BOOSTED ADAPTIVE FILTERS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR
THE DEGREE OF
MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

By
Dariush Kari
July 2017

BOOSTED ADAPTIVE FILTERS
By Dariush Kari
July 2017

We certify that we have read this thesis and that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Süleyman Serdar Kozat(Advisor)

Sinan Gezici

Sevinç Figen Öktem

Approved for the Graduate School of Engineering and Science:

Ezhan Karaşan
Director of the Graduate School

ABSTRACT

BOOSTED ADAPTIVE FILTERS

Dariush Kari
M.S. in Electrical and Electronics Engineering
Advisor: Süleyman Serdar Kozat
July 2017

We investigate boosted online regression and propose a novel family of regression algorithms with strong theoretical bounds. In addition, we implement several variants of the proposed generic algorithm. We specifically provide theoretical bounds for the performance of our proposed algorithms that hold in a strong mathematical sense. We achieve guaranteed performance improvement over the conventional online regression methods without any statistical assumptions on the desired data or feature vectors. We demonstrate an intrinsic relationship, in terms of boosting, between the adaptive mixture-of-experts and data reuse algorithms. Furthermore, we introduce a boosting algorithm based on random updates that is significantly faster than the conventional boosting methods and other variants of our proposed algorithms while achieving an enhanced performance gain. Hence, the random updates method is specifically applicable to the fast and high dimensional streaming data. Specifically, we investigate Recursive Least Squares (RLS)-based and Least Mean Squares (LMS)-based linear regression algorithms in a mixture-of-experts setting, and provide several variants of these well known adaptation methods. Moreover, we extend the proposed algorithms to other filters. Specifically, we investigate the effect of the proposed algorithms on piecewise linear filters. Furthermore, we provide theoretical bounds for the computational complexity of our proposed algorithms. We demonstrate substantial performance gains in terms of mean square error over the constituent filters through an extensive set of benchmark real data sets and simulated examples.

Keywords: Online boosting, online regression, boosted regression, ensemble learning, smooth boost, mixture methods.

ÖZET

İYİLEŞTİRİLMİŞ UYARLANIR SÜZGEÇLER

Dariush Kari
Elektrik ve Elektronik Mühendisliği, Yüksek Lisans
Tez Danışmanı: Süleyman Serdar Kozat
Temmuz 2017

İyileştirilmiş çevrimiçi regresyonu araştırıyoruz ve güçlü teorik sınırları olan yeni bir regresyon algoritma ailesi önermekteyiz. Buna ek olarak, önerilen genel algoritmanın çeşitli türlerini uyguluyoruz. Özellikle, önerilen algoritmalarımızın performansı için matematiksel anlamda sağlanan güçlü teorik sınırlar sağlarız. Veri veya öznitelik vektörleri üzerinde herhangi bir istatistiksel varsayım yapmadan geleneksel çevrimiçi geri regresyon yöntemlerine göre performans iyileşmesini garanti ediyoruz. Uzmanların uyarlamalı karışımı ile veriyi yeniden kullanma algoritmaları arasında, iyileştirme açısından içsel bir ilişki olduğunu gösteriyoruz. Ayrıca, geliştirilmiş bir performans kazancı elde ederken, geleneksel iyileştirme yöntemleri ve önerilen algoritmalarımızın diğer türlerinden daha hızlı olan rastgele güncellemelere dayanan bir iyileştirme algoritması sunuyoruz. Dolayısıyla, rastgele güncelleme yöntemi, özellikle hızlı ve yüksek boyutlu sürekli akan veriye uygulanabilir. Özellikle, uzman karışımı bağlamında Özyinelemeli En Küçük Kareler (RLS) tabanlı ve En Az Ortalama Kareler (LMS) tabanlı doğrusal regresyon algoritmalarını araştırıyor ve bu iyi bilinen uyarlama yöntemlerinin çeşitli türlerini sunuyoruz. Ayrıca, önerilen algoritmaları diğer süzgeçlere de genişletiriz. Özellikle, önerilen algoritmaların parçalı doğrusal süzgeçler üzerindeki etkisini araştırıyoruz. Ayrıca, önerilen algoritmalarımızın hesaplama karmaşıklığı için teorik sınırlar sağlarız. Oluşturulan süzgeçler üzerinde ortalama karesel hata açısından önemli performans artışını kapsamlı gerçek veri setleri ve temsili örnekler vasıtasıyla gösteriyoruz.

Anahtar sözcükler: Online güçlendirme algoritmaları, online bağlanım, güçlendirilmiş bağlanım, toplu öğrenim, düzgün güçlendirme, karışm metodları.

Acknowledgement

I would like to express my sincere appreciation to Assoc. Prof. Suleyman Serdar Kozat for his wise supervision, endless support, encouragement and being a role model for success. I could not have imagined a better advisor for my M.S. studies. I learned to be professional and productive thanks to the work ethics in Assoc. Prof. Kozat's team.

I would like to state my deep gratitude to Assoc. Prof. Sinan Gezici and Assist. Prof. Sevinç Figen Öktem for allocating their time to investigate my work and providing me with invaluable comments to make this thesis stronger.

Also, I would like to thank all of my mentors in Bilkent University, especially, Prof. Tolga Mete Duman, Assoc. Prof. Sinan Gezici, and Prof. Orhan Arikan, for their invaluable guidance and support during my master studies.

Last but not least, I would like to dedicate this thesis to the unconditional love and support of my family, my mother, father, brother, sisters, and brother in law, who had to bear with my rare visits. I could not have imagined a better upbringing if they were not always there for me.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Boosting is considered as one of the most important ensemble learning methods in the machine learning literature and it is extensively used in several different real life applications from classification to regression [1, 2, 3, 4, 5, 6, 7, 8]. As an ensemble learning method [9, 10, 11, 12, 13, 14, 15], boosting combines several parallel running "weakly" performing algorithms to build a final "strongly" performing algorithm [16, 17, 18]. 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 commonly using a functional gradient descent [19, 20]. Boosting is successfully applied to several different problems in the machine learning literature including classification [1, 20, 21], regression [19, 21, 22], and prediction [23, 24]. However, significantly less attention is given to the idea of boosting in online regression framework. To this end, our goal is (a) to introduce a new boosting approach for online regression, (b) derive several different online regression algorithms based on the boosting approach, (c) provide mathematical guarantees for the performance improvements of our algorithms, and (d) demonstrate the intrinsic connections of boosting with the adaptive mixture-of-experts algorithms [25, 26] and data reuse algorithms [27].

Although boosting is initially introduced in the batch setting [20], where algorithms boost themselves over a fixed set of training data, it is later extended to the

online setting [28, 29]. In the online setting, however, we neither need nor have access to a fixed set of training data, since the data samples arrive one by one as a stream [30, 14]. Each newly arriving data sample 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 may not be enough storage space available or the constraints of the problem require instant processing [31]. Therefore, we concentrate on the online boosting framework and propose several algorithms for online regression tasks. In addition, since our algorithms are online, they can be directly used in adaptive filtering applications to improve the performance of conventional mixture-of-experts methods [25]. For adaptive filtering purposes, the online setting is especially important, where the sequentially arriving data is used to adjust the internal parameters of the filter, either to dynamically learn the underlying model or to track the nonstationary data statistics [25, 32].

Specifically, we have m parallel running constituent filters (CF) [17] that receive the input vectors sequentially. Each CF uses an update method, such as the Recursive Least Squares (RLS) or Least Mean Squares (LMS), depending on the target of the applications or problem constraints [32]. 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 update 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 of the m CFs 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 CF to the last one to achieve the "boosting" effect [33]. Furthermore, unlike the usual mixture approaches [25, 26], the update of each CF depends on the previous CFs in the mixture. In particular, at each time t, after the k^{th} CF calculates its error over $(\boldsymbol{x}_t, d_t)$ pair, it passes a certain weight to the next CF, the $(k+1)^{th}$ CF, quantifying how much error the constituent CFs from 1^{st} to k^{th} made on the current $(\boldsymbol{x}_t, d_t)$ pair. Based on the performance of the CFs from 1 to k on the current $(\boldsymbol{x}_t, d_t)$ pair, the $(k+1)^{th}$ CF may give a

different emphasis (importance weight) to $(\boldsymbol{x}_t, d_t)$ pair in its adaptation in order to rectify the mistake of the previous CFs.

The proposed idea for online boosting is clearly related to the adaptive mixture-of-experts algorithms widely used in the machine learning literature, where several parallel running adaptive algorithms are combined to improve the performance [34]. In the mixture methods, the performance improvement is achieved due to the diversity provided by using several different adaptive algorithms each having a different view or advantage [26]. 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 [26], there are significant differences. In the online boosting notion, the parallel running algorithms are not independent, i.e., one deliberately introduces the diversity by updating the CFs one by one from the first CF to the m^{th} CF for each new sample based on the performance of all the previous CFs on this sample. In this sense, each adaptive algorithm, say the $(k+1)^{th}$ CF, receives feedback from the previous CFs, i.e., 1^{st} to k^{th}, and updates its inner parameters accordingly. As an example, if the current $(\boldsymbol{x}_t, d_t)$ is well modeled by the previous CFs, then the $(k+1)^{th}$ CF performs minor update using $(\boldsymbol{x}_t, d_t)$ and may give more emphasis (importance weight) to the later arriving samples that may be worse modeled by the previous CFs. Thus, by boosting, each adaptive algorithm in the mixture can concentrate on different parts of the input and output pairs achieving diversity and significantly improving the gain.

The linear online learning algorithms, such as LMS or RLS, are among the simplest as well as the most widely used regression algorithms in the real-life applications [32]. Therefore, we use such algorithms as base CFs in our boosting algorithms. To this end, we first apply the boosting notion to several parallel running linear RLS-based CFs and introduce three different approaches to use the importance weights [33], namely "weighted updates","data reuse", and "random updates". 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 ([29]). However, data reuse in boosting, such as [29], is significantly different from the usual data reusing

approaches in adaptive filtering [27]. As an example, in boosting, the importance weight coming from the k^{th} CF determines the data reuse amount in the $(k+1)^{th}$ CF, i.e., it is not used for the k^{th} filter, hence, achieving the diversity. The third approach uses the importance weights to decide whether to update the constituent CFs or not, 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 [35] due to the reduced complexity. The output of the constituent CFs is also combined using a linear mixture algorithm to construct the final output. We then update the final combination algorithm using the LMS algorithm [26]. Furthermore, we extend the boosting idea to parallel running linear LMS-based algorithm similar to the RLS case.

Note that although linear filters have a low complexity, piecewise linear filters deliver a significantly superior performance in real life applications [36, 37], with a comparable complexity. These filters mitigate the overfitting, stability and convergence issues tied to nonlinear models [38, 39, 40], while effectively improving the modeling power relative to linear filters [36]. Nevertheless, in order to justify the boosting effect of our algorithm, we use linear base learners with exactly the same parameters and demonstrate that even in this case we can get performance improvement by our algorithm since any gain obtained in this way reflects the sole effect of the boosting mechanism. We then extend our algorithms to piecewise linear filters.

We start our discussions by investigating the related works in Section 1.1. We then introduce the problem setup and background in Chapter 2, where we provide individual sequence as well as MSE convergence results for the RLS and LMS algorithms. We introduce our generic boosted online regression algorithm in Chapter 3 and provide the mathematical justifications for its performance. Then, in Sections 4.1 and 4.2 of the Chapter 4, three different variants of the proposed boosting algorithm are derived, using the RLS and LMS, respectively. Also, we proceed to investigate the proposed boosting approach for piecewise linear adaptive filters in Chapter 5. Then, in Chapter 6 we provide the mathematical analysis for the computational complexity of the proposed algorithms. The thesis concludes with extensive sets of experiments over the well-known benchmark data

sets and simulation models widely used in the machine learning literature to demonstrate the significant gains achieved by the boosting notion.

1.1 Related Works

AdaBoost is one of the earliest and most popular boosting methods, which has been used for binary and multiclass classifications as well as regression [20]. This algorithm has been well studied and has clear theoretical guarantees, and its excellent performance is explained rigorously [41]. However, AdaBoost cannot perform well on the noisy data sets [42], therefore, other boosting methods have been suggested that are more robust against noise.

In order to reduce the effect of noise, SmoothBoost was introduced in [42] in a batch setting. Moreover, in [42], the author proves the termination time of the SmoothBoost algorithm by simultaneously obtaining upper and lower bounds on the weighted advantage of all samples over all of the constituent filters. We note that the SmoothBoost algorithm avoids overemphasizing the noisy samples, hence, provides robustness against noise. In [29], the authors extend bagging and boosting methods to an online setting, where they use a Poisson sampling process to approximate the reweighting algorithm. However, the online boosting method in [29] corresponds to AdaBoost, which is susceptible to noise. In [43], the authors use a greedy optimization approach to develop the boosting notion to the online setting and introduce stochastic boosting. Nevertheless, while most of the online boosting algorithms in the literature seek to approximate AdaBoost, [33] investigates the inherent difference between batch and online learning, extend the SmoothBoost algorithm to an online setting, and provide the mathematical guarantees for their algorithm. [33] points out that the online constituent filters do not need to perform well on all possible distributions of data, instead, they have to perform well only with respect to smoother distributions. Recently, in [44], the authors have developed two online boosting algorithms for classification, an optimal algorithm in terms of the number of constituent filters, and also an adaptive algorithm using the potential functions and boost-by-majority [45].

In addition to the classification task, the boosting approach has also been developed for the regression [19]. In [46], a boosting algorithm for regression is proposed, which is an extension of Adaboost.R [46]. Moreover, in [19], several gradient descent algorithms are presented, and some bounds on their performances are provided. In [43], the authors present a family of boosting algorithms for online regression through greedy minimization of a loss function. Also, in [47] the authors propose an online gradient boosting algorithm for regression.

In this thesis we propose a novel family of boosted online algorithms for the regression task using the "online boosting" notion introduced in [33], and investigate three different variants of the introduced algorithm. Furthermore, we show that our algorithm can achieve a desired mean squared error (MSE), given a sufficient amount of data and a sufficient number of constituent filters. In addition, we use similar techniques to [42] to prove the correctness of our algorithm. We emphasize that our algorithm has a guaranteed performance in an individual sequence manner, i.e., without any statistical assumptions on the data. In establishing our algorithm and its justifications, we refrain from changing the regression problem to the classification problem, unlike the AdaBoost.R [20]. Furthermore, unlike the online SmoothBoost [33], our algorithm can learn the guaranteed MSE of the constituent filters, which in turn improves its adaptivity.

Chapter 2

Problem Description and Background

All vectors are column vectors and represented by bold lower case letters. Matrices are represented by bold upper case letters. For a vector $\boldsymbol{a}$ (or a matrix $\boldsymbol{A}$), $\boldsymbol{a}^T$ (or $\boldsymbol{A}^T$) is the transpose and $\mathrm{Tr}(\boldsymbol{A})$ is the trace of the matrix $\boldsymbol{A}$. Here, $\boldsymbol{I}_m$ and $\boldsymbol{0}_m$ represent the identity matrix of dimension $m \times m$ and the all zeros vector of length m, respectively. Except $\boldsymbol{I}_m$ and $\boldsymbol{0}_m$, 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 $E[x]$. Also, we show the cardinality of a set S by $|S|$.

