A Comprehensive Approach to Universal Piecewise Nonlinear Regression Based on Trees

N. Denizcan Vanli and Suleyman Serdar Kozat, *Senior Member, IEEE*

Abstract—In this paper, we investigate adaptive nonlinear regression and introduce tree based piecewise linear regression algorithms that are highly efficient and provide significantly improved performance with guaranteed upper bounds in an individual sequence manner. We use a tree notion in order to partition the space of regressors in a nested structure. The introduced algorithms adapt not only their regression functions but also the complete tree structure while achieving the performance of the "best" linear mixture of a doubly exponential number of partitions, with a computational complexity only polynomial in the number of nodes of the tree. While constructing these algorithms, we also avoid using any artificial "weighting" of models (with highly data dependent parameters) and, instead, directly minimize the final regression error, which is the ultimate performance goal. The introduced methods are generic such that they can readily incorporate different tree construction methods such as random trees in their framework and can use different regressor or partitioning functions as demonstrated in the paper.

Index Terms—Nonlinear regression, nonlinear adaptive filtering, binary tree, universal, adaptive.

I. INTRODUCTION

NONLINEAR adaptive filtering and regression are extensively investigated in the signal processing [1]–[19] and machine learning literatures [20]–[23], especially for applications where linear modeling [24], [25] is inadequate, hence, does not provide satisfactory results due to the structural constraint on linearity. Although nonlinear approaches can be more powerful than linear methods in modeling, they usually suffer from overfitting, stability and convergence issues [1], [26]–[28], which considerably limit their application to signal processing problems. These issues are especially exacerbated in adaptive filtering due to the presence of feedback, which is even hard to control for linear models [26], [27], [29]. Furthermore, for applications involving big data, which require to process input vectors with considerably large dimensions, nonlinear models are usually avoided due to unmanageable computational complexity increase [30]. To overcome these difficulties, "tree" based nonlinear adaptive filters or regressors
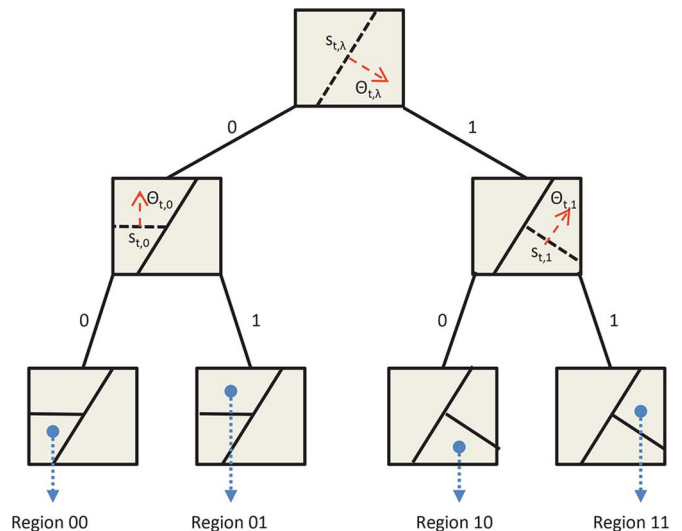
Fig. 1. The partitioning of a two dimensional regressor space using a complete tree of depth-2 with hyperplanes for separation. The whole regressor space is first bisected by $s_{t,\lambda}$, which is defined by the hyperplane $\boldsymbol{\theta}_{t,\lambda}$, where the region on the direction of $\boldsymbol{\theta}_{t,\lambda}$ vector corresponds to the child with "1" label. We then continue to bisect children regions using $s_{t,0}$ and $s_{t,1}$, defined by $\boldsymbol{\theta}_{t,0}$ and $\boldsymbol{\theta}_{t,1}$, respectively.

are introduced as elegant alternatives to linear models since these highly efficient methods retain the breadth of nonlinear models while mitigating the overfitting and convergence issues [2], [4], [30]–[33].

In its most basic form, a regression tree defines a hierarchical or nested partitioning of the regressor space [2]. As an example, consider the binary tree in Fig. 1, which partitions a two dimensional regressor space. On this tree, each node represents a bisection of the regressor space, e.g., using hyperplanes for separation, resulting a complete nested and disjoint partition of the regressor space. After the nested partitioning is defined, the structure of the regressors in each region can be chosen as desired, e.g., one can assign a linear regressor in each region yielding an overall piecewise linear regressor. In this sense, tree based regression is a natural nonlinear extension to linear modeling, in which the space of regressors is partitioned into a union of disjoint regions where a different regressor is trained. This nested architecture not only provides an efficient and tractable structure, but also is shown to easily accommodate to the intrinsic dimension of data, naturally alleviating the overfitting issues [30], [34].

