbf{x}_t)\right)^2\right], \qquad (8)$$

to compute a weight $\lambda_t^{(k)}$. The total loss parameter $l_t^{(k)}$, indi-

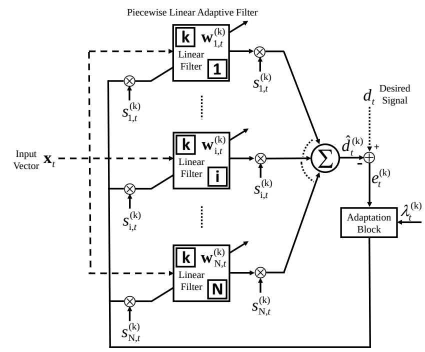


Fig. 4: A sample piecewise linear adaptive filter, used as the kth constituent filter in the system depicted in Fig. 2. This fliter consists of N linear filters, one of which produces the estimate at each iteration t. Based on where the input vector at time t, $\mathbf{x}_t$, lies in the input vector space, one of the $s_{i,t}^{(k)}$'s is 1 and all others are 0. Hence, at each iteration only one of the linear filters is used for estimation and upadated correspondingly.

cates the sum of the differences between the desired MSE (σ^2) and the squared error of the first $k-1$ filters at time t, as shown in Fig. 3. 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 $(d_t, \mathbf{x}_t)$ 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 [13], [26]. We define the weights as

$$\lambda_t^{(k)} = \min\left\{1, \left(\delta_{t-1}^{(k)}\right)^{c\, l_t^{(k)}}\right\}, \qquad (9)$$

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 [8], 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 [13], [26], $\left(1 - \delta_{t-1}^{(k)}\right)$ is set to the classification advantage of the weak learners [26], 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

$$\delta_t^{(k)} = \frac{\sum_{\tau=1}^{t} \frac{\lambda_\tau^{(k)}}{4} \left(d_\tau - \left[f_\tau^{(k)}(\mathbf{x}_\tau) \right]^+ \right)^2}{\sum_{\tau=1}^{t} \lambda_\tau^{(k)}} = \frac{\Lambda_{t-1}^{(k)} \delta_{t-1}^{(k)} + \frac{\lambda_t^{(k)}}{4} \left(d_t - \left[f_t^{(k)}(\mathbf{x}_t) \right]^+ \right)^2}{\Lambda_{t-1}^{(k)} + \lambda_t^{(k)}} \quad (10)$$

where $\Lambda_t^{(k)} \triangleq \sum_{\tau=1}^{t} \lambda_\tau^{(k)}$, and $\left[f_\tau^{(k)}(\mathbf{x}_\tau) \right]^+$ 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 (10) can be recursively calculated as in Fig. 3.

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 [8]. Furthermore, following this idea, in (9), 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 RLS algorithm.

A. Directly Using λ's as Sample Weights

As depicted in Fig. 4, each constituent filter is a piecewise linear filter consisting of N linear filters. At each time t, all of the constituent filters (shown in Fig. 2) estimate the desired data d_t in parallel, and the final estimate is a linear combination of the results generated by the constituent filters. However, at each time t, exactly one of the N linear filters in each constituent filter is used for estimating d_t. Correspondingly, when we update the constituent filters, only the filter that has been used for the estimation will be updated. To this end, we use the indicator function $s_{i,t}^{(k)}$ for the ith linear filter embedded in the kth constituent filter, as was explained in Section II. Therefore, at each time t, only the filters whose indicator functions equal 1, will be updated. When the kth constituent filter receives the weigh $\lambda_t^{(k)}$, it updates the linear coefficients $\mathbf{w}_{i,t}^{(k)}$, assuming that $\mathbf{x}_t$ lies in the ith region of the kth constituent filter. We consider $\lambda_t^{(k)}$ as the weight for the observation pair $(d_t, \mathbf{x}_t)$ and apply a weighted RLS update to $\mathbf{w}_{i,t}^{(k)}$. For this particular weighted RLS algorithm, we define the autocorrelation matrix and the cross correlation vector as

$$\mathbf{R}_{i,t+1}^{(k)} \triangleq \beta \mathbf{R}_{i,t}^{(k)} + \lambda_t^{(k)} \mathbf{x}_t \mathbf{x}_t^T, \quad (11)$$

$$\mathbf{p}_{i,t+1}^{(k)} \triangleq \beta \mathbf{p}_{i,t}^{(k)} + \lambda_t^{(k)} \mathbf{x}_t d_t, \quad (12)$$

where β is the forgetting factor [12] and $\mathbf{w}_{i,t+1}^{(k)} = \left(\mathbf{R}_{i,t+1}^{(k)} \right)^{-1} \mathbf{p}_{i,t+1}^{(k)}$ can be calculated in a recursive manner as

$$\begin{aligned} e_t^{(k)} &= d_t - \mathbf{x}_t^T \mathbf{w}_{i,t}^{(k)}, \\ \mathbf{g}_{i,t}^{(k)} &= \frac{\lambda_t^{(k)} \mathbf{P}_{i,t}^{(k)} \mathbf{x}_t}{\beta + \lambda_t^{(k)} \mathbf{x}_t^T \mathbf{P}_{i,t}^{(k)} \mathbf{x}_t}, \\ \mathbf{w}_{t+1}^{(k)} &= \mathbf{w}_{i,t}^{(k)} + e_t^{(k)} \mathbf{g}_{i,t}^{(k)}, \\ \mathbf{P}_{i,t+1}^{(k)} &= \beta^{-1} \left(\mathbf{P}_{i,t}^{(k)} - \mathbf{g}_{i,t}^{(k)} \mathbf{x}_t^T \mathbf{P}_{i,t}^{(k)} \right). \end{aligned} \quad (13)$$

where $\mathbf{P}_{i,t}^{(k)} \triangleq \left(\mathbf{R}_{i,t}^{(k)} \right)^{-1}$, and $\mathbf{P}_{i,0}^{(k)} = v^{-1} \mathbf{I}$ for $i = 1, \ldots, N$, and $0 < v \ll 1$.

Remark 1: We emphasize that one can be inclined to use a single boosted RLS algorithm instead of running m RLS algorithms in parallel due to the computational complexity considerations. In this single RLS implementation, the calculated importance weight can be used as a weight in the next time instant $t+1$ so that a single RLS algorithm may concentrate more on wrongly regressed samples. However, running m RLS filters with boosting provides "diversity" [8], where each constituent algorithm concentrates on different parts of the data, where the other filters are not successful, due to the boosting weights in (9). Hence, by this boosting and then final mixture-of-experts combination [8], we achieve a significantly improved filtering performance.