We sequentially receive r-dimensional input (regressor) vectors $\{\boldsymbol{x}_t\}_{t\geq 1}$, $\boldsymbol{x}_t \in \mathbb{R}^r$, and desired data $\{d_t\}_{t\geq 1}$, and estimate d_t by $\hat{d}_t = f_t(\boldsymbol{x}_t)$, where $f_t(.)$ is an online regression algorithm. 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 CF. For presentation purposes, we assume that $d_t \in [-1, 1]$, however, our derivations hold for any bounded but arbitrary desired data sequences. In our framework, we do not use any statistical assumptions on the input feature vectors or on the desired data such that our results are guaranteed to hold in an individual sequence manner [48].

The linear methods are considered as the simplest online modeling or learning algorithms, which estimate the desired data d_t by a linear model as $\hat{d}_t = \boldsymbol{w}_t^T \boldsymbol{x}_t$, where $\boldsymbol{w}_t$ is the linear algorithm's coefficients at time t. Note that the previous expression also covers the affine model if one includes a constant term in $\boldsymbol{x}_t$, hence we use the purely linear form for notational simplicity. When the true d_t is revealed, the algorithm updates its coefficients $\boldsymbol{w}_t$ based on the error e_t. As an example, in the basic implementation of the RLS algorithm, the coefficients are selected to minimize the accumulated squared regression error up to time $t-1$ as

$$\begin{aligned} \boldsymbol{w}_t &= \arg\min_{\boldsymbol{w}} \sum_{l=1}^{t-1} (d_l - \boldsymbol{x}_l^T \boldsymbol{w})^2, \\ &= \left(\sum_{l=1}^{t-1} \boldsymbol{x}_l \boldsymbol{x}_l^T \right)^{-1} \left(\sum_{l=1}^{t-1} \boldsymbol{x}_l d_l \right), \end{aligned} \tag{2.1}$$

where $\boldsymbol{w}$ is a fixed vector of coefficients. The RLS algorithm is shown to enjoy several optimality properties under different statistical settings [32]. Apart from these results and more related to the framework of this thesis, the RLS algorithm is also shown to be rate optimal in an individual sequence manner [49]. As shown in [49] (Section V), when applied to any sequence $\{\boldsymbol{x}_t\}_{t\geq 1}$ 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 least squares (LS) method that is directly optimized for these realizations of the sequences, i.e., for all T, $\{\boldsymbol{x}_t\}_{t\geq 1}$ and $\{d_t\}_{t\geq 1}$, the RLS achieves

$$\sum_{l=1}^{T} (d_l - \boldsymbol{x}_l^T \boldsymbol{w}_l)^2 - \min_{\boldsymbol{w}} \sum_{l=1}^{T} (d_l - \boldsymbol{x}_l^T \boldsymbol{w})^2 \leq O(\ln T). \tag{2.2}$$

The RLS algorithm is a member of the Follow-the-Leader type algorithms [50] (Section 3), where one uses the best performing linear model up to time $t-1$ to predict d_t. Hence, (2.2) follows by direct application of the online convex optimization results [51] after regularization. The convergence rate (or the rate of the regret) of the RLS algorithm is also shown to be optimal so that $O(\ln T)$ in the upper bound cannot be improved [52]. It is also shown in [52] that one can reach the optimal upper bound (with exact scaling terms) by using a slightly

modified version of (2.1)

$$\boldsymbol{w}_t = \left(\sum_{l=1}^{t} \boldsymbol{x}_l \boldsymbol{x}_l^T \right)^{-1} \left(\sum_{l=1}^{t-1} \boldsymbol{x}_l d_l \right). \tag{2.3}$$

Note that the extension (2.3) of (2.1) is a forward algorithm (Section 5 of [53]) and one can show that, in the scalar case, the predictions of (2.3) are always bounded (which is not the case for (2.1)) [52].

We emphasize that in the basic application of the RLS algorithm, all data pairs $(\boldsymbol{x}_l, d_l)$, $l = 1, \ldots, t$, receive the same "importance" or weight in (2.1). Although there exists exponentially weighted or windowed versions of the basic RLS algorithm [32], these methods weight (or concentrate on) the most recent samples for better modeling of the nonstationarity [32]. However, in the boosting framework [20], 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 a CF performs worse on a sample, the next CF concentrates more on this example to better rectify this mistake. In the following sections, we use this notion to derive different boosted online regression algorithms.

Although in this thesis, we use linear CFs for the sake of notational simplicity, one can readily extend our approach to nonlinear and piecewise linear regression methods. For example, one can use tree based online regression methods [54, 55] as the constituent filters, and boost them with the proposed approach.

Chapter 3

New Boosted Online Regression Algorithms

In this section we present the generic form of our proposed algorithms and provide the guaranteed performance bounds for that. Regarding the notion of "online boosting" introduced in [33], the online constituent filters need to perform well only over smooth distributions of data points. We first present the generic algorithm in Algorithm 1 and provide its theoretical justifications, then discuss about its structure and the intuition behind it.

In this algorithm, each constituent filter receives a sequence of data points $(\boldsymbol{x}_t, d_t)$ and a corresponding weight $0 \leq \lambda_t \leq 1$ for each point. Since $d_t \in [-1, 1]$, we define the Weighted MSE (WMSE) of a learning algorithm as $\frac{\sum_{t=1}^{T} \lambda_t (e_t)^2}{4 \sum_{t=1}^{T} \lambda_t}$, where $e_t = d_t - \hat{d}_t \in [-2, 2]$. In the following theorem, we show that if $\sum_{t=1}^{T} \lambda_t$ is large enough (the meaning of which become clear at the proof of Theorem 1), there exists an online constituent filter that achieves a specific (better than the trivial solution) WMSE.

Assumption: (H-strong convexity [56]) We use the e_t^2 as a measure of loss and assume that

$$\|\nabla e_t^2\| \leq G,$$

and

$$\|\nabla^2 e_t^2\| \geq H\ \boldsymbol{I}_n.$$

Theorem 1. *Suppose for any sequence of data points and corresponding weights λ_t, where $\lambda_1 = 1$, there is an offline algorithm with a WMSE of σ^2_{off}, i.e.,*

$$\frac{\sum_{t=1}^{T} \lambda_t (e_t^{\text{off}})^2}{4\sum_{t=1}^{T} \lambda_t} = \sigma^2_{\text{off}}$$

Moreover, assume that

$$\sum_{t=1}^{T} \lambda_t \geq \frac{G^2}{4\epsilon H \sigma^2_{\text{off}}}, \tag{3.1}$$

where ϵ is a positive number. Under the stated conditions, there exists an online algorithm with a WMSE of at most $\sigma^2 = (1+\epsilon)\sigma^2_{\text{off}}$.

Proof. According to [56], if we use online gradient descent algorithm with the step sizes η_t, we reach the following upper bound on the regret of the online algorithm with respect to the best offline algorithm (which uses the constant vector $\boldsymbol{w}^*$).

$$\sum_{t=1}^{T} \lambda_t \left(e_t^2(\boldsymbol{w}_t) - e_t^2(\boldsymbol{w}^*)\right) \leq \sum_{t=1}^{T} \lambda_t \|\boldsymbol{w}_t - \boldsymbol{w}^*\|^2 \left(\frac{1}{\eta_{t+1}} - \frac{1}{\eta_t} - H\right) + G^2 \sum_{t=1}^{T} \lambda_t \eta_{t+1}. \tag{3.2}$$

Also, by mathematical induction, it can be shown that if $0 \leq \lambda_t \leq 1$ and $\lambda_1 = 1$, we have

$$\sum_{t=1}^{T} \frac{\lambda_t}{\sum_{i=1}^{t} \lambda_i} \leq 1 + \ln \sum_{t=1}^{T} \lambda_t.$$

Hence, by choosing $\eta_{t+1} \triangleq \frac{1}{H\sum_{i=1}^{t} \lambda_i}$, it is straightforward to show that

$$\sum_{t=1}^{T} \lambda_t \left(e_t^2(\boldsymbol{w}_t) - e_t^2(\boldsymbol{w}^*)\right) \leq \frac{G^2}{H} \left(1 + \ln \sum_{t=1}^{T} \lambda_t\right). \tag{3.3}$$

Now, by dividing both sides by $4\sum_{t=1}^{T} \lambda_t$, and taking into account the Assumption in (3.1), we observe that

$$1 + \ln \sum_{t=1}^{T} \lambda_t \leq (\frac{4\epsilon H \sigma^2_{\text{off}}}{G^2}) \sum_{t=1}^{T} \lambda_t,$$

or equivalently,

$$\frac{G^2}{4H\sum_{t=1}^{T}\lambda_t}\left(1+\ln\sum_{t=1}^{T}\lambda_t\right)\leq\epsilon\sigma_{\text{off}}^2$$

This concludes the proof of Theorem 1.

Algorithm 1 Boosted online regression algorithm

1: Input: $(\boldsymbol{x}_t, d_t)$ (data stream), m (number of constituent filters running in parallel), σ_m^2 (the modified desired MSE), and σ^2 (the guaranteed achievable WMSE).
2: Initialize the regression coefficients $\boldsymbol{w}_1^{(k)}$ for each CF; and the combination coefficients as $\boldsymbol{z}_1=\frac{1}{m}[1,1,\ldots,1]^T$; $\lambda_1^{(k)}=1$;
3: **for** $t=1$ **to** T **do**
4: Receive the regressor data instance $\boldsymbol{x}_t$;
5: Compute the CFs outputs $\hat{d}_t^{(k)}$;
6: Produce the final estimate $\hat{d}_t=\boldsymbol{z}_t^T\boldsymbol{y}_t=\boldsymbol{z}_t^T[\hat{d}_t^{(1)},\ldots,\hat{d}_t^{(m)}]^T$;
7: Receive the true output d_t (desired data);
8: $\lambda_t^{(1)}=1$; $l_t^{(1)}=0$;
9: **for** $k=1$ **to** m **do**
10: $\lambda_t^{(k)}=\min\left\{1,(\sigma^2)^{\,l_t^{(k)}/2}\right\}$ (for $t\geq 2$);
11: Update the CF$^{(k)}$, such that it has a WMSE $\leq\sigma^2$;
12: $e_t^{(k)}=d_t-\hat{d}_t^{(k)}$;
13: $l_t^{(k+1)}=l_t^{(k)}+\left[\sigma_m^2-\left(e_t^{(k)}\right)^2\right]$;
14: **end for**
15: Update $\boldsymbol{z}_t$ based on $e_t=d_t-\boldsymbol{z}_t^T\boldsymbol{y}_t$;
16: **end for**

In Algorithm 1, we have m copies of an online CF, each of which is guaranteed to have a WMSE of at most σ^2. We prove that the Algorithm 1 can reach a desired MSE, σ_d^2, through Lemma 1, Lemma 2, and Theorem 2. Note that since we assume $d_t\in[-1,1]$, the trivial solution $\hat{d}_t=0$ incurs an MSE of at most 1. Therefore, we define a constituent filter as an algorithm which has an MSE less than 1, i.e., a WMSE less than 1/4.

Lemma 1. *In Algorithm 1, if there is an integer M such that $\sum_{t=1}^{T}\lambda_t^{(k)}\geq\kappa T$*

for every $k \leq M$, and also $\sum_{t=1}^{T} \lambda_t^{(M+1)} < \kappa T$, where $0 < \kappa < \sigma_d^2$ is arbitrarily chosen, it can reach a desired MSE, $\frac{\sum_{t=1}^{T} e_t^2}{T} \leq \sigma_d^2$.

Proof. The proof of Lemma 1 is given in A.1.

Lemma 2. *If the constituent filters are guaranteed to have a weighted MSE less than σ^2, i.e.,*

$$\forall k: \qquad \frac{\sum_{t=1}^{T} \lambda_t^{(k)} (e_t^{(k)})^2}{4 \sum_{t=1}^{T} \lambda_t^{(k)}} \leq \sigma^2 \leq \frac{1}{4},$$

there is an integer M that satisfies the conditions in Lemma 1.

Proof. The proof of Lemma 2 is given in A.2.

Theorem 2. *If the constituent filters in line 11 of Algorithm 1 achieve a weighted MSE of at most $\sigma^2 < \frac{1}{4}$, there exists an upper bound for m such that the algorithm reaches the desired MSE.*

Proof. This theorem is a direct consequence of combining Lemma 1 and Lemma 2.

Note that if $T \geq \frac{G^2}{4\kappa\epsilon H \sigma_{\text{off}}^2}$, then the Assumption (3.1) will be satisfied for all constituent filters, i.e., in order to boost the performance of the constituent filters using the Algorithm 1, we only need to have a sufficiently large number of data. Furthermore, although we are using copies of a base learner as the constituent filters and seek to improve its performance, the constituent CFs can be different. However, by using the boosting approach, we can improve the MSE performance of the overall system as long as the CFs can provide a weighted MSE of at most σ^2. For example, we can improve the performance of mixture-of-experts algorithms ([25]) by leveraging the boosting approach introduced in this thesis.

As shown in Fig. 3.1, at each iteration t, we have m parallel running CFs with estimating functions $f_t^{(k)}$, producing estimates $\hat{d}_t^{(k)} = f_t^{(k)}(\boldsymbol{x}_t)$ of d_t, $k = 1, \ldots, m$. As an example, if we use m "linear" algorithms, $\hat{d}_t^{(k)} = \boldsymbol{x}_t^T \boldsymbol{w}_t^{(k)}$ is the estimate generated by the k^{th} CF. The outputs of these m CFs are then combined using the linear weights $\boldsymbol{z}_t$ to produce the final estimate as $\hat{d}_t = \boldsymbol{z}_t^T \boldsymbol{y}_t$ [26], where $\boldsymbol{y}_t \triangleq [\hat{d}_t^{(1)}, \ldots, \hat{d}_t^{(m)}]^T$ is the vector of outputs. After the desired output d_t is revealed, the m parallel running CFs will be updated for the next iteration. Moreover,

the linear combination coefficients $\boldsymbol{z}_t$ are also updated using the normalized LMS [32], as detailed later in Section 3.1.