Although nonlinear regressors using decision trees are powerful and efficient tools for modeling, there exist several algorithmic preferences and design choices that affect their performance in real life applications [2], [4], [31]. Especially their
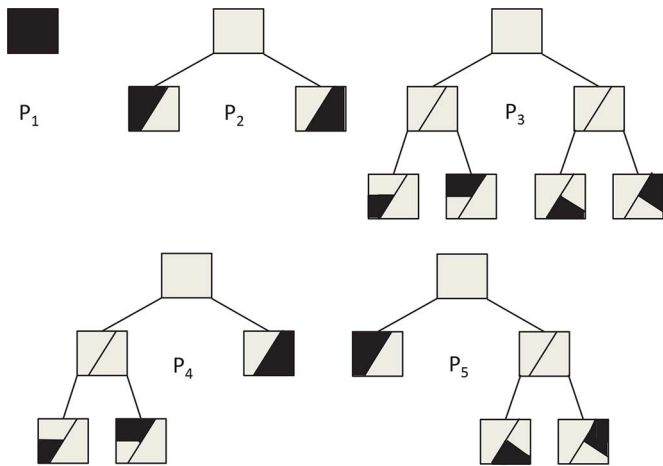
Fig. 2. All different partitions of the regressor space that can be obtained using a depth-2 tree. Any of these partition can be used to construct a piecewise linear model, which can be adaptively trained to minimize the regression error. These partitions are based on the separation functions shown in Fig. 1.

adaptive learning performance may greatly suffer if the algorithmic parameters are not tuned carefully, which is particularly hard to accommodate for applications involving nonstationary data exhibiting saturation effects, threshold phenomena or chaotic behavior [4]. In particular, the success of the tree based regressors heavily depends on the "careful" partitioning of the regressor space. Selection of a good partition, including its depth and regions, from the hierarchy is essential to balance the bias and variance of the regressor [30], [34]. As an example, even for a uniform binary tree, while increasing the depth of the tree improves the modeling power, such an increase usually results in overfitting [4]. There exist numerous approaches that provide "good" partitioning of the regressor space that are shown to yield satisfactory results on the average under certain statistical assumptions on the data or on the application [30].

We note that on the other extreme, there exist methods in adaptive filtering and computational learning theory, which avoid such a direct commitment to a particular partitioning but instead construct a weighted average of all possible piecewise models defined on a tree [4], [35], [36]. Note that a full binary tree of depth-d, as shown in Fig. 1 for $d = 2$, with hard separation boundaries, defines a doubly exponential number [37] of complete partition of the regressor space (see Fig. 2). Each such partitioning of the regressor space is represented by the collection of the nodes of the full tree where each node is assigned to a particular region of the regressor space. Any of these partitions can be used to construct and then train a piecewise linear or nonlinear regressor. Instead of fixing one of these partitions, one can run all the models (or subtrees) in parallel and combine the final outputs based on their performance. Such approaches are shown to mitigate the bias variance trade off in a deterministic framework [4], [35], [36]. However, these methods are naturally constraint to work on a specific tree or partitionings, i.e., the tree is fixed and cannot be adapted to the data, and the weighting among the models usually have no theoretical justifications (although they may be inspired from information theoretic considerations [38]). As an example, the "universal weighting" coefficients in [4], [24], [39]–[41] or

the exponentially weighted performance measure are defined based on algorithmic concerns and provide universal bounds, however, do not minimize the final regression error. In particular, the performance of these methods highly depends on these weighting coefficients and algorithmic parameters that should be tuned to the particular application for successful operation [4], [31].

To this end, we provide a *comprehensive* solution to nonlinear regression using decision trees. In this paper, we introduce algorithms that are shown *i*) to be highly efficient *ii*) to provide significantly improved performance over the state of the art approaches in different applications *iii*) to have guaranteed performance bounds without any statistical assumptions. Our algorithms not only adapt the corresponding regressors in each region, but also learn the corresponding region boundaries, as well as the "best" linear mixture of a doubly exponential number of partitions to minimize the final estimation or regression error. We introduce algorithms that are guaranteed to achieve the performance of the best linear combination of a doubly exponential number of models with a significantly reduced computational complexity. The introduced approaches significantly outperform [4], [24], [39] based on trees in different applications in our examples, since we avoid any artificial weighting of models with highly data dependent parameters and, instead, "directly" minimize the final error, which is the ultimate performance goal. Our methods are generic such that they can readily incorporate random projection (RP) or k-d trees in their framework as commented in our simulations, e.g., the RP trees can be used as the starting partitioning to adaptively learn the tree, regressors and weighting to minimize the final error as data progress.

In this paper, we first introduce an algorithm that asymptotically achieves the performance of the "best" linear combination of a doubly exponential number of different models that can be represented by a depth-d tree a with fixed regressor space partitioning with a computational complexity only linear in the number of nodes of the tree. We then provide a guaranteed upper bound on the performance of this algorithm and prove that as the data length increases, this algorithm achieves the performance of the "best" linear combination of a doubly exponential number of models without any statistical assumptions. Furthermore, even though we refrain from any statistical assumptions on the underlying data, we also provide the mean squared performance of this algorithm compared to the mean squared performance of the best linear combination of the mixture. These methods are generic and truly sequential such that they do not need any a priori information, e.g., upper bounds on the data [2], [4], (such upper bounds does not hold in general, e.g., for Gaussian data). Although the combination weights in [4], [35], [36] are artificially constraint to be positive and sum up to 1 [42], we have no such restrictions and directly adapt to the data without any constraints. We then extend these results and provide the final algorithm (with a slightly increased computational complexity), which "adaptively" learns also the corresponding regions of the tree to minimize the final regression error. This approach learns *i*) the "structure" of the tree, *ii*) the regressors in each region, and *iii*) the linear combination weights to merge all possible partitions, to minimize the final regression error. In this sense, this algorithm can readily capture the salient characteristics of the underlying data while avoiding bias to a particular model or structure.