B. Data Reuse Approaches Based on the Weights

Another approach follows Ozaboost [10]. In this approach, from $\lambda_t^{(k)}$, we generate an integer, say $n_t^{(k)} = \text{ceil}(K \lambda_t^{(k)})$, and apply the RLS update on the $(d_t, \mathbf{x}_t)$ pair repeatedly $n_t^{(k)}$ times, i.e., run the RLS update on the same $(d_t, \mathbf{x}_t)$ pair $n_t^{(k)}$ times consecutively. Here, K is an integer, as an example, we use $K = 2$ in our simulations. The final $\mathbf{w}_{t+1}^{(k)}$ is calculated after $n_t^{(k)}$ RLS updates. As a major advantage, clearly, this reusing approach can be readily generalized to other adaptive algorithms in a straightforward manner.

We point out that Ozaboost [10] uses a different data reuse strategy. In this approach, $\lambda_t^{(k)}$ is used a parameter of a Poisson distribution and an integer $n_t^{(k)}$ is randomly generated from this Poisson distribution. We then apply the RLS update $n_t^{(k)}$ times.

C. Random Updates Approach Based on The Weights

In this approach, we simply use the weight $\lambda_t^{(k)}$ as a probability of updating the kth filter at time t. To this end, we generate a Bernoulli random variable, which is 1 with probability $\lambda_t^{(k)}$ and is 0 with probability $1 - \lambda_t^{(k)}$. Then, in each of the constituent filters, we update one of the linear filters that is involved in estimation process, only if the Bernoulli random variable equals 1. With this method, we significantly reduce the computational complexity of the algorithm. Moreover, due to the dependence of this Bernoulli random variable on the MSE performance of the previous constituent filters, this method does not degrade the MSE performance severely, while offering a considerably lower complexity, i.e., when the MSE is low, there is no need for further updates, hence, the probability of an update is low, while, this probability is larger when the MSE is high.

Algorithm 1 Boosted RLS with the weighting scheme in (13)

Input: $(\mathbf{x}_t, d_t)$ (data stream), m (number of RLS 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 RLS 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 RLS 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
  $e_t = d_t - \mathbf{z}_t^T \mathbf{y}_t$;
  $\mathbf{g}_t = \frac{\mathbf{P}_t \mathbf{y}_t}{\lambda + \mathbf{y}_t^T \mathbf{P}_t \mathbf{y}_t}$;
  $\mathbf{z}_{t+1} = \mathbf{z}_t + e_t \mathbf{g}_t$;
  $\mathbf{P}_{t+1} = \lambda^{-1}\mathbf{P}_t - \lambda^{-1}\mathbf{g}_t \mathbf{y}_t^T \mathbf{P}_t$;
end for

D. The Final Algorithm

After d_t is revealed, we also update the final combination weights $\mathbf{z}_t$ based on the final output $\hat{d}_t = \mathbf{z}_t^T \mathbf{y}_t$, where $\hat{d}_t = \mathbf{z}_t^T \mathbf{y}_t$, $\mathbf{y}_t = [\hat{d}_t^{(1)}, \ldots, \hat{d}_t^{(m)}]^T$. To update the final combination weights, we use an exponentially weighted RLS algorithm yielding [8]

$$\mathbf{R}_{t+1} = \lambda \mathbf{R}_t + \mathbf{y}_t \mathbf{y}_t^T,$$
$$\mathbf{p}_{t+1} = \lambda \mathbf{p}_t + \mathbf{y}_t d_t,$$

and

$$\begin{aligned} e_t &= d_t - \mathbf{z}_t^T \mathbf{y}_t, \\ \mathbf{g}_t &= \frac{\mathbf{P}_t \mathbf{y}_t}{\lambda + \mathbf{y}_t^T \mathbf{P}_t \mathbf{y}_t}, \\ \mathbf{z}_{t+1} &= \mathbf{z}_t + e_t \mathbf{g}_t, \\ \mathbf{P}_{t+1} &= \lambda^{-1}\mathbf{P}_t - \lambda^{-1}\mathbf{g}_t \mathbf{y}_t^T \mathbf{P}_t, \end{aligned} \tag{14}$$

where $0 < \lambda \leq 1$ is the exponential weighting. The complete algorithm is given in Algorithm 1 with the weighted RLS implementation in (13).

IV. Boosted LMS Algorithms

In this case, as shown in Fig. 2, we have m parallel running piecewise linear filters, each of which updated using LMS algorithm with a different learning rate, i.e., if the input vector $\mathbf{x}_t$ lies in the ith region of the kth filter partition, $s_{i,t}^{(k)} = 1$, hence, we use $\mathbf{w}_{i,t}^{(k)}$ to estimate d_t, and update this linear filter with its own learning rate $\mu_i^{(k)}$. Based on the weights given in (9) and the total loss and MSE parameters in equations (8) and (10), we next introduce three LMS based boosting algorithms, similar to those introduced in Section III.

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

We note that by construction method in (9), $0 < \lambda_t^{(k)} \leq 1$, thus, 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

$$\mathbf{w}_{i,t+1}^{(k)} = \left(\mathbf{I} - \mu_i^{(k)} \lambda_t^{(k)} \mathbf{x}_t \mathbf{x}_t^T\right) \mathbf{w}_{i,t}^{(k)} + \mu_i^{(k)} \lambda_t^{(k)} \mathbf{x}_t d_t, \tag{15}$$

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 filters of different constituent filters will have different learning rates $\mu_i \lambda_t^{(k)}$.

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 to obtain the $\mathbf{w}_{t+1}^{(k)}$ as

$$\begin{aligned} &\mathbf{q}^{(0)} = \mathbf{w}_{i,t}^{(k)}, \\ &\mathbf{q}^{(a)} = \left(\mathbf{I} - \mu_i^{(k)} \mathbf{x}_t \mathbf{x}_t^T\right) \mathbf{q}^{(a-1)} + \mu_i^{(k)} \mathbf{x}_t d_t,\ a = 1, \ldots, n_t^{(k)}, \\ &\mathbf{w}_{t+1}^{(k)} = \mathbf{q}^{\left(n_t^{(k)}\right)}. \end{aligned} \tag{16}$$

Similar to the RLS case, if we follow the Ozaboost [10], we use the weights to generate a random number $n_t^{(k)}$ from a Poisson distribution with parameter $\lambda_t^{(k)}$, and perform the LMS update $n_t^{(k)}$ times on $\mathbf{w}_{i,t}^{(k)}$ as explained above.

C. Random Updates Based on the Weights

Again, in this scenario, similar to the RLS case, 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.

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

$$\mathbf{z}_{t+1} = \left(\mathbf{I} - \mu \mathbf{y}_t \mathbf{y}_t^T\right) \mathbf{z}_t + \mu \mathbf{y}_t d_t, \tag{17}$$

where $\mu > 0$ and $\mathbf{y}_t = [\hat{d}_t^{(1)}, \ldots, \hat{d}_t^{(m)}]^T$.