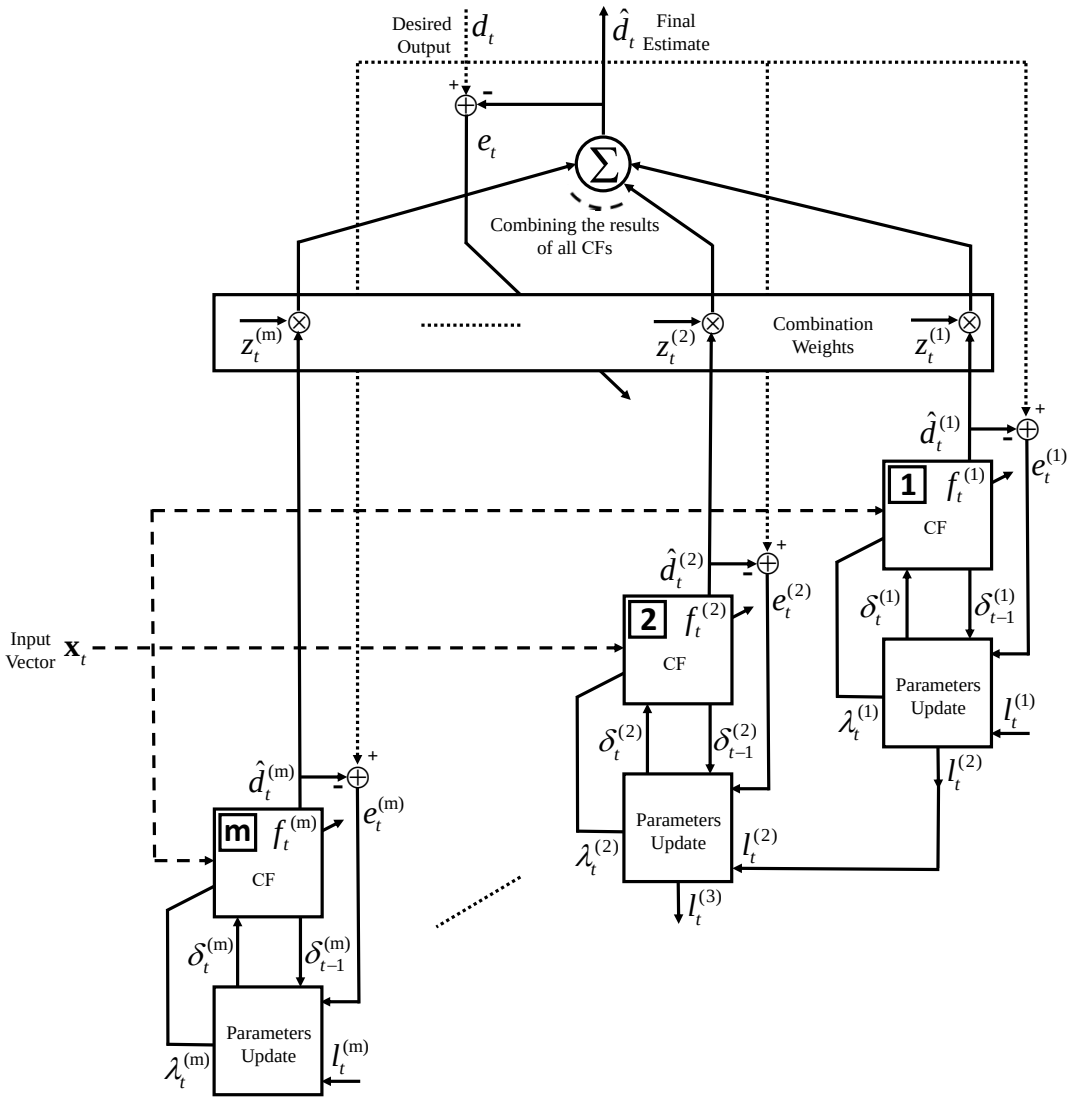


Figure 3.1: The block diagram of a boosted online regression system that uses the input vector $\boldsymbol{x}_t$ to produce the final estimate $\hat{d}_t$. There are m constituent CFs $f_t^{(1)}, \ldots, f_t^{(m)}$, each of which is an adaptive filter that generates its own estimate $\hat{d}_t^{(k)}$. The final estimate $\hat{d}_t$ is a linear combination of the estimates generated by all these CFs, 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 k^{th} CF 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 WMSE of the k^{th} CF over the first t estimations, and is used in computing $\lambda_t^{(k)}$.

After d_t is revealed, the constituent CFs, $f_t^{(k)}$, $k = 1, \ldots, m$, are consecutively

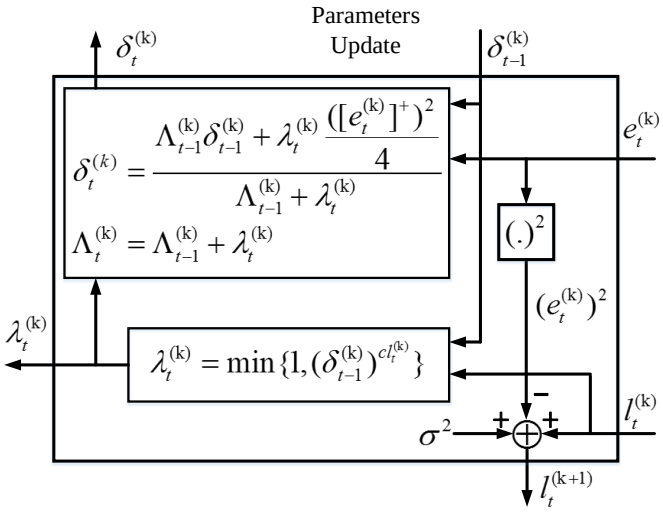


Figure 3.2: Parameters update block of the kth constituent filter, which is embedded in the kth filter block as depicted in Fig. 3.1. This block receives the parameter $l_t^{(k)}$ provided by the $(k-1)$th filter, and uses that in computing $\lambda_t^{(k)}$. It also computes $l_t^{(k+1)}$ and passes it to the $(k+1)$th filter. The parameter $[e_t^{(k)}]^+$ represents the error of the thresholded estimate as explained in (3.7), 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 $t-1$.

updated, as shown in Fig. 3.1, 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 ([20]), such that the $(k+1)^{th}$ CF receives a "total loss" parameter, $l_t^{(k+1)}$, from the k^{th} CF, as

$$l_t^{(k+1)} = l_t^{(k)} + \left[\sigma_m^2 - \left(d_t - f_t^{(k)}(\boldsymbol{x}_t)\right)^2\right], \tag{3.4}$$

to compute a weight $\lambda_t^{(k)}$. The total loss parameter $l_t^{(k)}$, indicates the sum of the differences between the modified desired MSE (σ_m^2) and the squared error of the first $k-1$ CFs at time t. Then, we add the difference $\sigma_m^2 - (e_t^{(k)})^2$ to $l_t^{(k)}$, to generate $l_t^{(k+1)}$, and pass $l_t^{(k+1)}$ to the next CF, as shown in Fig. 3.1. Here, $\left[\sigma_m^2 - \left(d_t - f_t^{(k)}(\boldsymbol{x}_t)\right)^2\right]$ measures how much the k^{th} CF is off with respect to the final MSE performance goal. For example, in a stationary environment, if $d_t = f(\boldsymbol{x}_t) + \nu_t$, where $f(\cdot)$ is a deterministic function and ν_t is the observation noise, one can select the desired MSE σ_d^2 as an upper bound on the variance of the noise process ν_t, and define a *modified* desired MSE as $\sigma_m^2 \triangleq \frac{\sigma_d^2 - 4\kappa}{1-\kappa}$. In this sense, $l_t^{(k)}$ measures how the CFs $j = 1, \ldots, k$ are cumulatively performing on

$(\boldsymbol{x}_t, d_t)$ pair with respect to the final performance goal.

We then use the weight $\lambda_t^{(k)}$ to update the k^{th} CF with the "weighted updates", "data reuse", or "random updates" method, which we explain later in Sections 4.1 and 4.2. Our aim is to make $\lambda_t^{(k)}$ large if the first $k-1$ CFs made large errors on d_t, so that the k^{th} CF gives more importance to $(\boldsymbol{x}_t, d_t)$ 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 [42, 33]. We define the weights as

$$\lambda_t^{(k)} = \min\left\{1, \left(\sigma^2\right)^{l_t^{(k)}/2}\right\}, \tag{3.5}$$

where σ^2 is the guaranteed upper bound on the WMSE of the constituent filters. However, since there is no prior information about the exact MSE performance of the constituent filters, we use the following weighting scheme

$$\lambda_t^{(k)} = \min\left\{1, \left(\delta_{t-1}^{(k)}\right)^{c\, l_t^{(k)}}\right\}, \tag{3.6}$$

where $\delta_{t-1}^{(k)}$ indicates an estimate of the k^{th} constituent filter's MSE, and $c \geq 0$ is a design parameter, which determines the "dependence" of each CF update on the performance of the previous CFs, i.e., $c = 0$ corresponds to "independent" updates, like the ordinary combination of the CFs in adaptive filtering [26, 25], while a greater c indicates the greater effect of the previous CFs performance on the weight $\lambda_t^{(k)}$ of the current CF. Note that including the parameter c does not change the validity of our proofs, since one can take $\left(\delta_{t-1}^{(k)}\right)^c$ as the new guaranteed WMSE. Here, $\delta_{t-1}^{(k)}$ is an estimate of the WMSE of the k^{th} CF over $\{\boldsymbol{x}_t\}_{t\geq 1}$ and $\{d_t\}_{t\geq 1}$. In the basic implementation of the online boosting [42, 33], $\left(1-\delta_{t-1}^{(k)}\right)$ is set to the classification advantage of the constituent filters [42], where this advantage is assumed to be the same for all constituent filters. In this thesis, 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 k^{th} CF up to time $t-1$

as

$$\delta_t^{(k)} = \frac{\sum_{\tau=1}^{t} \frac{\lambda_\tau^{(k)}}{4} \left(d_\tau - \left[f_\tau^{(k)}(\boldsymbol{x}_\tau) \right]^+ \right)^2}{\sum_{\tau=1}^{t} \lambda_\tau^{(k)}}$$
$$= \frac{\Lambda_{t-1}^{(k)} \delta_{t-1}^{(k)} + \frac{\lambda_t^{(k)}}{4} \left(d_t - \left[f_t^{(k)}(\boldsymbol{x}_t) \right]^+ \right)^2}{\Lambda_{t-1}^{(k)} + \lambda_t^{(k)}}, \tag{3.7}$$

where $\Lambda_t^{(k)} \triangleq \sum_{\tau=1}^{t} \lambda_\tau^{(k)}$, and $\left[f_\tau^{(k)}(\boldsymbol{x}_\tau) \right]^+$ thresholds $f_\tau^{(k)}(\boldsymbol{x}_\tau)$ 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 (3.7) can be recursively calculated.

Regarding the definition of $\lambda_t^{(k)}$, if the first k CFs are "good", we will pass less weight to the next CFs, such that those CFs can concentrate more on the other samples. Hence, the CFs can increase the diversity by concentrating on different parts of the data [26]. Furthermore, following this idea, in (3.6), the weight $\lambda_t^{(k)}$ is larger, i.e., close to 1, if most of the CFs, $1, \ldots, k-1$, have errors larger than σ_m^2 on $(\boldsymbol{x}_t, d_t)$, and smaller, i.e., close to 0, if the pair $(\boldsymbol{x}_t, d_t)$ is easily modeled by the previous CFs such that the CFs $k, \ldots, m$ do not need to concentrate more on this pair.

3.1 The Combination Algorithm

Although in the proof of our algorithm, we assume a constant combination vector $\boldsymbol{z}$ over time, we use a time varying combination vector in practice, since there is no knowledge about the exact number of the required week learners for each problem. Hence, after d_t is revealed, we also update the final combination weights $\boldsymbol{z}_t$ based on the final output $\hat{d}_t = \boldsymbol{z}_t^T \boldsymbol{y}_t$, where $\hat{d}_t = \boldsymbol{z}_t^T \boldsymbol{y}_t$, $\boldsymbol{y}_t = [\hat{d}_t^{(1)}, \ldots, \hat{d}_t^{(m)}]^T$. To update the final combination weights, we use the normalized LMS algorithm [32] yielding

$$\boldsymbol{z}_{t+1} = \boldsymbol{z}_t + \mu_z e_t \frac{\boldsymbol{y}_t}{\|\boldsymbol{y}_t\|^2}. \tag{3.8}$$

3.2 Choice of Parameter Values

The choice of σ_m^2 is a crucial task, i.e., we cannot reach any desired MSE for any data sequence unconditionally. Suppose we aim to boost the performance of a constituent filter from a specific class of learning algorithms. Clearly, we cannot perform better than the best offline algorithm in this class. Therefore, it is reasonable to assume that $\sigma_m^2 \geq \gamma_{\text{off}}^2$, where γ_{off}^2 indicates the (unweighted) MSE of the best offline algorithm in the class. This, in turn, results in the following upper bound on the κ.

$$\kappa \leq \frac{\gamma_{\text{off}}^2 - \sigma_d^2}{\gamma_{\text{off}}^2 - 4} \tag{3.9}$$

Since the upper bound in (3.9) is a decreasing function of the γ_{off}^2, if we use a stronger class, i.e., if γ_{off}^2 is smaller, we can use a greater value for κ. We emphasize that a greater κ leads to a smaller number of CFs (M), i.e., less computational complexity.

Intuitively, there is a guaranteed upper bound (i.e., σ^2) on the worst case performance, since in the weighted MSE, the samples with a higher error have a more important effect. On the other hand, if one chooses a σ_m^2 smaller than the noise power, $l_t^{(k)}$ will be negative for almost every k, turning most of the weights into 1, and as a result the constituent filters fail to reach a WMSE smaller than σ^2. Nevertheless, in practice we have to choose the parameter σ_m^2 reasonably and precisely such that the conditions of Theorem 2 are satisfied. For instance, we set σ_m^2 to be an upper bound on the noise power.

In addition, the number of constituent filters, m, is chosen regarding to the computational complexity constraints. However, in our experiments we choose a moderate number of constituent filters, $m = 20$, which successfully improves the performance. Moreover, according to the results in Section 7.1.3, the optimum value for c is around 1, hence, we set the parameter $c = 1$ in our simulations.

Chapter 4

Boosted Linear Adaptive Filters

4.1 Boosted RLS Algorithms

At each time t, all of the CFs (shown in Fig. 3.1) estimate the desired data d_t in parallel, and the final estimate is a linear combination of the results generated by the CFs. When the k^{th} CF receives the weight $\lambda_t^{(k)}$, it updates the linear coefficients $\boldsymbol{w}_t^{(k)}$ using one of the following methods.