In Section III, we first present an algorithm with a fixed regressor space partitioning and present a guaranteed upper bound on its performance. We then significantly reduce the computational complexity of this algorithm using the tree structure. In Section IV, we extend these results and present the final algorithm that adaptively learns the tree structure, region boundaries, region regressors and combination weights to minimize the final regression error. We then demonstrate the performance of our algorithms through simulations in Section V. We then finalize our paper with concluding remarks.

II. PROBLEM DESCRIPTION

In this paper, all vectors are column vectors and denoted by boldface lower case letters. Matrices are represented by boldface uppercase letters. For a vector $\boldsymbol{u}$, $\|\boldsymbol{u}\| = \sqrt{\boldsymbol{u}^T \boldsymbol{u}}$ is the ℓ^2-norm, where $\boldsymbol{u}^T$ is the ordinary transpose. Here, $\boldsymbol{I}_k$ represents a $k \times k$ dimensional identity matrix.

We study sequential nonlinear regression, where we observe a desired signal $\{d_t\}_{t \geq 1}$, $d_t \in \mathbb{R}$, and regression vectors $\{\boldsymbol{x}_t\}_{t \geq 1}$, $\boldsymbol{x}_t \in \mathbb{R}^m$, such that we sequentially estimate d_t by

$$\hat{d}_t = f_t(\boldsymbol{x}_t),$$

and $f_t(\cdot)$ is an adaptive nonlinear regression function. At each time t, the regression error is given by

$$e_t = d_t - \hat{d}_t.$$

Although there exist several different approaches to select the corresponding nonlinear regression function, we particularly use piecewise models such that the space of the regression vectors, i.e., $\boldsymbol{x}_t \in \mathbb{R}^m$, is adaptively partitioned using hyperplanes based on a tree structure. We also use adaptive linear regressors in each region. However, our framework can be generalized to any partitioning of the regression space, i.e., not necessarily using hyperplanes, such as using [30], or any regression function in each region, i.e., not necessarily linear. Furthermore, both the region boundaries as well as the regressors in each region are adaptive.

A. A Specific Partition on a Tree

To clarify the framework, suppose the corresponding space of regressor vectors is two dimensional, i.e., $\boldsymbol{x}_t \in \mathbb{R}^2$, and we partition this regressor space using a depth-2 tree as in Fig. 1. A depth-2 tree is represented by three separating functions $s_{t,\lambda}$, $s_{t,0}$ and $s_{t,1}$, which are defined using three hyperplanes with direction vectors $\boldsymbol{\theta}_{t,\lambda}$, $\boldsymbol{\theta}_{t,0}$ and $\boldsymbol{\theta}_{t,1}$, respectively (See Fig. 1). Due to the tree structure, three separating hyperplanes generate only four regions, where each region is assigned to a leaf on the tree given in Fig. 1 such that the partitioning is defined in a hierarchical manner, i.e., $\boldsymbol{x}_t$ is first processed by $s_{t,\lambda}$ and then by $s_{t,i}$, $i = 0, 1$. A complete tree defines a doubly exponential number, $O(2^{2^d})$, of subtrees each of which can also be used to partition the space of past regressors. As an example, a depth-2 tree defines 5 different subtrees or partitions as shown in Fig. 2, where each of these subtrees is constructed using the leaves and the nodes of the original tree. Note that a node of the tree represents a region which is the union of regions assigned to its left and right children nodes [38].

The corresponding separating (indicator) functions can be hard, e.g., $s_t = 1$ if the data falls into the region pointed by the direction vector $\boldsymbol{\theta}_t$, and $s_t = 0$ otherwise. Without loss of generality, the regions pointed by the direction vector $\boldsymbol{\theta}_t$ are labeled as "1" regions on the tree in Fig. 1. The separating functions can also be soft. As an example, we use the logistic regression classifier [43]

$$s_t = \frac{1}{1 + e^{\boldsymbol{x}_t^T \boldsymbol{\theta}_t + b_t}} \tag{1}$$

as the soft separating function, where $\boldsymbol{\theta}_t$ is the direction vector and b_t is the offset, describing a hyperplane in the m-dimensional regressor space. With an abuse of notation we combine the direction vector $\boldsymbol{\theta}_t$ with the offset parameter b_t and denote it by $\boldsymbol{\theta}_t = [\boldsymbol{\theta}_t; b_t]$. Then the separator function in (1) can be rewritten as

$$s_t = \frac{1}{1 + e^{\boldsymbol{x}_t^T \boldsymbol{\theta}_t}}, \tag{2}$$

where $\boldsymbol{x}_t = [\boldsymbol{x}_t; 1]$. One can easily use other differentiable soft separating functions in this setup in a straightforward manner as remarked later in the paper.