The complete final algorithm is similar to Algorithm 1, with the weighted LMS implementation in (15), i.e., for updating the constituent filters (line 11 in Algorithm 1), we use the LMS recursion given in (15) (for weighted updates), and for updating the combination weights $\mathbf{z}_t$, we use the LMS update given in (17).

Remark 2: Similar updates can be written for all different types of online learning methods.

Remark 3: We supposed that each constituent filter is built up based upon a fixed partition, which means that the partition cannot be updated during the algorithm. However, one can use a method similar to that in [14] to make the partitioning adaptive. As an example, suppose that each constituent filter is defined on a 2-region partition, as shown in Fig. 1, the regions of which are separated using a hyper-plane with the direction vector $\boldsymbol{\theta}_t^{(k)}$, which is going to be updated at each time t. In order to boost the performance of a system made up of N such piecewise linear filters, we not only apply the weights effects to update the linear filters updates in each region of each constituent filter, but also update the direction vectors $\boldsymbol{\theta}_t^{(k)}$ in a boosted manner. In order to indicate the region in which $\mathbf{x}_t$ lies, we use an indicator function $s_t^{(k)}$ defined as follows

$$s_t^{(k)} = \frac{1}{1+\exp(-\boldsymbol{\theta}_t^T \mathbf{x}_t)}, \tag{18}$$

and the estimate made by the kth filter is represented by

$$\hat{d}_t^{(k)} = s_t^{(k)} \hat{d}_{1,t}^{(k)} + \left(1 - s_t^{(k)}\right) \hat{d}_{2,t}^{(k)} \tag{19}$$

which, yields the following ordinary LMS update for $\boldsymbol{\theta}_t^{(k)}$ [14]

$$\begin{aligned}\boldsymbol{\theta}_{t+1}^{(k)} &= \boldsymbol{\theta}_t^{(k)} + \mu_{\boldsymbol{\theta}} e_t^{(k)} \left(\hat{d}_{1,t}^{(k)} - \hat{d}_{2,t}^{(k)}\right) \nabla_{\boldsymbol{\theta}_t} \left(s_t^{(k)}\right) \\ &= \boldsymbol{\theta}_t^{(k)} + \mu_{\boldsymbol{\theta}} e_t^{(k)} \left(\hat{d}_{1,t}^{(k)} - \hat{d}_{2,t}^{(k)}\right) s_t^{(k)} \left(1 - s_t^{(k)}\right) \mathbf{x}_t.\end{aligned} \tag{20}$$

Then, in "random updates" scenario we either will or will not perform this update with probabilities $\lambda_t^{(k)}$ and $1-\lambda_t^{(k)}$, respectively, and for "weighted updates" scenario we achieve the following update for $\boldsymbol{\theta}_t^{(k)}$

$$\boldsymbol{\theta}_{t+1}^{(k)} = \boldsymbol{\theta}_t^{(k)} + \mu_{\boldsymbol{\theta}} \lambda_t^{(k)} e_t^{(k)} \left(\hat{d}_{2,t}^{(k)} - \hat{d}_{1,t}^{(k)}\right) s_t^{(k)} \left(1 - s_t^{(k)}\right) \mathbf{x}_t. \tag{21}$$

However, for the "data reuse" scenario, we perform this update $n_t^{(k)} = \text{ceil}(\lambda_t^{(k)} K)$ times, along with updating the linear filters coefficients, which results in

$$\begin{aligned}\boldsymbol{\vartheta}^{(a+1)} &= \boldsymbol{\vartheta}^{(a)} + \mu_{\boldsymbol{\theta}} \epsilon^{(a)} \mathbf{x}_t \mathbf{x}_t^T \left(\mathbf{q}_1^{(a)} - \mathbf{q}_2^{(a)}\right) \psi^{(a)} \left(1 - \psi^{(a)}\right), \\ \mathbf{q}_1^{(a+1)} &= \mathbf{q}_1^{(a)} + \mu_i^{(k)} \psi^{(a)} \epsilon^{(a)} \mathbf{x}_t, \\ \mathbf{q}_2^{(a+1)} &= \mathbf{q}_2^{(a)} + \mu_i^{(k)} (1-\psi^{(a)}) \epsilon^{(a)} \mathbf{x}_t, \\ \psi^{(a+1)} &= \frac{1}{1+\exp(-\boldsymbol{\vartheta}_t^T \mathbf{x}_t)}, \\ \epsilon^{(a+1)} &= d_t - \left(\psi^{(a+1)} \mathbf{q}_1^{(a+1)} + (1-\psi^{(a+1)}) \mathbf{q}_2^{(a+1)}\right) \mathbf{x}_t,\end{aligned} \tag{22}$$

where $a = 0, \ldots, (n_t^{(k)} - 1)$, $\boldsymbol{\vartheta}^{(0)} = \boldsymbol{\theta}_t^{(k)}$, $\epsilon^{(0)} = e_t^{(k)}$,$\psi^{(0)} = s_t^{(k)}$, and $\mathbf{q}_i^{(0)} = \mathbf{w}_{i,t}^{(k)}$ for $i = 1, 2$. Also, the updated values are $\boldsymbol{\theta}_{t+1}^{(k)} = \boldsymbol{\vartheta}^{(n_t^{(k)})}$, and $\mathbf{w}_{i,t+1}^{(k)} = \mathbf{q}_i^{(n_t^{(k)})}$ for $i = 1, 2$.

V. Analysis Of The Proposed Algorithms

In this section we provide the MSE as well as complexity analysis for the proposed algorithms. We prove an upper bound for the weights $\lambda_t^{(k)}$, which is significantly less than 1. This bound shows that the complexity of the "random updates" algorithm is significantly less than other proposed algorithms, and slightly greater than that of a single piecewise linear filter. Hence, it shows the considerable advantage of "boosting with random updates" in processing high dimensional data. Furthermore, we use this bound in MSE analysis of the algorithms. In addition, for the sake of simplicity, we have chosen the "dependence parameter" $c = 1$. Nevertheless, the results can be easily extended to the general case.

A. Complexity Analysis

In this section we compare the complexity of the proposed algorithms and find an upper bound for the computational complexity of random updates scenario (introduced in Section III-C for RLS, and in Section IV-C for LMS updates), which shows its significantly lower computational burden with respect to two other approaches. 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 if updated using the RLS algorithm, requires $O(r^2)$ computations due to updating the matrix $\mathbf{R}_{i,t}^{(k)}$, while it needs $O(r)$ computations when updated using the LMS method (in their most basic implementations).