4.1.1 Directly Using λ's as Sample Weights

Here, we consider $\lambda_t^{(k)}$ as the weight for the observation pair $(\boldsymbol{x}_t, d_t)$ and apply a weighted RLS update to $\boldsymbol{w}_t^{(k)}$. For this particular weighted RLS algorithm, we define the Hessian matrix and the gradient vector as

$$\boldsymbol{R}_{t+1}^{(k)} \triangleq \beta \boldsymbol{R}_t^{(k)} + \lambda_t^{(k)} \boldsymbol{x}_t \boldsymbol{x}_t^T, \tag{4.1}$$

$$\boldsymbol{p}_{t+1}^{(k)} \triangleq \beta \boldsymbol{p}_t^{(k)} + \lambda_t^{(k)} \boldsymbol{x}_t d_t, \tag{4.2}$$

where β is the forgetting factor [32] and $\boldsymbol{w}_{t+1}^{(k)} = \left(\boldsymbol{R}_{t+1}^{(k)}\right)^{-1} \boldsymbol{p}_{t+1}^{(k)}$ can be calculated in a recursive manner as

$$\begin{aligned}
e_t^{(k)} &= d_t - \boldsymbol{x}_t^T \boldsymbol{w}_t^{(k)}, \\
\boldsymbol{g}_t^{(k)} &= \frac{\lambda_t^{(k)} \boldsymbol{P}_t^{(k)} \boldsymbol{x}_t}{\beta + \lambda_t^{(k)} \boldsymbol{x}_t^T \boldsymbol{P}_t^{(k)} \boldsymbol{x}_t}, \\
\boldsymbol{w}_{t+1}^{(k)} &= \boldsymbol{w}_t^{(k)} + e_t^{(k)} \boldsymbol{g}_t^{(k)}, \\
\boldsymbol{P}_{t+1}^{(k)} &= \beta^{-1} \left(\boldsymbol{P}_t^{(k)} - \boldsymbol{g}_t^{(k)} \boldsymbol{x}_t^T \boldsymbol{P}_t^{(k)} \right).
\end{aligned} \tag{4.3}$$

where $\boldsymbol{P}_t^{(k)} \triangleq \left(\boldsymbol{R}_t^{(k)}\right)^{-1}$, and $\boldsymbol{P}_0^{(k)} = v^{-1}\boldsymbol{I}$, and $0 < v \ll 1$. The complete algorithm is given in Algorithm 2 with the weighted RLS implementation in (4.3).

4.1.2 Data Reuse Approaches Based on The Weights

Another approach follows Ozaboost [29]. In this approach, from $\lambda_t^{(k)}$, we generate an integer, say $n_t^{(k)} = \text{ceil}(K\lambda_t^{(k)})$, where K is a design parameter that takes on positive integer values. We then apply the RLS update on the $(\boldsymbol{x}_t, d_t)$ pair repeatedly $n_t^{(k)}$ times, i.e., run the RLS update on the same $(\boldsymbol{x}_t, d_t)$ pair $n_t^{(k)}$ times consecutively. Note that K should be determined according to the computational complexity constraints. However, increasing K does not necessarily result in a better performance, therefore, we use moderate values for K, e.g., we use $K = 5$ in our simulations. The final $\boldsymbol{w}_{t+1}^{(k)}$ 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 [29] uses a different data reuse strategy. In this approach, $\lambda_t^{(k)}$ is used as the parameter of a Poisson distribution and an integer $n_t^{(k)}$ is randomly generated from this Poisson distribution. One then applies the RLS update $n_t^{(k)}$ times.

4.1.3 Random Updates Approach Based on The Weights

In this approach, we simply use the weight $\lambda_t^{(k)}$ as a probability of updating the k^{th} CF 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, we update the k^{th} CF, 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 performance of the previous constituent CFs, this method does not degrade the MSE performance, 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.

4.2 Boosted LMS Algorithms

In this case, as shown in Fig. 3.1, we have m parallel running CFs, each of which is updated using the LMS algorithm. Based on the weights given in (3.6) and the total loss and MSE parameters in (3.4) and (3.7), we next introduce three LMS based boosting algorithms, similar to those introduced in Section 4.1.

4.2.1 Directly Using λ's to Scale The Learning Rates

We note that by construction method in (3.6), $0 < \lambda_t^{(k)} \leq 1$, thus, these weights can be directly used to scale the learning rates for the LMS updates. When the k^{th} CF receives the weight $\lambda_t^{(k)}$, it updates its coefficients $\boldsymbol{w}_t^{(k)}$, as

$$\boldsymbol{w}_{t+1}^{(k)} = \left(\boldsymbol{I} - \mu^{(k)}\lambda_t^{(k)}\boldsymbol{x}_t\boldsymbol{x}_t^T\right)\boldsymbol{w}_t^{(k)} + \mu^{(k)}\lambda_t^{(k)}\boldsymbol{x}_t d_t, \tag{4.4}$$

where $0 < \mu^{(k)}\lambda_t^{(k)} \leq \mu^{(k)}$. Note that we can choose $\mu^{(k)} = \mu$ for all k, since the online algorithms work consecutively from top to bottom, and the k^{th} CF will have a different learning rate $\mu^{(k)}\lambda_t^{(k)}$.

4.2.2 A Data Reuse Approach Based on The Weights

In this scenario, for updating $\boldsymbol{w}_t^{(k)}$, we use the LMS update $n_t^{(k)} = \text{ceil}(K\lambda_t^{(k)})$ times to obtain the $\boldsymbol{w}_{t+1}^{(k)}$ as

$$\begin{aligned}
&\boldsymbol{q}^{(0)} = \boldsymbol{w}_t^{(k)}, \\
&\boldsymbol{q}^{(a)} = \left(\boldsymbol{I} - \mu^{(k)}\boldsymbol{x}_t\boldsymbol{x}_t^T\right)\boldsymbol{q}^{(a-1)} + \mu^{(k)}\boldsymbol{x}_t d_t,\ a = 1, \ldots, n_t^{(k)}, \\
&\boldsymbol{w}_{t+1}^{(k)} = \boldsymbol{q}^{\left(n_t^{(k)}\right)}.
\end{aligned} \tag{4.5}$$

where K is a constant design parameter.

Similar to the RLS case, if we follow the Ozaboost [29], 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 $\boldsymbol{w}_t^{(k)}$ as explained above.

4.2.3 Random Updates Based on The Weights

Again, in this scenario, similar to the RLS case, we use the weight $\lambda_t^{(k)}$ to generate a random number from a Bernoulli distribution, which equals 1 with probability $\lambda_t^{(k)}$, and equals 0 with probability $1 - \lambda_t^{(k)}$. Then we update $\boldsymbol{w}_t$ using LMS only if the generated number is 1.

Algorithm 2 Boosted RLS-based algorithm

Input: $(\boldsymbol{x}_t, d_t)$ (data stream), m (number of CFs) and σ_m^2.
Initialize the regression coefficients $\boldsymbol{w}_1^{(k)}$ for each CF; and the combination coefficients as $\boldsymbol{z}_1 = \frac{1}{m}[1, 1, \ldots, 1]^T$; and for all k set $\delta_0^{(k)} = 0$.
for $t = 1$ **to** T **do**
  Receive the regressor data instance $\boldsymbol{x}_t$;
  Compute the CFs outputs $\hat{d}_t^{(k)} = \boldsymbol{x}_t^T \boldsymbol{w}_t^{(k)}$;
  Produce the final estimate $\hat{d}_t = \boldsymbol{z}_t^T [\hat{d}_t^{(1)}, \ldots, \hat{d}_t^{(m)}]^T$;
  Receive the true output d_t (desired data);
  $\lambda_t^{(1)} = 1$; $l_t^{(1)} = 0$;
  for $k = 1$ **to** m **do**
    $\lambda_t^{(k)} = \min \left\{ 1, \left(\delta_{t-1}^{(k)} \right)^{c\, l_t^{(k)}} \right\}$;
    Update the regression coefficients $\boldsymbol{w}_t^{(k)}$ by using the RLS and the weight $\lambda_t^{(k)}$ based on one of the introduced algorithms in Section 4.1;
    $e_t^{(k)} = d_t - \hat{d}_t^{(k)}$;
    $\delta_t^{(k)} = \frac{\Lambda_{t-1}^{(k)} \delta_{t-1}^{(k)} + \frac{\lambda_t^{(k)}}{4} \left(d_t - \left[f_t^{(k)}(\boldsymbol{x}_t) \right]^+ \right)^2}{\Lambda_{t-1}^{(k)} + \lambda_t^{(k)}}$;
    $\Lambda_t^{(k)} = \Lambda_{t-1}^{(k)} + \lambda_t^{(k)}$
    $l_t^{(k+1)} = l_t^{(k)} + \left[\sigma_m^2 - \left(e_t^{(k)} \right)^2 \right]$;
  end for
  $e_t = d_t - \boldsymbol{z}_t^T \boldsymbol{y}_t$;
  $\boldsymbol{z}_{t+1} = \boldsymbol{z}_t + \mu_z e_t \frac{\boldsymbol{y}_t}{\|\boldsymbol{y}_t\|^2}$;
end for

Chapter 5

Boosted Piecewise Linear Adaptive Filters

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

$$\hat{d}_t = \sum_{i=1}^{N} s_{i,t} \boldsymbol{w}_{i,t}^T \boldsymbol{x}_t, \tag{5.1}$$

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

$$s_{i,t} = \begin{cases} 1 & \text{if } \boldsymbol{x}_t \in \mathcal{R}_i \\ 0 & \text{if } \boldsymbol{x}_t \notin \mathcal{R}_i. \end{cases} \tag{5.2}$$

Note that at each time t, only one of the $s_{i,t}$'s is nonzero, which indicates the region in which $\boldsymbol{x}_t$ lies. Thus, if $x_t \in \mathcal{R}_i$, we update only the ith linear filter. As an example, consider 2-dimensional input vectors $\boldsymbol{x}_t$, as depicted in Fig. 5.1. Here, we construct the piecewise linear filter f_t such that

$$\begin{aligned} \hat{d}_t = f_t(\boldsymbol{x}_t) &= s_{1,t} \boldsymbol{w}_{1,t}^T \boldsymbol{x}_t + s_{2,t} \boldsymbol{w}_{2,t}^T \boldsymbol{x}_t \\ &= s_t \boldsymbol{w}_{1,t}^T \boldsymbol{x}_t + (1 - s_t) \boldsymbol{w}_{2,t}^T \boldsymbol{x}_t, \end{aligned} \tag{5.3}$$

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

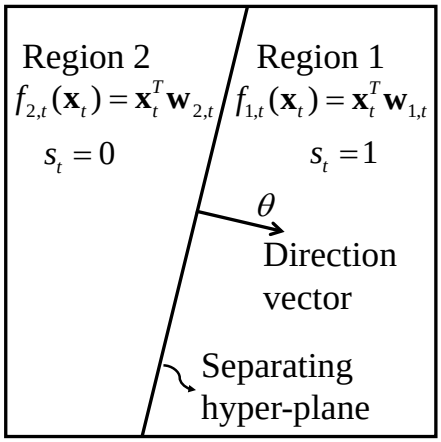


Figure 5.1: A sample 2-region partition of the input vector (i.e., $\boldsymbol{x}_t$) space, which is 2-dimensional in this example. s_t determines whether $\boldsymbol{x}_t$ is in Region 1 or not, hence, can be used as the indicator function for this region. Similarly, $1 - s_t$ serves as the indicator function of Region 2.

Now, we present different variants of the aforementioned piecewise linear filter, based on the introduced boosting algorithm in Chapter 3. We emphasize that one can use either LMS or RLS algorithm to update the linear filters in each region of a piecewise linear constituent filter. However, as we show now, extending of our method to these scenarios is straightforward.

5.1 Boosted RLS-based Piecewise Linear Algorithms

As depicted in Fig. 5.2, 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. 3.1) 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

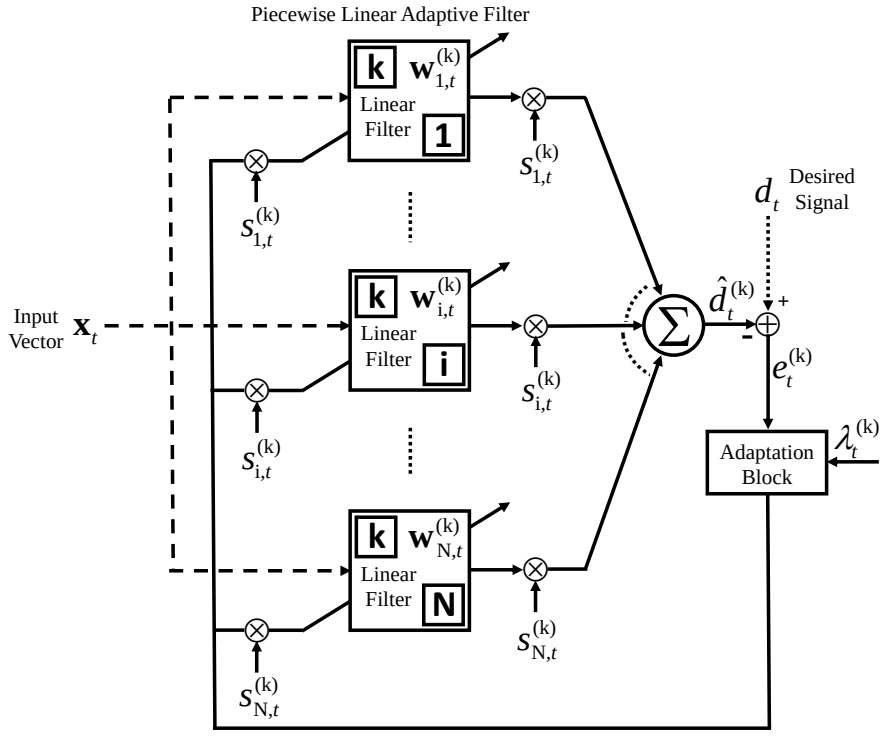


Figure 5.2: A sample piecewise linear adaptive filter, used as the kth constituent filter in the system depicted in Fig. 3.1. This fliter consists of N linear filters, one of which produces the estimate at each iteration t. Based on where the input vector at time t, $\boldsymbol{x}_t$, lies in the input 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.

filter, as was explained before. 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 $\boldsymbol{w}_{i,t}^{(k)}$, assuming that $\boldsymbol{x}_t$ lies in the ith region of the kth constituent filter. We consider $\lambda_t^{(k)}$ as the weight for the observation pair $(d_t, \boldsymbol{x}_t)$ and apply a weighted RLS update to $\boldsymbol{w}_{i,t}^{(k)}$.
Consider a "Weighted Updates" approach for boosting. Therefore, for this particular weighted RLS algorithm, we define the autocorrelation matrix and the cross correlation vector as

$$\boldsymbol{R}_{i,t+1}^{(k)} \triangleq \beta \boldsymbol{R}_{i,t}^{(k)} + \lambda_t^{(k)} \boldsymbol{x}_t \boldsymbol{x}_t^T, \tag{5.4}$$

$$\boldsymbol{p}_{i,t+1}^{(k)} \triangleq \beta \boldsymbol{p}_{i,t}^{(k)} + \lambda_t^{(k)} \boldsymbol{x}_t d_t, \tag{5.5}$$

where β is the forgetting factor [32] and $\boldsymbol{w}_{i,t+1}^{(k)} = \left(\boldsymbol{R}_{i,t+1}^{(k)}\right)^{-1} \boldsymbol{p}_{i,t+1}^{(k)}$ can be calculated in a recursive manner as

$$\begin{aligned}
e_t^{(k)} &= d_t - \boldsymbol{x}_t^T \boldsymbol{w}_{i,t}^{(k)}, \\
\boldsymbol{g}_{i,t}^{(k)} &= \frac{\lambda_t^{(k)} \boldsymbol{P}_{i,t}^{(k)} \boldsymbol{x}_t}{\beta + \lambda_t^{(k)} \boldsymbol{x}_t^T \boldsymbol{P}_{i,t}^{(k)} \boldsymbol{x}_t}, \\
\boldsymbol{w}_{t+1}^{(k)} &= \boldsymbol{w}_{i,t}^{(k)} + e_t^{(k)} \boldsymbol{g}_{i,t}^{(k)}, \\
\boldsymbol{P}_{i,t+1}^{(k)} &= \beta^{-1} \left(\boldsymbol{P}_{i,t}^{(k)} - \boldsymbol{g}_{i,t}^{(k)} \boldsymbol{x}_t^T \boldsymbol{P}_{i,t}^{(k)} \right).
\end{aligned} \tag{5.6}$$

where $\boldsymbol{P}_{i,t}^{(k)} \triangleq \left(\boldsymbol{R}_{i,t}^{(k)}\right)^{-1}$, and $\boldsymbol{P}_{i,0}^{(k)} = v^{-1}\boldsymbol{I}$ for $i = 1, \ldots, N$, and $0 < v \ll 1$. One can obtain similar updating methods for the "Data Reuse" and "Random Updates" as well.