To each region, we assign a regression function to generate an estimate of d_t. For a depth-2 (or a depth-d) tree, there are 7 (or $2^{d+1} - 1$) nodes (including the leaves) and 7 (or $2^{d+1} - 1$) regions corresponding to these nodes, where the combination of these nodes or regions form a complete partition. In this paper, we assign linear regressors to each region. For instance consider the third model in Fig. 2, i.e., P_3, where this partition is the union of 4 regions each corresponding to a leaf of the original complete tree in Fig. 1, labeled as 00, 01, 10, and 11. The P_3 defines a complete partitioning of the regressor space, hence can be used to construct a piecewise linear regressor. At each region, say the 00th region, we generate the estimate

$$\hat{d}_{t,00} = \boldsymbol{x}_t^T \boldsymbol{v}_{t,00}, \tag{3}$$

where $\boldsymbol{v}_{t,00} \in \mathbb{R}^m$ is the linear regressor vector assigned to region 00. Considering the hierarchical structure of the tree and having calculated the region estimates, the final estimate of P_3 is given by

$$\hat{d}_t = s_{t,\lambda} s_{t,0} \hat{d}_{t,00} + s_{t,\lambda}(1 - s_{t,0}) \hat{d}_{t,01}$$
$$+ (1 - s_{t,\lambda}) s_{t,1} \hat{d}_{t,10} + (1 - s_{t,\lambda})(1 - s_{t,1}) \hat{d}_{t,11}, \tag{4}$$

for any $\boldsymbol{x}_t$. We emphasize that any P_i, $i = 1, \ldots, 5$ can be used in a similar fashion to construct a piecewise linear regressor.

Continuing with the specific partition P_3, we adaptively train the region boundaries and regressors to minimize the final regression error. As an example, if we use a stochastic gradient descent algorithm [42], [44]–[46], we update the regressor of the node "00" as

$$\boldsymbol{v}_{t+1,00} = \boldsymbol{v}_{t,00} - \frac{1}{2} \mu_t \nabla e_t^2$$
$$= \boldsymbol{v}_{t,00} + \mu_t e_t s_{t,\lambda} s_{t,0} \boldsymbol{x}_t,$$

where μ_t is the step size to update the region regressors. Similarly, region regressors can be updated for all regions $r = 00, 01, 10, 11$. Separator functions can also be trained using the same approach, e.g., the separating function of the node 0, $s_{t,0}$, can be updated as

$$\boldsymbol{\theta}_{t+1,0} = \boldsymbol{\theta}_{t,0} - \frac{1}{2} \eta_t \nabla e_t^2$$
$$= \boldsymbol{\theta}_{t,0} + \eta_t e_t (s_{t,\lambda} \hat{d}_{t,00} - s_{t,\lambda} \hat{d}_{t,01}) \frac{\partial s_{t,0}}{\partial \boldsymbol{\theta}_{t,0}},$$

where η_t is the step size to update the separator functions and

$$\frac{\partial s_{t,0}}{\partial \boldsymbol{\theta}_{t,0}} = \frac{-\boldsymbol{x}_t e^{\boldsymbol{x}_t^T \boldsymbol{\theta}_{t,0}}}{\left(1 + e^{\boldsymbol{x}_t^T \boldsymbol{\theta}_{t,0}}\right)^2}, \quad (5)$$

according to the separator function in (2). Other separating functions (different than the logistic regressor classifier) can also be trained in a similar fashion by simply calculating the gradient with respect to the extended direction vector and plugging in (5).

Until now a specific partition, i.e., P_3, is used to construct a piecewise linear regressor, although the tree can represent P_i, $i = 1, \ldots, 5$. However, since the data structure is unknown, one may not prefer a particular model [4], [35], [36], i.e., there may not be a specific best model or the best model can change in time. As an example, the simpler models, e.g., P_1, may perform better while there is not sufficient data at the start of training and the finer models, e.g., P_3, can recover through the learning process. Hence, we hypothetically construct all doubly exponential number of piecewise linear regressors corresponding to all partitions (see Fig. 2) and then calculate an adaptive linear combination of the outputs of all, while these algorithms learn the region boundaries as well as the regressors in each region.

In Section III, we first consider the scenario in which the regressor space is partitioned using hard separator functions and combine $O(2^{2^d})$ different models for a depth-d tree with a computational complexity $O(d2^d)$. In Section IV, we partition the regressor space with soft separator functions and adaptively update the region boundaries to achieve the best partitioning of the m-dimensional regressor space with a computational complexity $O(m4^d)$.

III. REGRESSOR SPACE PARTITIONING VIA HARD SEPARATOR FUNCTIONS

In this section, we consider the regression problem in which the sequential regressors (as described in Section II.A) for all partitions in the doubly exponential tree class are combined when hard separation functions are used, i.e., $s_t \in \{0, 1\}$. In this section, the hard boundaries are not trained, however, both the regressors of each region and the combination parameters to merge the outputs of all partitions are trained. To partition the regressor space, we first construct a tree with an arbitrary depth, say a tree of depth-d, and denote the number of different models of this class by $\beta_d \approx (1.5)^{2^d}$, e.g., one can use RP trees as the starting tree [30]. While the kth model (i.e., P_k partition) generates the regression output $\hat{d}_t^{(k)}$ at time t for all $k = 1, \ldots, \beta_d$, we linearly combine these estimates using the weighting vector $\boldsymbol{w}_t \triangleq [w_t^{(1)}, \ldots, w_t^{(\beta_d)}]^T$ such that the final estimate of our algorithm at time t is given as