We first derive the computational complexity of using the RLS 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 $O(mr^2)$ 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, yielding

$$C_t^{(k)} = \begin{cases} O(r^2) & \text{with probability } \lambda_t^{(k)} \\ O(r) & \text{with probability } 1-\lambda_t^{(k)}, \end{cases} \tag{23}$$

where $C_t^{(k)}$ indicates the complexity of running the kth filter at iteration t. Therefore, the total computational complexity C_t at iteration t will be $C_t = \sum_{k=1}^m C_t^{(k)}$, which yields

$$E\left[C_t\right] = E\left[\sum_{k=1}^m C_t^{(k)}\right] = \sum_{k=1}^m E[\lambda_t^{(k)}] O(r^2) \tag{24}$$

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

$$E\left[C_t\right] < \sum_{k=1}^m \tilde{\lambda}^{(k)} O(r^2). \tag{25}$$

In Theorem 1, 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 RLS update $\lambda_t^{(k)} K$ times, resulting a computational complexity of order $E\left[C_t\right] < \sum_{k=1}^m K \; E[\tilde{\lambda}^{(k)}](O(r^2))$. For the LMS updates, we similarly obtain the computational complexities $O(mr)$, $O(\tilde{\lambda}^{(k)}r)$, and $O(K\tilde{\lambda}^{(k)}r)$, for the "weighted samples", "random updates", and "data reuse" scenarios respectively.

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

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,*

$$E\left[\lambda_t^{(k)}\right] \leq \tilde{\lambda}^{(k)} = \left(\gamma^{-2\sigma^2}(1+2\zeta^2\ln\gamma)\right)^{\frac{1-k}{2}}, \qquad (26)$$

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 results, 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.

Proof: For simplicity, in this proof, we have assumed that $c=1$, however, the results are readily extended to the general values of c. We construct our proof based on the following assumption:

Assumption: assume that $e_t^{(k)}$'s are independent and identically distributed (i.i.d) zero-mean Gaussian random variables with variance ζ^2.

We have

$$\begin{aligned} E\left[\lambda_t^{(k)}\right] &= E\left[\min\left\{1, \left(\delta_{t-1}^{(k)}\right)^{l_t^{(k)}}\right\}\right] \\ &\leq \min\left\{1, E\left[\left(\delta_{t-1}^{(k)}\right)^{l_t^{(k)}}\right]\right\} \end{aligned} \qquad (27)$$

Now, we show that under certain conditions, $E\left[\left(\delta_{t-1}^{(k)}\right)^{l_t^{(k)}}\right]$ will be less than 1, hence, we get an upper bound for $E[\lambda_t^{(k)}]$. We define $s \triangleq \ln(\delta_{t-1}^{(k)})$, yielding

$$E\left[\left(\delta_{t-1}^{(k)}\right)^{l_t^{(k)}}\right] = E\left[E\left[\exp\left(s\ l_t^{(k)}\right)\Big|s\right]\right] = E\left[M_{l_t^{(k)}}(s)\Big|s\right], \qquad (28)$$

where $M_{l_t^{(k)}}(.)$ is the moment generating function of the random variable $l_t^{(k)}$. From the Algorithm 1, $l_t^{(k)} = (k-1)\sigma^2 - \sum_{j=1}^{k-1}\left(e_t^{(j)}\right)^2$. According to the Assumption, $\frac{e_t^{(j)}}{\zeta}$ is a standard normal random variable. Therefore, $\sum_{j=1}^{k-1}\left(e_t^{(j)}\right)^2$ has a Gamma distribution as $\Gamma\left(\frac{k-1}{2}, 2\zeta^2\right)$ [27], which results in the following moment generating function for $l_t^{(k)}$

$$\begin{aligned} M_{l_t^{(k)}}(s) &= \exp\left(s(k-1)\sigma^2\right)\left(1+2\zeta^2 s\right)^{\frac{1-k}{2}} \\ &= \left(\delta_{t-1}^{(k)}\right)^{(k-1)\sigma^2}\left(1+2\zeta^2\ln\left(\delta_{t-1}^{(k)}\right)\right)^{\frac{1-k}{2}}. \end{aligned} \qquad (29)$$

In the above equality $\delta_{t-1}^{(k)}$ is a random variable, the mean of which is denoted by γ. We point out that γ will approach to ζ^2 in convergence. We define a function $\varphi(.)$ such that $E\left[\lambda_t^{(k)}\right] = E\left[\varphi\left(\delta_{t-1}^{(k)}\right)\right]$, and seek to find a condition for $\varphi(.)$ to be a concave function. Then, by using Jenssen's inequality for concave functions, we have

$$E\left[\lambda_t^{(k)}\right] \leq \varphi(\gamma). \qquad (30)$$

Inspired by (29), we define $A\left(\delta_{t-1}^{(k)}\right) \triangleq {\delta_{t-1}^{(k)}}^{-2\sigma^2}\left(1+2\zeta^2\ln\left(\delta_{t-1}^{(k)}\right)\right)$ and $\varphi\left(\delta_{t-1}^{(k)}\right) \triangleq \left(A\left(\delta_{t-1}^{(k)}\right)\right)^{\frac{1-k}{2}}$. By these definitions we obtain

$$\begin{aligned} \varphi''\left(\delta_{t-1}^{(k)}\right) = \frac{1-k}{2}\left(A\left(\delta_{t-1}^{(k)}\right)\right)^{\frac{-k-3}{2}}\Bigg[\left(\frac{-k-1}{2}\right)\left(A'\left(\delta_{t-1}^{(k)}\right)\right)^2 \\ + \left(A\left(\delta_{t-1}^{(k)}\right)\right)^2 A''\left(\delta_{t-1}^{(k)}\right)\Bigg]. \end{aligned} \qquad (31)$$

Considering that $k>1$, in order for $\varphi(.)$ to be concave, it suffices to have

$$\left(A\left(\delta_{t-1}^{(k)}\right)\right)^2 A''\left(\delta_{t-1}^{(k)}\right) > \left(\frac{k+1}{2}\right)\left(A'\left(\delta_{t-1}^{(k)}\right)\right)^2, \qquad (32)$$

which reduces to the following necessary and sufficient conditions:

$$\frac{\left(\delta_{t-1}^{(k)}\right)^{2\sigma^2}}{\left(1+2\zeta^2\ln\left(\delta_{t-1}^{(k)}\right)\right)^2} < \frac{\left(1+2\sigma^2\right)^2}{4(k+1)}, \qquad (33)$$

and

$$\frac{(1-\xi_1)\sigma^2}{1-2\sigma^2\ln\left(\delta_{t-1}^{(k)}\right)} < \zeta^2 < \frac{(1-\xi_2)\sigma^2}{1-2\sigma^2\ln\left(\delta_{t-1}^{(k)}\right)}, \qquad (34)$$