5.2 Boosted LMS-based Piecewise Linear Algorithms

In this case, as shown in Fig. 3.1, 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 $\boldsymbol{x}_t$ lies in the ith region of the kth filter partition, $s_{i,t}^{(k)} = 1$, hence, we use $\boldsymbol{w}_{i,t}^{(k)}$ to estimate d_t, and update this linear filter with its own learning rate $\mu_i^{(k)}$. Based on the weights given in (3.6) and the total loss and MSE parameters in equations (3.4) and (3.7), we can use three LMS based boosting algorithms, similar to those introduced in Chapter 4.

For instance, in the "Weighted Updates" scenario, we adjust the filter coefficients in each region of the constituent filters using the following equation.

$$\boldsymbol{w}_{i,t+1}^{(k)} = \left(\boldsymbol{I} - \mu_i^{(k)} \lambda_t^{(k)} \boldsymbol{x}_t \boldsymbol{x}_t^T \right) \boldsymbol{w}_{i,t}^{(k)} + \mu_i^{(k)} \lambda_t^{(k)} \boldsymbol{x}_t d_t, \tag{5.7}$$

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 $\mu_i \lambda_t^{(k)}$. Also, other variants can be straightforwardly obtained in a similar manner.

Remark 2: 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 [36] to make the partitioning adaptive. As an example, suppose that each constituent filter is defined on a 2-region partition, as shown in Fig. 5.1, 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 $\boldsymbol{x}_t$ lies, we use an indicator function $s_t^{(k)}$ defined as follows

$$s_t^{(k)} = \frac{1}{1 + \exp(-\boldsymbol{\theta}_t^T \boldsymbol{x}_t)}, \tag{5.8}$$

and the estimate made by the kth filter is represented by

$$\hat{d}_t^{(k)} = s_t^{(k)} \hat{d}_{1,t}^{(k)} + \left(1 - s_t^{(k)}\right) \hat{d}_{2,t}^{(k)} \tag{5.9}$$

which, yields the following ordinary LMS update for $\boldsymbol{\theta}_t^{(k)}$ [36]

$$\begin{aligned} \boldsymbol{\theta}_{t+1}^{(k)} &= \boldsymbol{\theta}_t^{(k)} + \mu_{\boldsymbol{\theta}} e_t^{(k)} \left(\hat{d}_{1,t}^{(k)} - \hat{d}_{2,t}^{(k)}\right) \nabla_{\boldsymbol{\theta}_t} \left(s_t^{(k)}\right) \\ &= \boldsymbol{\theta}_t^{(k)} + \mu_{\boldsymbol{\theta}} e_t^{(k)} \left(\hat{d}_{1,t}^{(k)} - \hat{d}_{2,t}^{(k)}\right) s_t^{(k)} \left(1 - s_t^{(k)}\right) \boldsymbol{x}_t. \end{aligned} \tag{5.10}$$

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 $\boldsymbol{\theta}_t^{(k)}$

$$\boldsymbol{\theta}_{t+1}^{(k)} = \boldsymbol{\theta}_t^{(k)} + \mu_{\boldsymbol{\theta}} \lambda_t^{(k)} e_t^{(k)} \left(\hat{d}_{2,t}^{(k)} - \hat{d}_{1,t}^{(k)}\right) s_t^{(k)} \left(1 - s_t^{(k)}\right) \boldsymbol{x}_t. \tag{5.11}$$

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

$$
\begin{aligned}
\boldsymbol{\vartheta}^{(a+1)} &= \boldsymbol{\vartheta}^{(a)} + \mu_{\boldsymbol{\theta}} \epsilon^{(a)} \boldsymbol{x}_t \boldsymbol{x}_t^T \left(\boldsymbol{q}_1^{(a)} - \boldsymbol{q}_2^{(a)} \right) \psi^{(a)} \left(1 - \psi^{(a)} \right), \\
\boldsymbol{q}_1^{(a+1)} &= \boldsymbol{q}_1^{(a)} + \mu_i^{(k)} \psi^{(a)} \epsilon^{(a)} \boldsymbol{x}_t, \\
\boldsymbol{q}_2^{(a+1)} &= \boldsymbol{q}_2^{(a)} + \mu_i^{(k)} (1 - \psi^{(a)}) \epsilon^{(a)} \boldsymbol{x}_t, \\
\psi^{(a+1)} &= \frac{1}{1 + \exp(-\boldsymbol{\vartheta}_t^T \boldsymbol{x}_t)}, \\
\epsilon^{(a+1)} &= d_t - \left(\psi^{(a+1)} \boldsymbol{q}_1^{(a+1)} + (1 - \psi^{(a+1)}) \boldsymbol{q}_2^{(a+1)} \right) \boldsymbol{x}_t,
\end{aligned} \tag{5.12}
$$

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 $\boldsymbol{q}_i^{(0)} = \boldsymbol{w}_{i,t}^{(k)}$ for $i = 1, 2$. Also, the updated values are $\boldsymbol{\theta}_{t+1}^{(k)} = \boldsymbol{\vartheta}^{(n_t^{(k)})}$, and $\boldsymbol{w}_{i,t+1}^{(k)} = \boldsymbol{q}_i^{(n_t^{(k)})}$ for $i = 1, 2$.

Chapter 6

Analysis Of The Proposed Algorithms

In this section we provide the 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 the other proposed algorithms, and slightly greater than that of a single CF. Hence, it shows the considerable advantage of "boosting with random updates" in processing of high dimensional data.

6.1 Complexity Analysis

Here we compare the complexity of the proposed algorithms and find an upper bound for the computational complexity of random updates scenario (introduced in Section 4.1.3 for RLS, and in Section 4.2.3 for LMS updates), which shows its significantly lower computational burden with respect to two other approaches. For $\boldsymbol{x}_t \in \mathbb{R}^r$, each CF 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 $\mathbf{R}_t^{(k)}$, while it needs $O(r)$ computations when updated using the LMS

method (in their most basic implementation).

We first derive the computational complexity of using the RLS updates in different boosting scenarios. Since there are a total of m CFs, all of which are updated in the "weighted updates" method, this method has a computational cost of order $O(mr^2)$ per each iteration t. However, in the "random updates", at iteration t, the k^{th} CF may or may not be updated with probabilities $\lambda_t^{(k)}$ and $1-\lambda_t^{(k)}$ respectively, yielding

$$C_t^{(k)} = \begin{cases} O(r^2) & \text{with probability } \lambda_t^{(k)} \\ O(r) & \text{with probability } 1-\lambda_t^{(k)}, \end{cases} \tag{6.1}$$

where $C_t^{(k)}$ indicates the complexity of running the k^{th} CF 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

$$E\left[C_t\right] = E\left[\sum_{k=1}^{m} C_t^{(k)}\right] = \sum_{k=1}^{m} E[\lambda_t^{(k)}]O(r^2) \tag{6.2}$$

Hence, if $E\big[\lambda_t^{(k)}\big]$ is upper bounded by $\tilde{\lambda}^{(k)} < 1$, the average computational complexity of the random updates method, will be

$$E\left[C_t\right] < \sum_{k=1}^{m} \tilde{\lambda}^{(k)} O(r^2). \tag{6.3}$$

In Theorem 3, 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 CF $f_t^{(k)}$, we perform the RLS update $\lambda_t^{(k)} K$ times, resulting a computational complexity of order $E\left[C_t\right] < \sum_{k=1}^{m} K\ \tilde{\lambda}^{(k)}(O(r^2))$. For the LMS updates, we similarly obtain the computational complexities $O(mr)$, $\sum_{k=1}^{m} O\big(\tilde{\lambda}^{(k)} r\big)$, and $\sum_{k=1}^{m} O\big(K\tilde{\lambda}^{(k)} r\big)$, for the "weighted updates", "random updates", and "data reuse" scenarios respectively.

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

Theorem 3. *If the CFs 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_m^2 *is chosen properly,*

$$E\left[\lambda_t^{(k)}\right] \leq \tilde{\lambda}^{(k)} = \left(\gamma^{-2\sigma_m^2}(1+2\zeta^2\ln\gamma)\right)^{\frac{1-k}{2}}, \tag{6.4}$$

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_m^2, and reasonable values for the MSE according to the proof. This theorem states that if we adjust σ_m^2 such that it is achievable, i.e., the CFs can provide a slightly lower MSE than σ_m^2, the probability of updating the CFs in the random updates scenario will decrease. This is of course our desired results, since if the CFs are performing sufficiently well, there is no need for additional updates. Moreover, if σ_m^2 is opted such that the CFs cannot achieve a MSE equal to σ_m^2, the CFs 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

$$\begin{aligned} E\left[\lambda_t^{(k)}\right] &= E\left[\min\left\{1, \left(\delta_{t-1}^{(k)}\right)^{l_t^{(k)}}\right\}\right] \\ &\leq \min\left\{1, E\left[\left(\delta_{t-1}^{(k)}\right)^{l_t^{(k)}}\right]\right\} \end{aligned} \tag{6.5}$$

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 obtain an upper bound for $E\left[\lambda_t^{(k)}\right]$. We define $s \triangleq \ln(\delta_{t-1}^{(k)})$, yielding

$$E\left[\left(\delta_{t-1}^{(k)}\right)^{l_t^{(k)}}\right] = E\left[E\left[\exp\left(s\ l_t^{(k)}\right)\middle| s\right]\right] = E\left[M_{l_t^{(k)}}(s)\middle| s\right], \tag{6.6}$$

where $M_{l_t^{(k)}}(.)$ is the moment generating function of the random variable $l_t^{(k)}$. From the Algorithm 2, $l_t^{(k)} = (k-1)\sigma_m^2 - \sum_{j=1}^{k-1}\left(e_t^{(j)}\right)^2$. 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 $\Gamma\left(\frac{k-1}{2}, 2\zeta^2\right)$ [57], which results in the following moment generating function for $l_t^{(k)}$

$$M_{l_t^{(k)}}(s) = \exp\left(s(k-1)\sigma_m^2\right)\left(1+2\zeta^2 s\right)^{\frac{1-k}{2}} = \left(\delta_{t-1}^{(k)}\right)^{(k-1)\sigma_m^2}\left(1+2\zeta^2\ln\left(\delta_{t-1}^{(k)}\right)\right)^{\frac{1-k}{2}}. \quad (6.7)$$

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 the Jenssen's inequality for concave functions, we have

$$E\left[\lambda_t^{(k)}\right] \leq \varphi(\gamma). \quad (6.8)$$

Inspired by (6.7), we define $A\left(\delta_{t-1}^{(k)}\right) \triangleq {\delta_{t-1}^{(k)}}^{-2\sigma_m^2}\left(1+2\zeta^2\ln\left(\delta_{t-1}^{(k)}\right)\right)$ 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

$$\varphi''\left(\delta_{t-1}^{(k)}\right) = \frac{1-k}{2}\left(A\left(\delta_{t-1}^{(k)}\right)\right)^{\frac{-k-3}{2}}\left[\left(\frac{-k-1}{2}\right)\left(A'\left(\delta_{t-1}^{(k)}\right)\right)^2 + \left(A\left(\delta_{t-1}^{(k)}\right)\right)^2 A''\left(\delta_{t-1}^{(k)}\right)\right]. \quad (6.9)$$

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

$$\left(A\left(\delta_{t-1}^{(k)}\right)\right)^2 A''\left(\delta_{t-1}^{(k)}\right) > \left(\frac{k+1}{2}\right)\left(A'\left(\delta_{t-1}^{(k)}\right)\right)^2, \quad (6.10)$$

which reduces to the following necessary and sufficient conditions:

$$\frac{\left(\delta_{t-1}^{(k)}\right)^{2\sigma_m^2}}{\left(1+2\zeta^2\ln\left(\delta_{t-1}^{(k)}\right)\right)^2} < \frac{\left(1+2\sigma_m^2\right)^2}{4(k+1)}, \quad (6.11)$$

and

$$\frac{(1-\xi_1)\sigma_m^2}{1-2\sigma_m^2\ln\left(\delta_{t-1}^{(k)}\right)} < \zeta^2 < \frac{(1-\xi_2)\sigma_m^2}{1-2\sigma_m^2\ln\left(\delta_{t-1}^{(k)}\right)}, \quad (6.12)$$

where

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

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

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 (6.8) we achieve (6.4). This concludes the proof of the Theorem 3. $\square$

Chapter 7

Experiments and Conclusion

7.1 Experiments

In this section, we demonstrate the efficacy of the proposed boosting algorithms for RLS and LMS linear, as well as piecewise linear, CFs under different scenarios. To this end, we first consider the "online regression" of data generated with a stationary linear model. Then, we illustrate the performance of our algorithms under nonstationary conditions, to thoroughly test the adaptation capabilities of the proposed boosting framework. Furthermore, since the most important parameters in the proposed methods are σ_m^2, c, and m, we investigate their effects on the final MSE performance. Finally, we provide the results of the experiments over several real and synthetic benchmark datasets.

Throughout this section, "LMS" represents the linear LMS-based CF, "RLS" represents the linear RLS-based CF, and a prefix "B" indicates the boosting algorithms. In addition, we use the suffixes "-WU", "-RU", or "-DR" to denote the "weighted updates", "random updates", or "data reuse" modes, respectively, e.g., the "BLMS-RU" represents the "Boosted LMS-based algorithm using Random Updates". Also, a prefix "P" before the "LMS" or "RLS" indicates a piecewise linear filter with two regions, based on the corresponding update method,

and "SPLMS" denotes an LMS-based piecewise linear filter with "Soft" (flexible) boundaries.