$$\hat{d}_t \triangleq \sum_{k=1}^{\beta_d} w_t^{(k)} \hat{d}_t^{(k)} = \boldsymbol{w}_t^T \hat{\boldsymbol{d}}_t, \quad (6)$$

where $\hat{\boldsymbol{d}}_t \triangleq [\hat{d}_t^{(1)}, \ldots, \hat{d}_t^{(\beta_d)}]^T$. The regression error at time t is calculated as

$$e_t(\boldsymbol{w}_t) \triangleq d_t - \hat{d}_t = d_t - \boldsymbol{w}_t^T \hat{\boldsymbol{d}}_t.$$

For β_d different models that are embedded within a depth-d tree, we introduce an algorithm (given in Algorithm 1) that asymptotically achieves the same cumulative squared regression error as the optimal linear combination of these models without any statistical assumptions. This algorithm is constructed in the proof of the following theorem and the computational complexity of the algorithm is only linear in the number of nodes of the tree.

Theorem 1: Let $\{d_t\}_{t \geq 1}$ and $\{\boldsymbol{x}_t\}_{t \geq 1}$ be arbitrary, bounded, and real-valued sequences. The algorithm $\hat{d}_t$ given in Algorithm 1 when applied to these data sequence yields

$$\sum_{t=1}^n (d_t - \hat{d}_t)^2 - \min_{\boldsymbol{w} \in \mathbb{R}^{\beta_d}} \sum_{t=1}^n (d_t - \boldsymbol{w}^T \hat{\boldsymbol{d}}_t)^2 \leq O(\ln(n)), \quad (7)$$

for all n, when $e_t^2(\boldsymbol{w})$ is strongly convex $\forall t$, where $\hat{\boldsymbol{d}}_t = [\hat{d}_t^{(1)}, \ldots, \hat{d}_t^{(\beta_d)}]^T$, and $\hat{d}_t^{(k)}$ are the estimates of d_t at time t for $k = 1, \ldots, \beta_d$.

This theorem implies that our algorithm (given in Algorithm 1), asymptotically achieves the performance of the best combination of the outputs of $O(2^{2^d})$ different models that can be represented using a depth-d tree with a computational complexity $O(d2^d)$. Note that as given in Algorithm 1, no a priori information, e.g., upper bounds, on the data is used to construct the algorithm. Furthermore, the algorithm can use different regressors, e.g., [4], or regions seperation functions, e.g., [30], to define the tree.

Assuming that the constituent partition regressors converge to stationary distributions, such as for Gaussian regressors, and under widely used separation assumptions [26], [47] such that the expectation of $\hat{d}_t^{(k)}$, $k = 1, \ldots, \beta_d$, and $\boldsymbol{w}_t$ are separable, we have the following theorem.

Theorem 2: Assuming that the partition regressors, i.e., $\hat{d}_t^{(k)}$, $k = 1, \ldots, \beta_d$, and d_t converge to zero mean stationary distributions, we have

$$\lim_{t \to \infty} E\left[e_t^2\right] = J^* + \frac{\mu J^* \mathrm{tr}(\boldsymbol{D})}{2 - \mu \mathrm{tr}(\boldsymbol{D})},$$

where μ is the learning rate of the stochastic gradient update,

$$J^* \triangleq \min_{\boldsymbol{w} \in \mathbb{R}^{\beta_d}} \lim_{t \to \infty} E[(d_t - \boldsymbol{w}^T \hat{\boldsymbol{d}}_t)^2],$$
$$\text{and}$$
$$\boldsymbol{D} \triangleq \lim_{t \to \infty} E\left[\hat{\boldsymbol{d}}_t \hat{\boldsymbol{d}}_t^T\right],$$

for the algorithm $\hat{d}_t$ (given in Algorithm 1).

Theorem 2 directly follows Chapter 6 of [26] since we use a stochastic gradient algorithm to merge the partition regressors [26], [47]. Hence, the introduced algorithm may also achieve the mean square error performance of the best linear combination of the constituent piecewise regressors if μ is selected carefully.

A. Proof of Theorem 1 and Construction of Algorithm 1

To construct the final algorithm, we first introduce a "direct" algorithm which achieves the corresponding bound in Theorem 1. This direct algorithm has a computational complexity $O(2^{2^d})$ since one needs to calculate the correlation information of $O(2^{2^d})$ models to achieve the performance of the best linear combination. We then introduce a specific labeling technique

and using the properties of tree structure, construct an algorithm to obtain the same upper bound as the "direct" algorithm, yet with a significantly smaller computational complexity, i.e., $O(d2^d)$.