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

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

Under these conditions, $\varphi(.)$ is concave, therefore, by substituting $\varphi(.)$ in (30) we achieve (26). This concludes the proof of Theorem. □

B. MSE Analysis

We provide the MSE analysis for the "weighted updates" scenarios, which can be straightforwardly extended to the other scenarios. Consider that each constituent filter consists of N different linear adaptive filters, as shown in Fig. 1. Based on the region in which the input vector $\mathbf{x}_t$ lies, one of these filters is used for estimating d_t. Note that we have considered a fixed partition over the space of the input vectors. Suppose that in each region, say ith region, which is corresponding to the ith linear filter of the constituent filter k, there is a vector $\mathbf{w}_{o,i}^{(k)}$ such that

$$d_t = \sum_{i=1}^{N} s_{i,t}^{(k)} \left[\left(\mathbf{w}_{o,i}^{(k)} \right)^T \mathbf{x}_t + \nu_t \right], \quad k = 1, \ldots, m$$

where we have supposed that the estimation error ν_t is independent of the region to which $\mathbf{x}_t$ belongs, and has a variance of σ_ν^2. Moreover, suppose that the input vector distributes over the regions of the kth constituent filter, with a probability distribution $\pi^{(k)}$, i.e., $\mathbf{x}_t$ lies in the ith region of the kth constituent filter with probability $\pi_i^{(k)}$.

We first analyze the LMS case. Therefore, under the general separation assumptions (see [12], chapter 6), and given that all linear filters converge, the MSE of the kth constituent filter at the steady state can be expressed as [12]

$$\gamma^{(k)} = \sum_{i=1}^{N} \pi_i^{(k)} \sigma_\nu^2 \left(1 + \frac{\mu \tilde{\lambda}^{(k)} \mathrm{Tr}(\mathbf{R}_{\mathbf{X}_i})}{2 - \mu \tilde{\lambda}^{(k)} \mathrm{Tr}(\mathbf{R}_{\mathbf{X}_i})} \right). \tag{35}$$

where $\mathbf{R}_{\mathbf{X}_i}$ shows the covariance matrix of the input vectors lying in the ith region. Assuming that $\mathbf{x}_t$ is i.i.d and has the covariance matrix $\mathbf{R}_{\mathbf{X}}$, we have the following MSE for the kth constituent filter at the steady state

$$\gamma_{LMS}^{(k)} = \sigma_\nu^2 \left(1 + \frac{\mu \tilde{\lambda}^{(k)} \mathrm{Tr}(\mathbf{R}_{\mathbf{X}})}{2 - \mu \tilde{\lambda}^{(k)} \mathrm{Tr}(\mathbf{R}_{\mathbf{X}})} \right). \tag{36}$$

In order to find the MSE of the final estimate, i.e., the estimate obtained by combining the results of all constituent filters, we define the covariance matrix $\mathbf{R}_{\mathbf{Y}}$ as $\mathbf{R}_{\mathbf{Y}} \triangleq E\left[\mathbf{y}\mathbf{y}^T\right]$, which leads to

$$\mathrm{Tr}\left(\mathbf{R}_{\mathbf{Y}}\right) = E\left[\mathbf{y}^T\mathbf{y}\right] = \sum_{k=1}^{m} \left(\sigma_d^2 + \gamma^{(k)} \right). \tag{37}$$

We further assume that the d_t can be modeled as $d_t = \mathbf{y}^T \mathbf{z}_o + \omega_t$, in which, ω_t is the estimation noise with the variance σ_ω^2, and in general different from ν_t. From [12] (chapter 6), we get

$$\eta_{LMS} = \left(\frac{\mu_{\mathbf{Z}} \mathrm{Tr}(\mathbf{R}_{\mathbf{Y}})}{2 - \mu_{\mathbf{Z}} \mathrm{Tr}(\mathbf{R}_{\mathbf{Y}})} + 1 \right) \sigma_\omega^2, \tag{38}$$

where $\mu_{\mathbf{Z}}$ is the step size used for updating the combination coefficients $\mathbf{z}_t$.

Now, consider the RLS case, in which we use (13) for updating each $\mathbf{w}_{i,t}^{(k)}$. We use the same model and assumptions as the LMS case. In this case, although the input vectors $\mathbf{x}_t$ do not appear with the same weight in computing the autocovariance matrix $\mathbf{R}_{i,t}^{(k)}$, it is reasonable to assume the same weight for all input vectors in steady state. We assume that the weights converge to the upper bound obtained in the Theorem. Thus, from [12] (chapter 6), we get

$$\gamma_{RLS}^{(k)} = \left(\frac{(1-\beta) r}{2\tilde{\lambda}^{(k)} - (1-\beta) r} + 1 \right) \sigma_\nu^2, \tag{39}$$

where r is the dimension of $\mathbf{x}_t$. Hence, in this case the overall MSE will be

$$\eta_{RLS} = \left(\frac{(1-\lambda) m}{2 - (1-\lambda) m} + 1 \right) \sigma_\omega^2. \tag{40}$$

VI. EXPERIMENTS

In this section, we demonstrate the efficacy of the proposed boosting algorithms for RLS and LMS piecewise linear filters under different scenarios. To this end, we first consider the "regression" of a signal generated with a piecewise linear model, under stationary conditions. Then, we illustrate the performance of our algorithms under nonstationary conditions, to thoroughly test the adaptation capabilities of the proposed boosting framework. Furthermore, we investigate the effect of the number of the constituent filters as well as the "dependence parameter" c, on the final MSE performance. We also compare the computational time used by each algorithm, to show the advantage of "random updates" boosting method, over other methods.

Throughout this section, "PLMS" represents the piecewise linear LMS-based filter, "SPLMS" represents the piecewise linear LMS-based with soft partition (explained in Section IV, Remark 3), "BPLMS" represents the boosted piecewise linear LMS-based filter, and "BSPLMS" represents the boosted piecewise linear LMS-based with soft partition. Similarly, "PRLS" represents the piecewise linear RLS-based filter, and "BPRLS" represents the boosted piecewise linear RLS-based filter. In addition, we use the suffixes "-WU", "-RU", or "-DR" to denote the "weighted updates", "random updates", or "data reuse" modes, respectively. Also, "LMS-MIX" and "RLS-MIX" denote the conventional LMS-based and RLS-based mixture methods, which are a special case of our methods, i.e., the case $c = 0$.