In order to observe the boosting effect, in all experiments, we set the step size of LMS and the forgetting factor of the RLS to their optimal values, and use those parameters for the CFs, too. In addition, the initial values of all of the constituent filters in all of the experiments are set to zero. However, in all experiments, since we use $K = 5$ in Data Reuse variants, we set the step size of the CFs in BLMS-DR method to $\mu/K = \mu/5$, where, μ is the step size of the LMS. To compare the performance, we have provided the MSE results.

7.1.1 Stationary and Non-Stationary Data

In this experiment, we consider the case where the desired data is generated by a stationary linear model. The input vectors $\boldsymbol{x}_t = [x_1 \; x_2 \; 1]$ are 3-dimensional, where $[x_1 \; x_2]$ is drawn from a jointly Gaussian random process and then scaled such that $\boldsymbol{x}_t = [x_1 \; x_2]^T \in [0 \; 1]^2$. We include 1 as the third entry of $\boldsymbol{x}_t$ to consider affine learners. Specifically the desired data is generated by $d_t = [1 \; 1 \; 0] \; \boldsymbol{x}_t + \nu_t$, where ν_t is a random Gaussian noise with a variance of 0.01. Moreover, in the non-stationary scenario, we have

$$d_t = \begin{cases} [1 \; 1 \; 0] \; \boldsymbol{x}_t + \nu_t, & t \leq T/2 \\ [1 \; -1 \; 0] \; \boldsymbol{x}_t + \nu_t, & \text{O.W.} \end{cases}$$

In our simulations, we use $m = 20$ CFs and $\mu = 0.1$ for all LMS learners. In addition, for RLS-based boosting algorithms, we set the forgetting factor $\beta = 0.9999$ for all algorithms. Moreover, we choose $\sigma_m^2 = 0.02$ for LMS-based algorithms and $\sigma_m^2 = 0.004$ for RLS-based algorithms, $K = 5$ for data reuse approaches, and $c = 1$ for all boosting algorithms. To achieve robustness, we average the results over 100 trials.

As depicted in Fig. 7.1, our proposed methods boost the performance of a

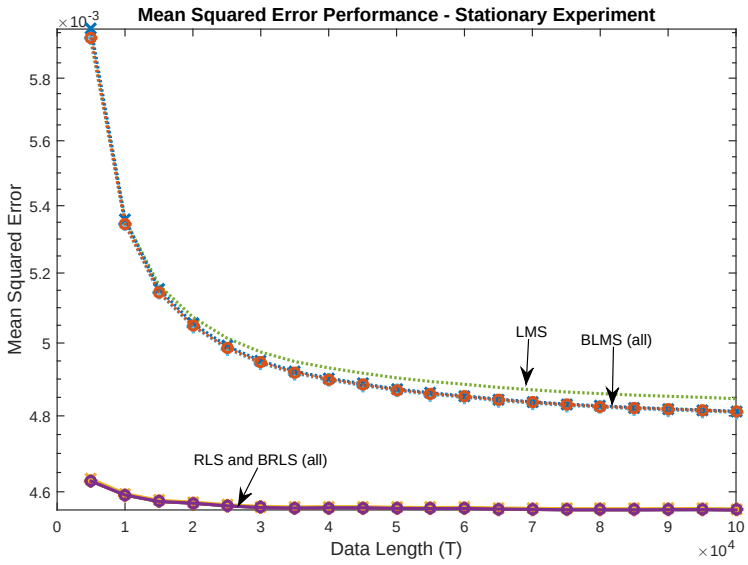


Figure 7.1: The MSE performnce of the proposed algorithms in the stationary data experiment.

single linear LMS-based CF. Nevertheless, we cannot further improve the performance of a linear RLS-based CF in such a stationary experiment since the RLS achieves the lowest MSE. We point out that the random updates method achieves the performance of the weighted updates method and the data reuse method with a much lower complexity. In addition, we observe that by increasing the data length, the performance improvement increases (Note that the distance between the ASE curves is slightly increasing). In addition, according to Fig. 7.2, our piecewise linear methods significantly outperform a single constituent filter, even in RLS-based filters.

7.1.2 Chaotic Data

Here, in order to show the tracking capability of our algorithms in nonstationary enviroRLSents, we consider the case where the desired data is generated by the Duffing map [58] as a chaotic model. Specifically, the data is generated by the following equation $x_{t+1} = 2.75x_t - x_t^3 - 0.2x_{t-1}$, where we set $x_{-1} = 0.9279$ and $x_0 = 0.1727$. We consider $d_t = x_{t+1}$ as the desired data and $[x_{t-1}\ x_t\ 1]$ as the

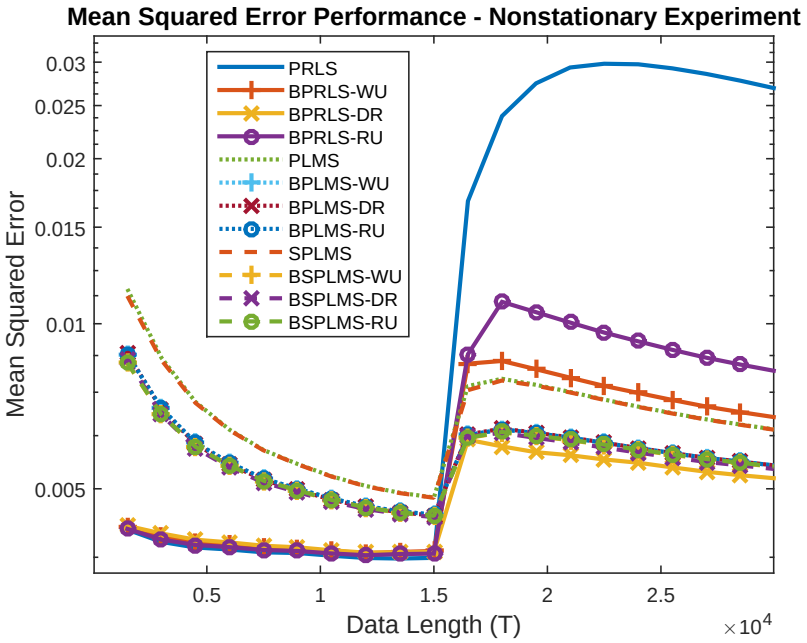


Figure 7.2: The MSE performnce of the piecewise linear filters in the non-stationary data experiment.

input vector. In this experiment, each boosting algorithm uses 20 CFs. The step sizes for the LMS-based algorithms are set to 0.1, the forgetting factor β for the RLS-based algorithms are set to 0.999, and the modified desired MSE parameter σ_m^2 is set to 0.25 for BLMS methods, and 0.17 for the BRLS methods. Note that although the value of σ_m^2 is higher than the achieved MSE, it can improve the performance significantly. This is because of the boosting effect, i.e., emphasizing on the harder data patterns. The figures show the superior performance of our algorithms over a single CF (whose step size is chosen to be the best), in this highly nonstationary enviroRLSent. Moreover, as shown in Fig. 7.3, in the LMS-based boosted algorithms, the data reuse method shows a better performance relative to the other boosting methods. However, the random updates method has a significantly lower time consumption, which makes it desirable for larger data lengths. From the Fig. 7.3, one can see that our method is truly boosting the performance of the conventional linear CFs in this chaotic enviroRLSent.

From the Fig. 7.5, we observe the approximate changes of the weights, in the BLMS-RU algorithm running over the Duffing data. As shown in this figure,

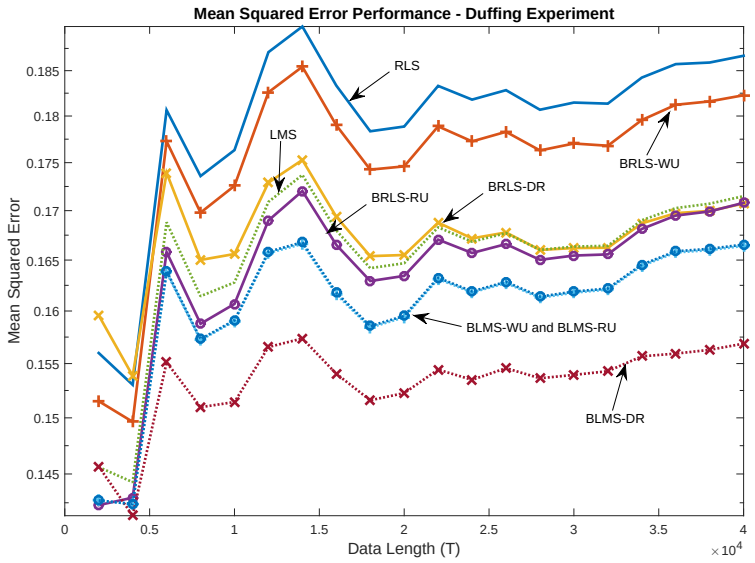


Figure 7.3: MSE performance of the proposed linear methods on a Duffing data set.

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 CFs in an ordered manner, i.e., we update the $(k+1)^{th}$ CF after the k^{th} CF is updated, the weights assigned to the last CFs are generally smaller than the weights assigned to the previous CFs. As an example, in Fig. 7.5 we see that the weights assigned to the 5^{th} CF are larger than those of the 10^{th} and 20^{th} CFs. Furthermore, note that in this experiment, the dependency parameter c is set to 1. We should mention that increasing the value of this parameter, in general, causes the lower weights, hence, it can considerably reduce the complexity of the random updates and data reuse methods.

7.1.3 The Effect of Parameters

In this section, we investigate the effects of the dependence parameter c and the modified desired MSE σ_m^2 as well as the number of CFs ,m, on the boosting performance of our methods in the Duffing data experiment, explained in Section 7.1.2. From the results in Fig. 7.6c, we observe that, increasing the number of CFs up to 30 can improve the performance significantly, while further increasing of m

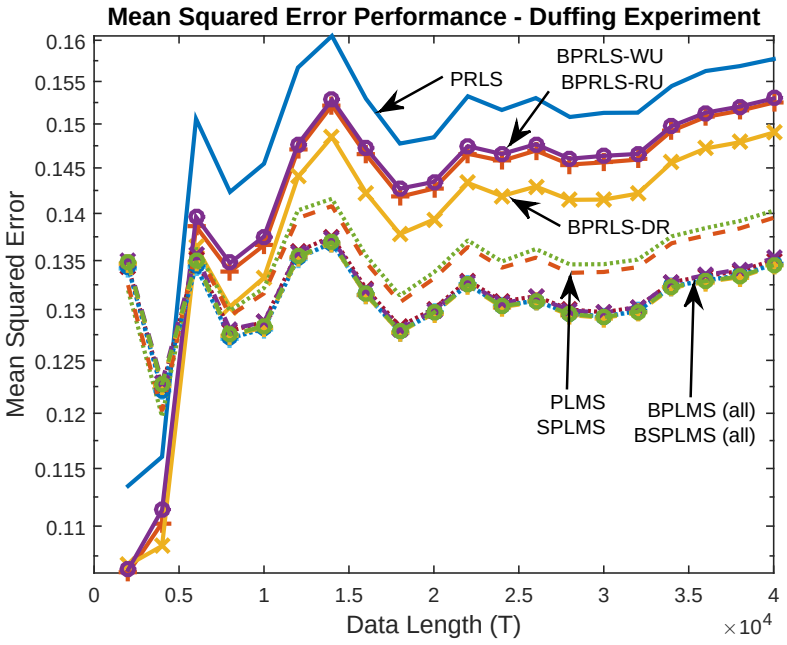


Figure 7.4: MSE performance of the proposed piecewise linear methods on a Duffing data set.

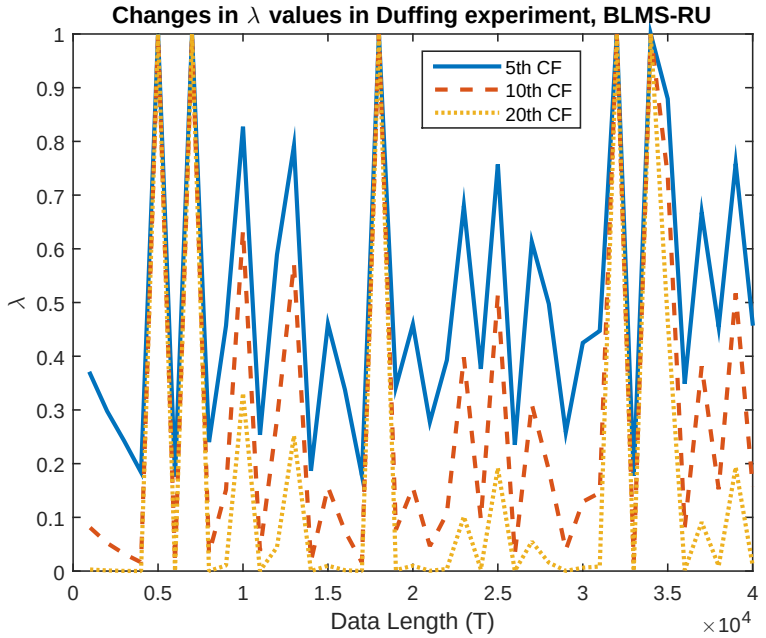


Figure 7.5: The changing of the weights in BLMS-RU algorithm in the Duffing data experiment.

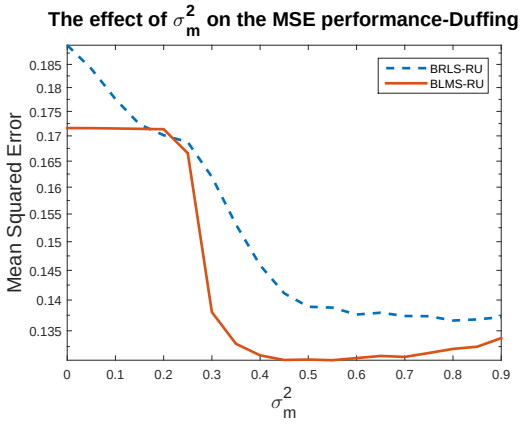
only increases the computational complexity without improving the performance. In addition, as shown in Fig. 7.6b, in this experiment, the dependency parameter c has an optimum value around 1. We note that choosing small values for c reduces the boosting effect, and causes the weights to be larger, which in turn increases the computational complexity in random updates and data reuse approaches. On the other hand, choosing very large values for c increases the dependency, i.e., in this case the generated weights are very close to 1 or 0, hence, the boosting effect is decreased.

According to the Figs. 7.6b, 7.7, 7.8, 7.9, and 7.10, we observe that the MSE curve has a minimum at a nonzero point. We emphasize that the performance improvement due to increasing c from 0, shows that our method truly boosts the adaptive filters. However, one may note the difference between the boosting effect in these scenarios. Since the ensemble of the piecewise linear filters has a slight diversity among them, the diversity risen from the boosting (that finally yield an improvement in the MSE performance) is less in these cases.