For a depth-d tree, suppose $\hat{d}_t^{(k)}$, $k = 1, \ldots, \beta_d$, are obtained as described in Section II.A. To achieve the upper bound in (7), we use the stochastic gradient descent approach and update the combination weights as

$$
\begin{aligned}
\boldsymbol{w}_{t+1} &= \boldsymbol{w}_t - \frac{1}{2}\mu_t \nabla e_t^2(\boldsymbol{w}_t) \\
&= \boldsymbol{w}_t + \mu_t e_t \hat{\boldsymbol{d}}_t,
\end{aligned} \tag{8}
$$

where μ_t is the step-size parameter (or the learning rate) of the gradient descent algorithm. We first derive an upper bound on the sequential learning regret R_n, which is defined as

$$
R_n \triangleq \sum_{t=1}^{n} e_t^2(\boldsymbol{w}_t) - \sum_{t=1}^{n} e_t^2(\boldsymbol{w}_n^*),
$$

where $\boldsymbol{w}_n^*$ is the optimal weight vector over n, i.e.,

$$
\boldsymbol{w}_n^* \triangleq \arg\min_{\boldsymbol{w} \in \mathbb{R}^{\beta_d}} \sum_{t=1}^{n} e_t^2(\boldsymbol{w}).
$$

Following [44], using Taylor series approximation, for some point $\boldsymbol{z}_t$ on the line segment connecting $\boldsymbol{w}_t$ to $\boldsymbol{w}_n^*$, we have

$$
\begin{aligned}
e_t^2(\boldsymbol{w}_n^*) = e_t^2(\boldsymbol{w}_t) &+ \left(\nabla e_t^2(\boldsymbol{w}_t)\right)^T (\boldsymbol{w}_n^* - \boldsymbol{w}_t) \\
&+ \frac{1}{2}(\boldsymbol{w}_n^* - \boldsymbol{w}_t)^T \nabla^2 e_t^2(\boldsymbol{z}_t)(\boldsymbol{w}_n^* - \boldsymbol{w}_t).
\end{aligned} \tag{9}
$$

According to the update rule in (8), at each iteration the update on weights are performed as $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \frac{\mu_t}{2}\nabla e_t^2(\boldsymbol{w}_t)$. Hence, we have

$$
\begin{aligned}
\|\boldsymbol{w}_{t+1} - \boldsymbol{w}_n^*\|^2 &= \left\| \boldsymbol{w}_t - \frac{\mu_t}{2}\nabla e_t^2(\boldsymbol{w}_t) - \boldsymbol{w}_n^* \right\|^2 \\
&= \|\boldsymbol{w}_t - \boldsymbol{w}_n^*\|^2 - \mu_t \left(\nabla e_t^2(\boldsymbol{w}_t)\right)^T (\boldsymbol{w}_t - \boldsymbol{w}_n^*) \\
&+ \frac{\mu_t^2}{4}\left\|\nabla e_t^2(\boldsymbol{w}_t)\right\|^2.
\end{aligned}
$$

Then we obtain

$$
\begin{aligned}
\left(\nabla e_t^2(\boldsymbol{w}_t)\right)^T (\boldsymbol{w}_t - \boldsymbol{w}_n^*) &= \frac{\|\boldsymbol{w}_t - \boldsymbol{w}_n^*\|^2 - \|\boldsymbol{w}_{t+1} - \boldsymbol{w}_n^*\|^2}{\mu_t} \\
&+ \mu_t \frac{\left\|\nabla e_t^2(\boldsymbol{w}_t)\right\|^2}{4}.
\end{aligned} \tag{10}
$$

Under the mild assumptions that $\|\nabla e_t^2(\boldsymbol{w}_t)\|^2 \leq A^2$ for some $A > 0$ and $e_t^2(\boldsymbol{w}_n^*)$ is λ-strong convex for some $\lambda > 0$ [44], we achieve the following upper bound

$$
\begin{aligned}
e_t^2(\boldsymbol{w}_t) - e_t^2(\boldsymbol{w}_n^*) &\leq \frac{\|\boldsymbol{w}_t - \boldsymbol{w}_n^*\|^2 - \|\boldsymbol{w}_{t+1} - \boldsymbol{w}_n^*\|^2}{\mu_t} \\
&- \frac{\lambda}{2}\|\boldsymbol{w}_t - \boldsymbol{w}_n^*\|^2 + \mu_t \frac{A^2}{4}.
\end{aligned} \tag{11}
$$

By selecting $\mu_t = 2/(\lambda t)$ and summing up the regret terms in (11), we get

$$
\begin{aligned}
R_n &= \sum_{t=1}^{n} \left\{ e_t^2(\boldsymbol{w}_t) - e_t^2(\boldsymbol{w}_n^*) \right\} \\
&\leq \sum_{t=1}^{n} \|\boldsymbol{w}_t - \boldsymbol{w}_n^*\|^2 \left(\frac{1}{\mu_t} - \frac{1}{\mu_{t-1}} - \frac{\lambda}{2} \right) + \frac{A^2}{4}\sum_{t=1}^{n}\mu_t \\
&= \frac{A^2}{4}\sum_{t=1}^{n}\frac{2}{\lambda t} \\
&\leq \frac{A^2}{2\lambda}(1 + \log(n)).
\end{aligned}
$$

Note that (8) achieves the performance of the best linear combination of $O(2^{2^d})$ piecewise linear models that are defined by the tree. However, in this form (8) requires a computational complexity of $O(2^{2^d})$ since the vector $\boldsymbol{w}_t$ has a size of $O(2^{2^d})$. We next illustrate an algorithm that performs the same adaptation in (8) with a complexity of $O(d2^d)$.