In our simulations, we have set the step sizes for the LMS filters to 0.02 in all algorithms, except the SPLMS. For SPLMS, the step size for updating the filter coefficients is set to 0.1, while the step size for the regions boundaries update is set to 0.5. In addition, for boosted RLS filters we have set $\beta = 0.99$ for weighted updates and random updates, and set $\beta = 0.995$ for data reuse updates. In all of the RLS-based algorithms, we have set $\lambda = 0.999$, which is the forgetting factor of PRLS, and the forgetting factor for updating the combination weights in boosting algorithms. Moreover, we have chosen σ^2 as the desired mse parameter, $K = 2$ for data reuse approach, $c = 1$ for all boosting algorithms, and $m = 5$ as the number of constituent filters, in all experiments, except the experiments through which we have investigated the effects of these parameters.

For the performance comparison, we compare the performance of our methods with that of the best constituent filter. To this end, we have provided the Accumulated Square Error (ASE) results as well as the relative improvements in the values of the ASE, which is defined as $\varrho_{alg} \triangleq 100 \times \dfrac{ASE_0 - ASE_{alg}}{ASE_0}$, where, ASE_0 denotes the ASE of the single piecewise linear filter that is to be boosted, e.g., ASE of PLMS, and ASE_{alg} indicates the underlying algorithm, e.g., BPLMS. All of the results have been averaged over 30 rounds.

A. Stationary Data

In this experiment, we have considered the case where the desired data is generated by a piecewise linear model with 3 regions. The input vectors $\mathbf{x}_t = [x_1\ x_2\ 1]$ are 3-dimensional, and $[x_1\ x_2]$ is drawn from a jointly Gaussian random process, and then scaled such that $\mathbf{x}_t = [x_1\ x_2]^T \in [0\ 1]^2$. We have

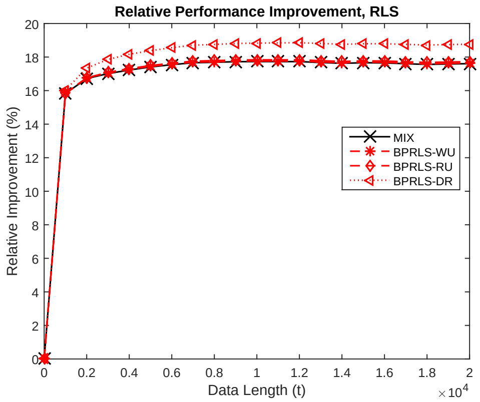


Fig. 5: The relative improvement in ASE performnce of the RLS-based algorithms in the stationary data experiment.

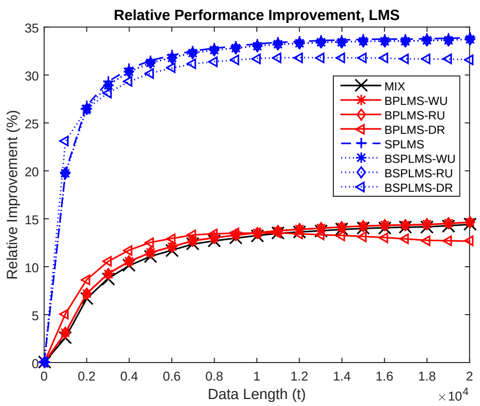


Fig. 6: The relative improvement in ASE performnce of the RLS-based algorithms in the stationary data experiment.

included 1 as the third entry of $\mathbf{x}_t$ to consider affine filters. Specifically the desired data is generated by the following model

$$d_t = \begin{cases} [1\ 1]\mathbf{x}_t + \nu_t & \text{if } \Phi_0^T \mathbf{x}_t < 0.5 \\ 0.5 + \nu_t & \text{if } 0.5 \leq \Phi_0^T \mathbf{x}_t < 0.8 \\ [1\ \ -0.5]\mathbf{x}_t + \nu_t & \text{if } \Phi_0^T \mathbf{x}_t \geq 0.8 \end{cases} \quad (41)$$

where $\Phi_0 = [1\ 1]^T$, and ν_t is a random Gaussian noise. We have used 5 piecewise linear filters, each with 2 regions, as the constituent filters in boosting. The filters regions are different, and for the linear filter in each region, the initial values are set to zero.

As depicted in Fig. 5 and Fig. 6 our proposed methods, significantly boost the performance of a single piecewise linear filter, in both LMS and RLS cases. Note that, the random updates method achieves the performance of the weighted updates method and the conventional mixture method, with a much lower complexity. The simulations show that we can perform better than the conventional mixture method using the data reuse boosting approach. However, since the complexity of data reuse updates is considerably higher than other approaches, one may prefer to use the other boosting approaches. Specifically, as shown in Fig. 5 and Fig. 6, the random update method, provides a satisfactory performance, while its complexity is much lower than other approaches.

B. Nontationary Data

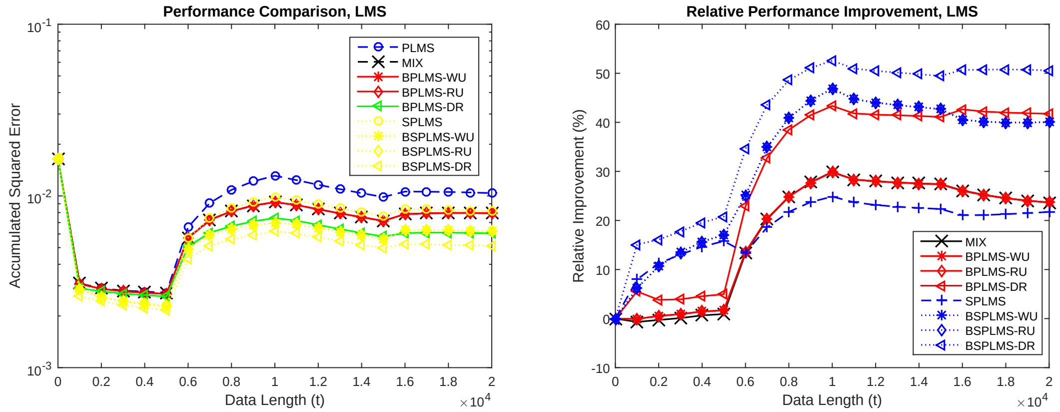
Here, we have considered the case where the desired data is generated by a nonstationary piecewise linear model with 3 regions, like the stationary data in the previous section. Again, $\mathbf{x}_t = [x_1\ x_2]^T \in [0\ 1]^2$ is randomly generated as in the stationary data experiment. However, in this 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 learning rates for the LMS-based algorithms are set to 0.02, the forgetting factor β for the RLS-based algorithms are set to 0.999, and the desired mse parameter σ^2 is set to 0.01. Also, the direction vector for the separating hyperplane is set to $\boldsymbol{\theta} = [\theta_1\ \theta_2\ \ -\theta_3]^T$, which is used as the initial direction vector in "SPLMS" and "BSPLMS" cases as well. $\boldsymbol{\theta}$ is consisted of three random variables, each with mean 1, to construct random constituent filters. The figures 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. 7b, the data reuse method shows a better performance relative to the other boosting methods. However, from the Fig. 8, the random updates method has a significantly lower time consumption, which makes it desirable for big data applications. We performed the simulations with RLS-based filters, and got good results, which shows that our methods can perform well with RLS updates too.