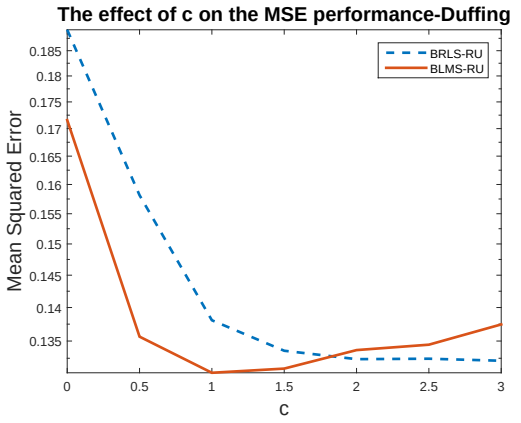
Furthermore, as depicted in Fig. 7.6a, there is an optimum value around 0.5 for σ_m^2 in this experiment. Note that, choosing small values for σ_m^2 results in large weights, thus, increases the complexity and reduces the diversity. However, choosing higher values for σ_m^2 results in smaller weights, and in turn reduces the complexity. Nevertheless, we note that increasing the value of σ_m^2 does not necessarily enhance the performance. Through the experiments, we find out that σ_m^2 must be in the order of the MSE amount to obtain the best performance.

7.1.4 Benchmark Real and Synthetic Data Sets

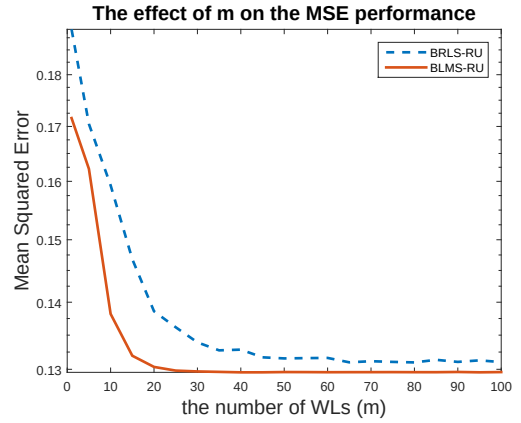
In this section, we demonstrate the efficiency of the introduced methods over some widely used real life machine learning regression data sets. We have normalized each dimension of the data to the interval $[-1, 1]$ in all algorithms. We present the MSE performance of the algorithms in Tables 7.1 and 7.2. These experiments show that our algorithms can successfully improve the performance of single linear CFs. We now describe the experiments and provide the results:



(a) The effect of the parameter σ_m^2



(b) The effect of the parameter c



(c) The effect of the parameter m

Figure 7.6: The effect of the parameters σ_m^2, c, and m, on the MSE performance of the BRLS-RU and BLMS-RU algorithms in the Duffing data experiment.

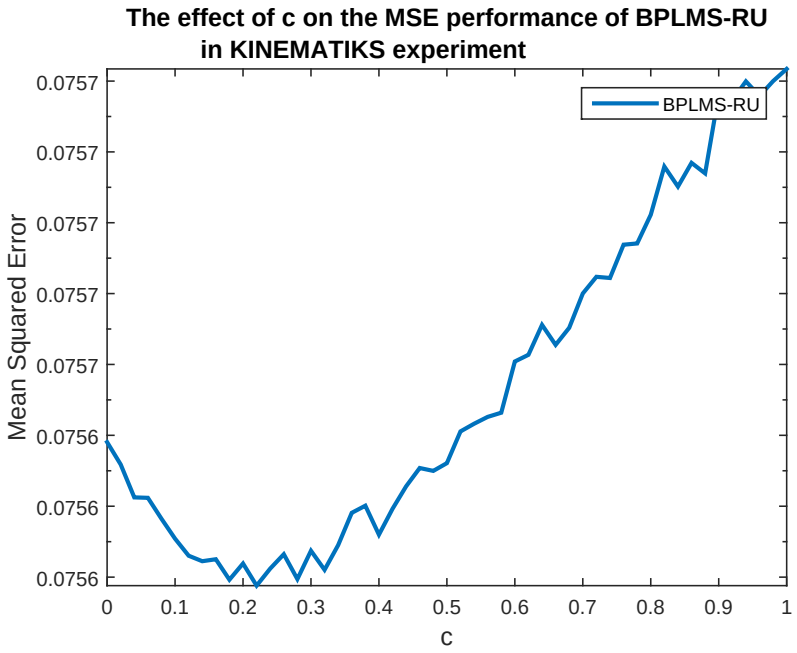


Figure 7.7: The effect of the dependency parameter on the performance of BPLMS-RU in kinematiks experiments.

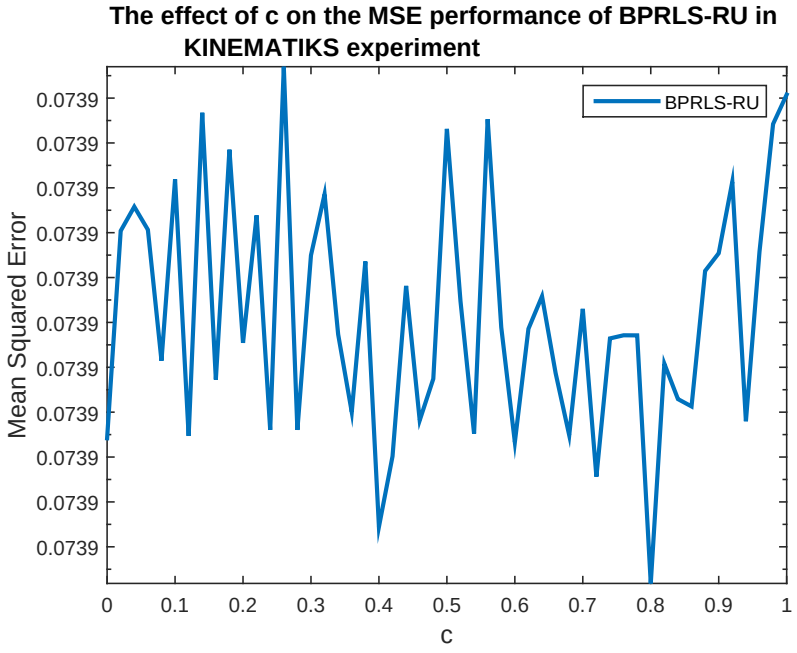


Figure 7.8: The effect of the dependency parameter on the performance of BPRLS-RU in kinematiks experiments.

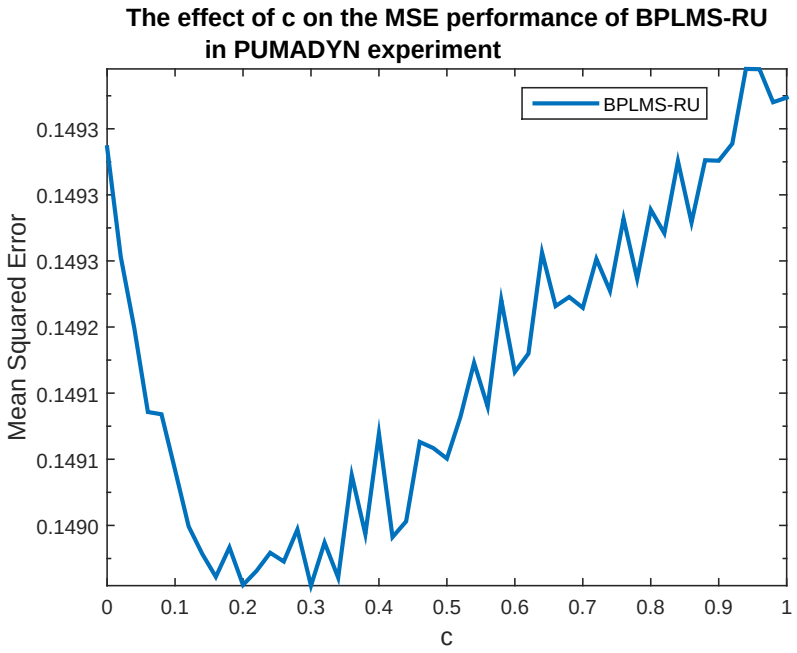


Figure 7.9: The effect of the dependency parameter on the performance of BPLMS-RU in the Puma8NH experiment.

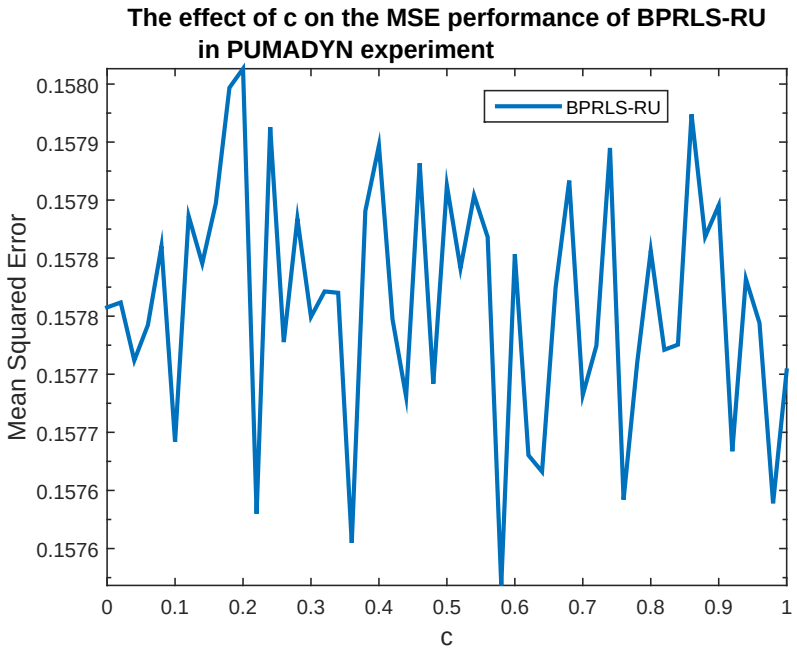


Figure 7.10: The effect of the dependency parameter on the performance of BPRLS-RU in the Puma8NH experiment.

Algorithms / Data Sets	LMS	BLMS-WU	BLMS-DR	BLMS-RU
MV	0.2711	0.2707	0.2706	0.2707
Puma8NH	0.1340	0.1334	0.1332	0.1334
Kinematics	0.0835	0.0831	0.0830	0.0831
Compactiv	0.0606	0.0599	0.0608	0.0598
Protein Tertiary	0.2554	0.2550	0.2549	0.2550
ONP	0.0015	0.0009	0.0009	0.0009
California Housing	0.0446	0.0450	0.0452	0.0448
YPMSD	0.0237	0.0237	0.0233	0.0237

Table 7.1: The MSE of the LMS-based methods on real data sets.

Algorithms / Data Sets	RLS	BRLS-WU	BRLS-DR	BRLS-RU
MV	0.2592	0.2645	0.2587	0.2584
Puma8NH	0.1296	0.1269	0.1295	0.1284
Kinematics	0.0804	0.0801	0.0803	0.0801
Compactiv	0.0137	0.0086	0.0304	0.0078
Protein Tertiary	0.2370	0.2334	0.2385	0.2373
ONP	0.0009	0.0009	0.0009	0.0009
California Housing	0.0685	0.0671	0.0579	0.0683
YPMSD	0.0454	0.0337	0.0302	0.0292

Table 7.2: The MSE of the RLS-based methods on real data sets.

Here, we briefly explain the details of the data sets:

1. MV: This is an artificial dataset with dependencies between the attribute values. One can refer to [59] for further details. There are 10 attributes and one target value. In this dataset, we can slightly improve the performance of a single linear CF by using any of the proposed methods.

2. Puma Dynamics (Puma8NH): This dataset describes a realistic modeling of the dynamics of a robot arm, called Puma 560 [59], where we seek to estimate the angular acceleration of one of the robot arm's links. To this end, we use the input features consisting of angular positions, velocities and torques of the robot arm. According to the ASE results in Fig. 7.11a, the BRLS-WU has the best boosting performance in this experiment. Nonetheless, the LMS-based methods also improve the performance.

3. Kinematics: This dataset is concerned with the forward kinematics of an 8 link robot arm [59]. We use the variant 8RLS, which is highly non-linear and noisy. As shown in Fig. 7.11b, our proposed algorithms slightly improve the performance in this experiment.

4. Computer Activity (Compactiv): This real dataset is a collection of computer systems activity measures [59]. The task is to predict USR, the portion of time that CPUs run in user mode from all attributes ([59]). The RLS-based boosting algorithms deliver a significant performance improvement in this experiment, as shown by the results in Tables 7.1 and 7.2.

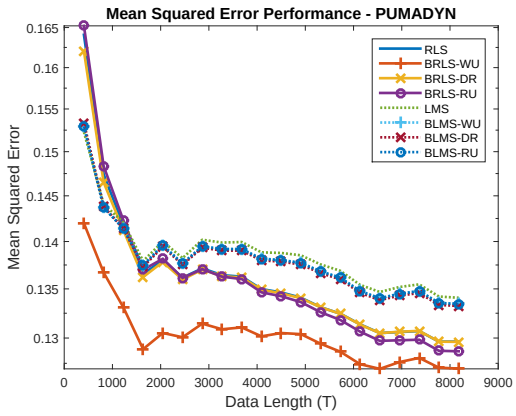
5. Protein Tertiary [60]: Having been collected from the Critical Assessment of protein Structure Prediction (CASP) experiments $5-9$, the 45730 data samples of this dataset are used to estimate the residue size using the 9 given attributes.

6. Online News Popularity (ONP) [60, 61]: This dataset consists of a heterogeneous features set regarding some articles that were published by Mashable in two consecutive years. We seek to estimate the total number of shares in social networks, which in turn shows the popularity of the articles.

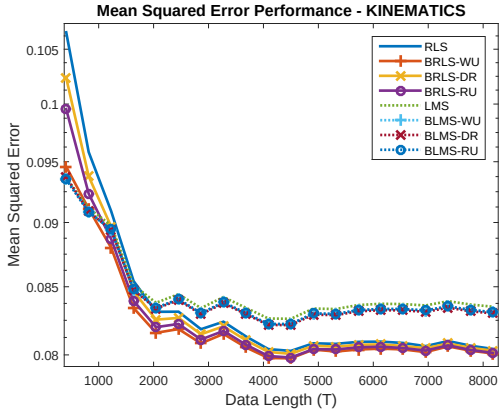
7. California Housing: This dataset has been obtained from StatLib repository. They have collected information on the variables using all the block groups in California from the 1990 Census. Here, we seek to find the house median values, based on the given attributes. For further description one can refer to [59].