We next introduce a labeling for the tree nodes following [38]. The root node is labeled with an empty binary string λ and assuming that a node has a label p, where p is a binary string, we label its upper and lower children as $p1$ and $p0$, respectively. Here we emphasize that a string can only take its letters from the binary alphabet $\{0, 1\}$, where 0 refers to the lower child, and 1 refers to the upper child of a node. We also introduce another concept, i.e., the definition of the prefix of a string. We say that a string $p' = q_1' \cdots q_{l'}'$ is a prefix to string $p = q_1 \ldots q_l$ if $l' \leq l$ and $q_i' = q_i$ for all $i = 1, \ldots, l'$, and the empty string λ is a prefix to all strings. Let $\mathcal{P}(p)$ represent all prefixes to the string p, i.e., $\mathcal{P}(p) \triangleq \{\nu_1, \ldots, \nu_{l+1}\}$, where $l \triangleq l(p)$ is the length of the string p, ν_i is the string with $l(\nu_i) = i - 1$, and $\nu_1 = \lambda$ is the empty string, such that the first $i - 1$ letters of the string p forms the string ν_i for $i = 1, \ldots, l + 1$.

We then observe that the final estimate of any model can be found as the combination of the regressors of its leaf nodes. According to the region $\boldsymbol{x}_t$ has fallen, the final estimate will be calculated with the separator functions. As an example, for the second model in Fig. 2 (i.e., P_2 partition), say $\boldsymbol{x}_t \in \mathcal{R}_{00}$, and hard separator functions are used. Then the final estimate of this model will be given as $\hat{d}_t^{(2)} = \hat{d}_{t,0}$. For any separator function, the final estimate of the desired data d_t at time t of the kth model, i.e., $\hat{d}_t^{(k)}$ can be obtained according to the hierarchical structure of the tree as the sum of regressors of its leaf nodes, each of which are scaled by the values of the separator functions of the nodes between the leaf node and the root node. Hence, we can compactly write the final estimate of the kth model at time t as

$$
\hat{d}_t^{(k)} = \sum_{p \in \mathcal{M}_k} \left(\hat{d}_{t,p} \prod_{i=1}^{l(p)} s_{t,\nu_i}^{q_i} \right), \tag{12}
$$

where $\mathcal{M}_k$ is the set of all leaf nodes in the kth model, $\hat{d}_{t,p}$ is the regressor of the node p, $l(p)$ is the length of the string p, $\nu_i \in \mathcal{P}(p)$ is the prefix to string p with length $i - 1$, q_i is the ith letter of the string p, i.e., $\nu_{i+1} = \nu_i q_i$, and finally $s_{t,\nu_i}^{q_i}$ denotes the separator function at node ν_i such that

$$
s_{t,\nu_i}^{q_i} \triangleq \begin{cases} s_{t,\nu_i}, & \text{if } q_i = 0 \\ 1 - s_{t,\nu_i}, & \text{otherwise} \end{cases}
$$

with s_{t,ν_i} defined as in (2). We emphasize that we dropped p-dependency of q_i and ν_i to simplify notation.

As an example, if we consider the third model P_3 in Fig. 2 as the kth model (i.e., $k = 3$), where $\mathcal{M}_k = \{00, 01, 10, 11\}$, then we can calculate the final estimate of that model as follows

$$
\begin{aligned}
\hat{d}_t^{(k)} &= \sum_{p \in \mathcal{M}_k} \left(\hat{d}_{t,p} \prod_{i=1}^{l(p)} s_{t,\nu_i}^{q_i} \right) \\
&= \hat{d}_{t,00} s_{t,0}^0 s_{t,\lambda}^0 + \hat{d}_{t,01} s_{t,0}^1 s_{t,\lambda}^0 \\
&\quad + \hat{d}_{t,10} s_{t,1}^0 s_{t,\lambda}^1 + \hat{d}_{t,11} s_{t,1}^1 s_{t,\lambda}^1 \\
&= \hat{d}_{t,00} s_{t,0} s_{t,\lambda} + \hat{d}_{t,01} (1 - s_{t,0}) s_{t,\lambda} \\
&\quad + \hat{d}_{t,10} s_{t,1} (1 - s_{t,\lambda}) + \hat{d}_{t,11} (1 - s_{t,1})(1 - s_{t,\lambda}). \quad (13)
\end{aligned}
$$

Note that (4) and (13) are the same special cases of (12).

We next denote the product terms in (12) as follows

$$
\hat{\delta}_{t,p} \triangleq \hat{d}_{t,p} \prod_{i=1}^{l(p)} s_{t,\nu_i}^{q_i}, \quad (14)
$$

to simplify the notation. Here, $\hat{\delta}_{t,p}$ can be viewed as the estimate of the node (i.e., region) p given that $\boldsymbol{x}_t \in \mathcal{R}_{p'}$ for some $p' \in \mathcal{L}_d$, where $\mathcal{L}_d$ denotes all leaf nodes of the depth-d tree class, i.e., $\mathcal{L}_d \triangleq \{p : l(p) = d\}$. Then (12) can be rewritten as follows

$$
\hat{d}_t^{(k)} = \sum_{p \in \mathcal{M}_k} \hat{\delta}_{t,p}.
$$

Since we now have a compact form to represent the tree and the outputs of each partition, we next introduce a method to calculate the combination weights of $O(2^{2^d})$ piecewise regressor outputs in a simplified manner.