From the Fig. 9, we observe the approximate changes of the weights, in a BPRLS algorithm run over the nonstationary data. As shown in this figure, the weights do not change monotonically, and this shows the capability of our algorithm in effective tracking of the nonstationary data. Furthermore, since we update the filters in an ordered manner, i.e., we update the filter $k+1$ after the filter k is updated, the weights assigned to the last filters are generally smaller than the weights assigned to the previous filters. As an example, in Fig. 9 we see that the weights assigned to the second constituent filter are larger than those of the third and fourth filters. Moreover, note that in this experiment, we have set $c = 1$ as the dependency parameter. We should mention that increasing this parameter, in general, causes the lower weights, hence, it can considerably reduce the complexity of random updates and make it a useful method for big data processing tasks.

C. The Effect of Parameters

In this section, we investigate the effects of the dependence parameter c as well as the number of constituent filters, on the boosting performance of our methods in the nonstationary data experiment, explained in Section VI-B. From the results in Fig. 10a and Fig. 10b, we observe that, increasing the number of constituent filters can improve the performance significantly. However, as shown in Fig. 10c and 10d, in this experiment, for the short-length data, increasing the dependency parameter cannot improve the performance. Nevertheless, in this experiment, as the data length increases, we can get a better performance by using a larger value for dependency parameter.



(a) ASE performance **(b)** relative performance improvement

Fig. 7: The performance results of the boosting approaches in the nonstationary experiment.

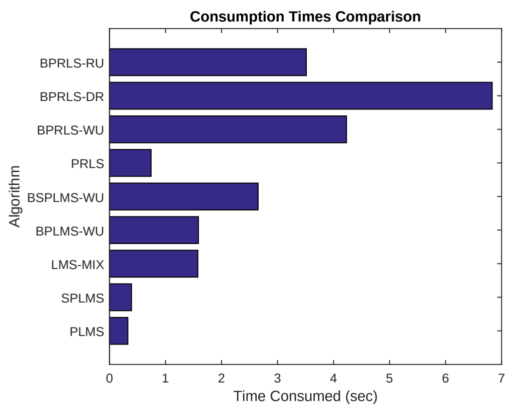


Fig. 8: The time consumed by each algorithm in the nonstationary data experiment.

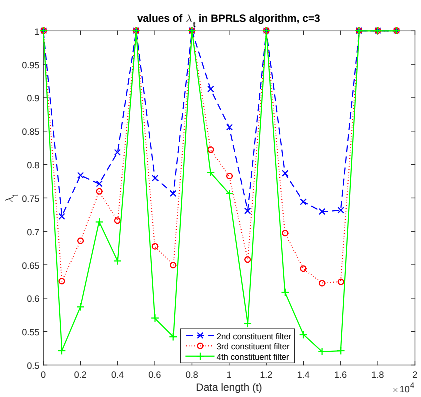


Fig. 9: The changing of the weights in BPRLS-WU algorithm in the nonstationary data experiment. All experiments have been done on the same computer.

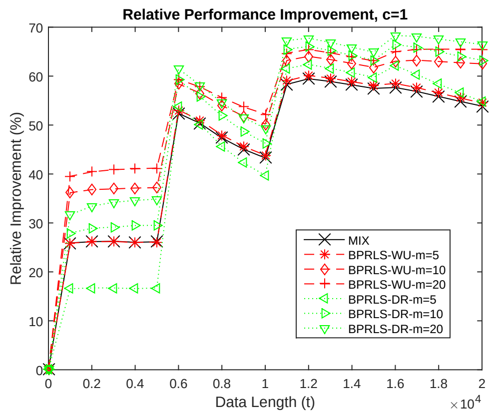
D. Benchmark Real and Synthetic Data Sets

In this section, we demonstrate the efficiency of the introduced methods over some widely used real and synthetic regression data sets. The data sets we have used can be found in [28]. We have normalized the data to the interval $[-1, 1]$ in all algorithms. 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 now describe the experiments and provide the results:

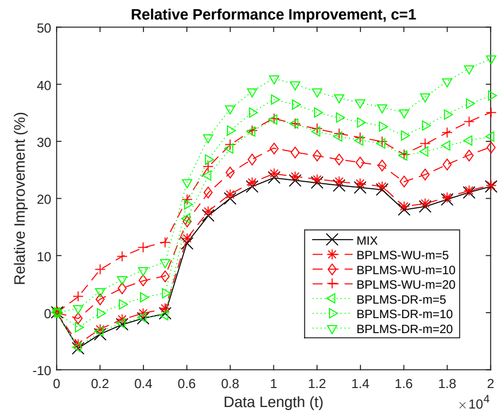
1) California Housing: This data set has been obtained from StatLib repository. They have collected information on the variables using all the block groups in California from the 1990 Census. We are going to find the house median values, based on the given attributes. For further description one can refer to [28]. In this experiment we have used the same parameters as the nonstationary experiment, for the filters. The results in Fig. 11 shows that our methods can perform well on this data set.
2) Computer Activity (CompAct): This real data set is a collection of computer systems activity measures [28]. The task is to predict usr, the portion of time that cpus run in user mode from all attributes [28]. In our simulation for this data set, we have set $K = 2$, $m = 5$, and $c = 1$. Also, for all of LMS based algorithms we have used 0.03 as the step size for linear filter coefficients updates, except for data reuse scenarios. For BPLMS-DR we have used a step size of 0.01, and for BSPLMS-DR we have used a step size of 0.05. The results shown in Fig. 12 and Fig. 13 indicate the superior performance of our algorithms.
3) Bank: This data set is generated from a simulation of how bank-customers choose their banks [28]. The algorithm should predict the fraction of bank customers who leave the bank because of full queues. We used the variant with m=32 (This is related to the data set, and one should not confuse this m with the number of constituent filters.) [28]. We have used the same parameters as the nonstationary data experiment described in Section VI-B for this experiment. The results are provided in Fig. 14.