8. Year Prediction Million Song Dataset (YPMSD) [62]: The aim is predicting the year when a song has been released, using its given audio features. This

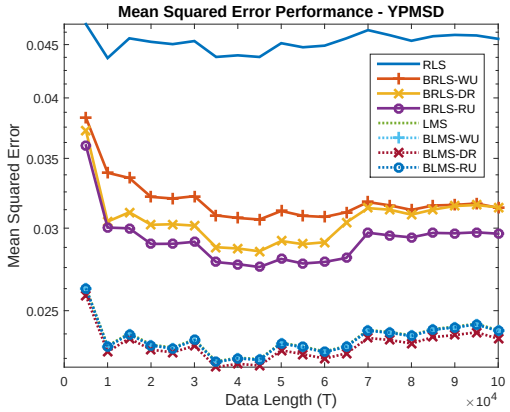
dataset mainly includes western commercial song tracks released between 1922 and 2011. We use a subset of the Million Song Dataset [62]. As shown in Tables 7.1 and 7.2 and Fig. 7.11c, our algorithms can significantly improve the performance of the linear CF in this experiment.



(a) Puma8NH

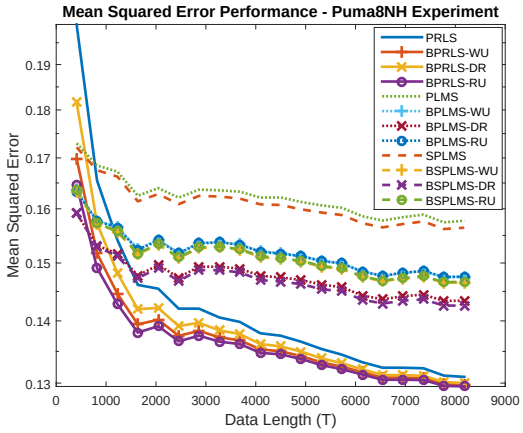


(b) Kinematics.

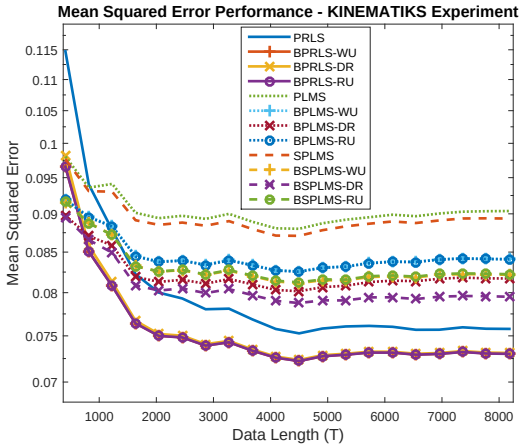


(c) YPMSD

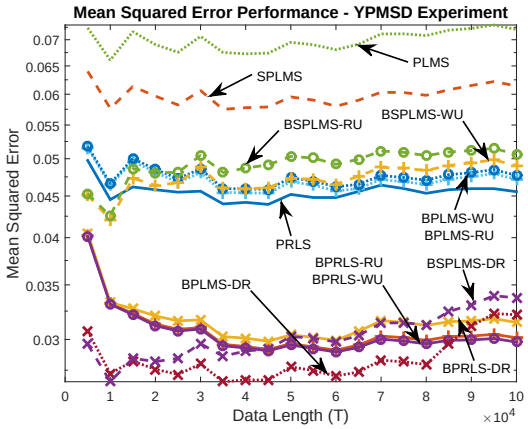
Figure 7.11: The performance of the linear methods on three real life data sets.



(a) Puma8NH



(b) Kinematics.



(c) YPMSD

Figure 7.12: The performance of the piecewise linear methods on three real life data sets.

7.2 Conclusion

We introduced a novel family of boosted adaptive filters and proposed three different boosting approaches, i.e., weighted updates, data reuse, and random updates, which can be applied to different online learning algorithms (adaptive filters). We provide theoretical bounds for the MSE performance of our proposed methods in a strong mathematical sense. We emphasize that while using the proposed techniques, we do not assume any prior information about the statistics of the desired data or feature vectors. We show that by the proposed boosting approaches, we can significantly improve the MSE performance of the conventional LMS and RLS algorithms, both in the linear and piecewise linear scenarios. Moreover, we provide an upper bound for the weights generated during the algorithm that leads us to a thorough analysis of the computational complexity of these methods. The computational complexity of the random updates method is remarkably lower than that of the conventional mixture-of-experts and other variants of the proposed boosting approaches, without degrading the performance. Therefore, the boosting using random updates approach is an elegant alternative to the conventional mixture-of-experts method when dealing with real life large scale problems. We provide several results that demonstrate the strength of the proposed algorithms over a wide variety of synthetic as well as real data.

Bibliography

[1] S. Pan, J. Wu, X. Zhu, G. Long, and C. Zhang, "Boosting for graph classification with universum," *Knowledge and Information Systems*, vol. 50, no. 1, pp. 53–77, 2017.

[2] E. Bauer and R. Kohavi, "An empirical comparison of voting classification algorithms: Bagging, boosting, and variants," *Machine Learning*, vol. 36, no. 1, pp. 105–139, 1999.

[3] T. G. Dietterich, "An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization," *Machine Learning*, vol. 40, no. 2, pp. 139–157, 2000.

[4] A. Habrard, J.-P. Peyrache, and M. Sebban, "A new boosting algorithm for provably accurate unsupervised domain adaptation," *Knowledge and Information Systems*, vol. 47, no. 1, pp. 45–73, 2016.

[5] B. X. Wang and N. Japkowicz, "Boosting support vector machines for imbalanced data sets," *Knowledge and Information Systems*, vol. 25, no. 1, pp. 1–20, 2010.

[6] K. Dela Rosa, V. Metsis, and V. Athitsos, "Boosted ranking models: a unifying framework for ranking predictions," *Knowledge and Information Systems*, vol. 30, no. 3, pp. 543–568, 2012.

[7] S. Shalev-Shwartz and Y. Singer, "On the equivalence of weak learnability and linear separability: new relaxations and efficient boosting algorithms," *Machine Learning*, vol. 80, no. 2, pp. 141–163, 2010.

[8] E. Cesario, F. Folino, A. Locane, G. Manco, and R. Ortale, "Boosting text segmentation via progressive classification," *Knowledge and Information Systems*, vol. 15, pp. 285–320, May 2008.

[9] M. J. Hosseini, A. Gholipour, and H. Beigy, "An ensemble of cluster-based classifiers for semi-supervised classification of non-stationary data streams," *Knowledge and Information Systems*, vol. 46, no. 3, pp. 567–597, 2016.

[10] C. Preisach and L. Schmidt-Thieme, "Ensembles of relational classifiers," *Knowledge and Information Systems*, vol. 14, no. 3, pp. 249–272, 2008.

[11] A. L. Prodromidis and S. J. Stolfo, "Cost complexity-based pruning of ensemble classifiers," *Knowledge and Information Systems*, vol. 3, no. 4, pp. 449–469, 2001.

[12] Y. Kim and J. Kim, "Convex hull ensemble machine for regression and classification," *Knowledge and Information Systems*, vol. 6, no. 6, pp. 645–663, 2004.

[13] S. Pan, Y. Zhang, and X. Li, "Dynamic classifier ensemble for positive unlabeled text stream classification," *Knowledge and Information Systems*, vol. 33, no. 2, pp. 267–287, 2012.

[14] A. Fern and R. Givan, "Online ensemble learning: An empirical study," *Machine Learning*, vol. 53, no. 1, pp. 71–109, 2003.

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

[16] Y. Freund, "An adaptive version of the boost by majority algorithm," *Machine Learning*, vol. 43, no. 3, pp. 293–318, 2001.

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

[18] S. Mannor and R. Meir, "On the existence of linear weak learners and applications to boosting," *Machine Learning*, vol. 48, no. 1, pp. 219–251, 2002.

[19] N. Duffy and D. Helmbold, "Boosting methods for regression," *Machine Learning*, vol. 47, no. 2, pp. 153–200, 2002.

[20] 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.

[21] C. K. Reddy and J.-H. Park, "Multi-resolution boosting for classification and regression problems," *Knowledge and Information Systems*, vol. 29, no. 2, pp. 435–456, 2011.

[22] 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.

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

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

[25] J. Arenas-Garcia, L. A. Azpicueta-Ruiz, M. T. M. Silva, V. H. Nascimento, and A. H. Sayed, "Combinations of adaptive filters: Performance and convergence properties," *IEEE Signal Processing Magazine*, vol. 33, pp. 120–140, Jan 2016.

[26] 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.

[27] 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.

[28] Q. Wu, X. Zhou, Y. Yan, H. Wu, and H. Min, "Online transfer learning by leveraging multiple source domains," *Knowledge and Information Systems*, pp. 1–21, 2017.

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

[30] S. Ben-David, E. Kushilevitz, and Y. Mansour, "Online learning versus offline learning," *Machine Learning*, vol. 29, no. 1, pp. 45–63, 1997.

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

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

[33] S.-T. Chen, H.-T. Lin, and C.-J. Lu, "An online boosting algorithm with theoretical justifications," in *ICML*, 2012.

[34] B. C. Civek, D. Kari, . Delibalta, and S. S. Kozat, "Big data signal processing using boosted rls algorithm," in *2016 24th Signal Processing and Communication Application Conference (SIU)*, pp. 1089–1092, May 2016.

[35] P. Malik, "Governing big data: Principles and practices," *IBM J. Res. Dev.*, vol. 57, pp. 1:1–1:1, May 2013.

[36] 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, pp. 5471–5486, Oct 2014.

[37] D. Kari, I. Marivani, I. Delibalta, and S. S. Kozat, "Boosted lms-based piecewise linear adaptive filters," in *2016 24th European Signal Processing Conference (EUSIPCO)*, pp. 1593–1597, Aug 2016.

[38] 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.

[39] 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.

[40] 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.

[41] L. Breiman, "Prediction games and arcing algorithms," 1997.

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

[43] B. Babenko, M. H. Yang, and S. Belongie, "A family of online boosting algorithms," in *Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on*, pp. 1346–1353, Sept 2009.

[44] A. Beygelzimer, S. Kale, and H. Luo, "Optimal and adaptive algorithms for online boosting," *International Conference on Machine Learning (ICML)*, pp. 2323–2331, 2015.

[45] Y. Freund, "Boosting a weak learning algorithm by majority," *Inf. Comput.*, vol. 121, pp. 256–285, Sept. 1995.

[46] A. Bertoni, P. Campadelli, and M. Parodi, "A boosting algorithm for regression.," vol. 1327, pp. 343–348, Springer, 1997.

[47] A. Beygelzimer, E. Hazan, S. Kale, and H. Luo, "Online gradient boosting," *Advances in Neural Information Processing Systems (NIPS)*, pp. 2458–2466, 2015.

[48] 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.

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

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

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

[52] 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.

[53] 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.

[54] F. Khan, D. Kari, I. A. Karatepe, and S. S. Kozat, "Universal nonlinear regression on high dimensional data using adaptive hierarchical trees," *IEEE Transactions on Big Data*, vol. 2, pp. 175–188, June 2016.

[55] S. S. Kozat, A. C. Singer, and G. C. Zeitler, "Universal piecewise linear prediction via context trees," *IEEE Transactions on Signal Processing*, vol. 55, pp. 3730–3745, July 2007.

[56] E. Hazan, A. Agarwal, and S. Kale, "Logarithmic regret algorithms for online convex optimization," *Mach. Learn.*, vol. 69, pp. 169–192, Dec. 2007.

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

[58] S. Wiggins, *Introduction to Applied Nonlinear Dynamical Systems and Chaos*. Springer New York, 2003.

[59] L. Torgo, "Regression data sets."

[60] M. Lichman, "UCI machine learning repository," 2013.

[61] F. Pereira, P. Machado, E. Costa, and A. Cardoso, *Progress in Artificial Intelligence: 17th Portuguese Conference on Artificial Intelligence, EPIA 2015, Coimbra, Portugal*. Lecture Notes in Computer Science, Springer International Publishing, 2015.

[62] T. Bertin-Mahieux, D. P. Ellis, B. Whitman, and P. Lamere, "The million song dataset," in *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011)*, 2011.

Appendix A

Proofs

A.1 Proof of Lemma 1.

We observe that according to Algorithm 1,

$$l_t^{(M+1)} = \sum_{k=1}^{M} [\sigma_m^2 - (e_t^{(k)})^2],$$
$$e_t = \frac{1}{M} \sum_{k=1}^{M} e_t^{(k)},$$

In addition, we have

$$\sum_{k=1}^{M} \left(e_t^{(k)}\right)^2 \geq \frac{1}{M} \left(\sum_{k=1}^{M} e_t^{(k)}\right)^2,$$

and as a result, if $e_t^2 > \sigma_m^2$, then $l_t^{(M+1)} \leq 0$, i.e., $\lambda_t^{(M+1)} = 1$. Hence by defining a *modified* desired MSE as $\sigma_m^2 \triangleq \frac{\sigma_d^2 - 4\kappa}{1-\kappa}$, and $\boldsymbol{z}_t = [1/M, ..., 1/M]$ for $t = 1, ..., T$, we have

$$\begin{aligned} \frac{|\{t : e_t^2 > \sigma_m^2\}|}{T} &= \frac{|\{t : \lambda_t^{(M+1)} = 1\}|}{T} \\ &\leq \frac{\sum_{t=1}^{T} \lambda_t^{(M+1)}}{T} \\ &\leq \kappa. \end{aligned}$$

Finally we have

$$\begin{aligned}\frac{\sum_{t=1}^{T} e_t^2}{T} &= \frac{\sum_{t;e_t^2\leq\sigma_m^2} e_t^2}{T} + \frac{\sum_{t;e_t^2>\sigma_m^2} e_t^2}{T} \\ &\leq \frac{\sum_{t;e_t^2\leq\sigma_m^2} \sigma_m^2}{T} + \frac{\sum_{t;e_t^2>\sigma_m^2} 4}{T} \\ &\leq (1-\kappa)\sigma_m^2 + 4\kappa \\ &= \sigma_d^2.\end{aligned}$$

This completes the proof of Lemma 1. □

A.2 Proof of Lemma 2.

We have

$$\sum_{t=1}^{T}\sum_{k=1}^{M} \lambda_t^{(k)} \left[1 - \left(e_t^{(k)}\right)^2\right] \geq (1-4\sigma^2) \sum_{t=1}^{T}\sum_{k=1}^{M} \lambda_t^{(k)}.$$

Moreover, since $0 \geq -\left(e_t^{(k)}\right)^2 = l_t^{(k+1)} - l_t^{(k)} - \sigma_m^2 \geq -4$, following the similar lines as the proof of Lemma 5 in [42], we find that

$$\sum_{t=1}^{T}\sum_{k=1}^{M} \lambda_t^{(k)} \left[1 - \left(e_t^{(k)}\right)^2\right] \leq -\sigma^4\sigma_m^2 \sum_{t=1}^{T}\sum_{k=1}^{M} \lambda_t^{(k)} + \frac{1}{\sigma\ln(1/\sigma)}.$$

Since $\sum_{t=1}^{T}\sum_{k=1}^{M} \lambda_t^{(k)} \geq \kappa TM$, we conclude that

$$M \leq \frac{1}{(\kappa\sigma\ln(1/\sigma))(1-4\sigma^2+\sigma^4\sigma_m^2)}.$$

This concludes the proof of Lemma 2. □