VII. Conclusion

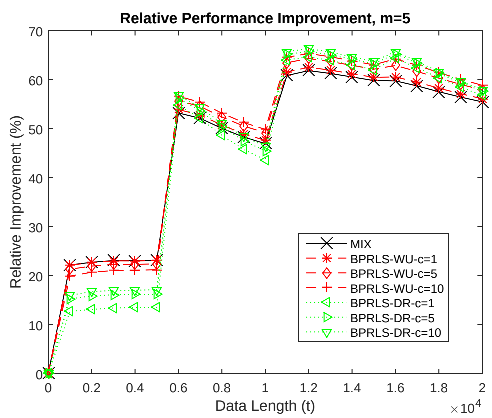
We introduce the boosting concept, extensively studied in machine learning literature, to adaptive filtering context, and propose three different boosting approaches, "weight updates","data reuse", and "random updates" which are applicable to different adaptive filtering algorithms. We show that



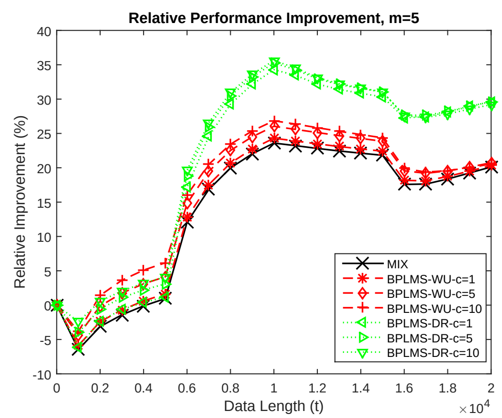
(a) The effect of the parameter m



(b) The effect of the parameter m



(c) The effect of the parameter c



(d) The effect of the parameter c

Fig. 10: The effect of the parameters m and c on the relative performance improvement of the RLS and LMS-based algorithms in the nonstationary data experiment.

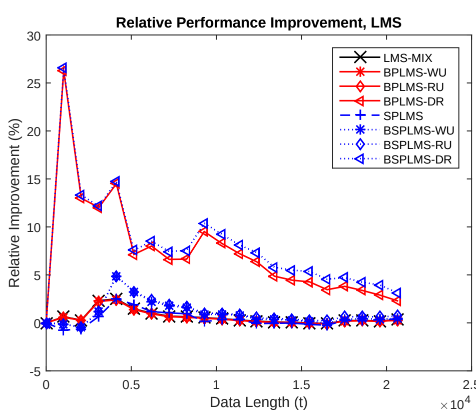


Fig. 11: The performance of LMS-based boosting methods on the California Housing data set.

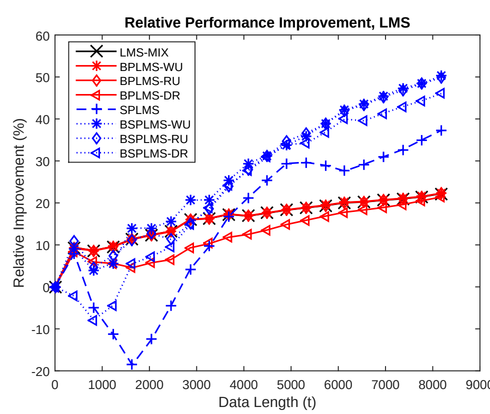


Fig. 12: The performance of LMS-based boosting methods on the CompAct data set.

by these approaches we can significantly improve the MSE performance of the conventional LMS and RLS algorithms 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. Therefore, the boosting using random updates approach can be applied to real life large scale problems, which boosts the MSE performance with only a slight increase in the complexity.

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.

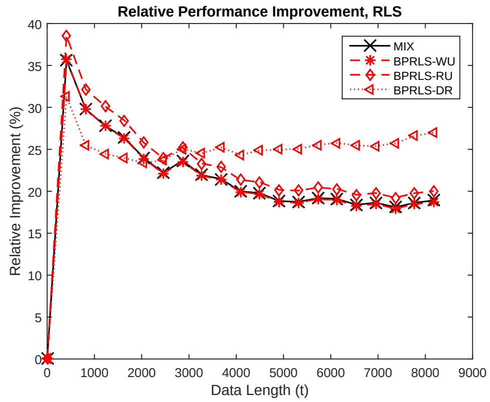


Fig. 13: The performance of RLS-based boosting methods on the CompAct data set.

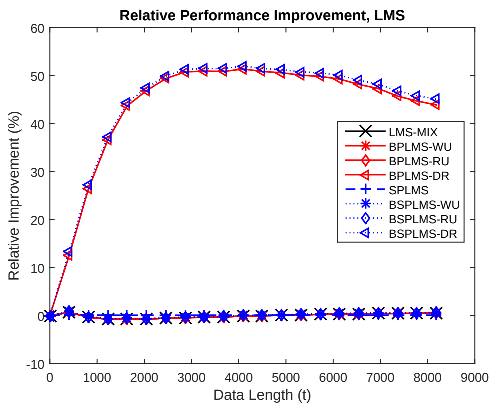


Fig. 14: The performance of LMS-based boosting methods on the Bank data set.

[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] B. Taieb and R. J. Hyndman, "Boosting multi-step autoregressive forecasts," in *ICML*, 2014.

[6] B. Taieb and R. J. Hyndman, "A gradient boosting approach to the kaggle load forecasting competition," *International Journal of Forecasting*, pp. 1–19, 2013.

[7] N. Shafik and G. Tutz, "Boosting nonlinear additive autoregressive time series," in *Computational Statistics and Data Analysis*, 2009.

[8] 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.

[9] 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.

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

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

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

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

[14] 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.

[15] A. C. Singer, G. W. Wornell, and A. V. Oppenheim, "Nonlinear autoregressive modeling and estimation in the presence of noise," *Digital Signal Processing*, vol. 4, no. 4, pp. 207–221, 1994.

[16] V. H. Nascimento and A. H. Sayed, "On the learning mechanism of adaptive filters," *IEEE Transactions on Signal Processing*, vol. 48, no. 6, pp. 1609–1625, 2000.

[17] T. Y. Al-Naffouri and A. H. Sayed, "Transient analysis of adaptive filters with error nonlinearities," *IEEE Transactions on Signal Processing*, vol. 51, no. 3, pp. 653–663, 2003.

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

[19] A. C. Singer, S. S. Kozat, and M. Feder, "Universal linear least squares prediction:upper and lower bounds," *IEEE Transactions on Information Theory*, vol. 48, no. 8, pp. 2354–2362, 2002.

[20] N. Cesa-Bianchi and G. Lugosi, *Prediction, Learning, and Games*, Cambridge University Press, 2006.

[21] 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.

[22] 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.

[23] N. Merhav and M. Feder, "Universal schemes for sequential decision from individual data sequences," *IEEE Trans. Inform. Theory*, vol. 39, pp. 1280–1291, 1993.

[24] Shai Shalev-Shwartz, "Online learning and online convex optimization," *Foundations and Trends in Machine Learning*, vol. 4, pp. 107–194, 2012.

[25] K. S. Azoury and M. K. Warmuth, "Relative loss bounds for on-line density estimation with the exponential family of distributions," *Machine Learning*, vol. 43, pp. 211–246, 2001.

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

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

[28] L. Torgo, "Regression data sets," [Online] Available at http://www.dcc.fc.up.pt/ ltorgo/Regression/DataSets.html.