To this end, we assign a particular linear weight to each node. We denote the weight of node p at time t as $w_{t,p}$ and then we define the weight of the kth model as the sum of weights of its leaf nodes, i.e.,

$$
w_t^{(k)} = \sum_{p \in \mathcal{M}_k} w_{t,p},
$$

for all $k = 1, \ldots, \beta_d$. Since the weight of each model, say model k, is recursively updated as

$$
w_{t+1}^{(k)} = w_t^{(k)} + \mu_t e_t \hat{d}_t^{(k)},
$$

we achieve the following recursive update on the node weights

$$
w_{t+1,p} \triangleq w_{t,p} + \mu_t e_t \hat{\delta}_{t,p}, \quad (15)
$$

where $\hat{\delta}_{t,p}$ is defined as in (14).

This result implies that instead of managing $O(2^{2^d})$ memory locations, and making $O(2^{2^d})$ calculations, only keeping track of the weights of every node is sufficient, and the number of nodes in a depth-d model is $|\mathcal{N}_d| = 2^{d+1} - 1$, where $\mathcal{N}_d$ denotes the set of all nodes in a depth-d tree. As an example, for $d = 2$ we obtain $\mathcal{N}_d = \{\lambda, 0, 1, 00, 01, 10, 11\}$. Therefore we can reduce the storage and computational complexity from $O(2^{2^d})$ to $O(2^d)$ by performing the update in (15) for all $p \in \mathcal{N}_d$. We then

continue the discussion with the update of weights performed at each time t when hard separator functions are used.

Without loss of generality assume that at time t, the regression vector $\boldsymbol{x}_t$ has fallen into the region $\mathcal{R}_{p'}$ specified by the node $p' \in \mathcal{L}_d$. Consider the node regressor defined in (14) for some node $p \in \mathcal{N}_d$. Since we are using hard separator functions, we obtain

$$
\hat{\delta}_{t,p} = \begin{cases} \hat{d}_{t,p}, & \text{if } p \in \mathcal{P}(p') \\ 0, & \text{otherwise} \end{cases},
$$

where $\mathcal{P}(p')$ represents all prefixes to the string p', i.e., $\mathcal{P}(p') = \{\nu_1', \ldots, \nu_{d+1}'\}$. Then at each time t we only update the weights of the nodes $p \in \mathcal{P}(p')$, hence we only make $|\mathcal{P}(p')| = d + 1$ updates since the hard separation functions are used for partitioning of the regressor space.

Before stating the algorithm that combines these node weights as well as node estimates, and generates the same final estimate as in (6) with a significantly reduced computational complexity, we observe that for a node $p \in \mathcal{N}_d$ with length $l(p) \geq 1$, there exist a total of

$$
\gamma_d(l(p)) \triangleq \prod_{j=1}^{l(p)} \beta_{d-j}
$$

different models in which the node $p \in \mathcal{N}_d$ is a leaf node of that model, where $\beta_0 = 1$ and $\beta_{j+1} = \beta_j^2 + 1$ for all $j \geq 1$. For $l(p) = 0$ case, i.e., for $p = \lambda$, one can clearly observe that there exists only one model having λ as the leaf node, i.e., the model having no partitions, therefore $\gamma_d(0) = 1$.

Having stated how to store all estimates and weights in $O(2^d)$ memory locations, and perform the updates at each iteration, we now introduce an algorithm to combine them in order to obtain the final estimate of our algorithm, i.e., $\hat{d}_t = \boldsymbol{w}_t^T \hat{\boldsymbol{d}}_t$. We emphasize that the sizes of the vectors $\boldsymbol{w}_t$ and $\hat{\boldsymbol{d}}_t$ are $O(2^{2^d})$, which forces us to make $O(2^{2^d})$ computations. We however introduce an algorithm with a complexity of $O(d2^d)$ that is able to achieve the exact same result.

Algorithm 1: Decision Fixed Tree (DFT) Regressor

1: **for** $t = 1$ **to** n **do**
2: $\quad p' \Leftarrow p \in \mathcal{L}_d : \boldsymbol{x}_t \in \mathcal{R}_p$
3: $\quad \hat{d}_t \Leftarrow 0$
4: $\quad$ **for all** $\nu_j' \in \mathcal{P}(p')$ **do**
5: $\quad\quad \hat{d}_{t,\nu_j'} \Leftarrow \boldsymbol{v}_{t,\nu_j'}^T \boldsymbol{x}_t$
6: $\quad\quad \kappa_{t,\nu_j'} \Leftarrow \gamma_d(l(\nu_j)) w_{t,\nu_j'}$
7: $\quad\quad$ **for all** $p \in \mathcal{N}_d - (\mathcal{P}(p') \cup \mathcal{S}_d(p'))$ **do**
8: $\quad\quad\quad \bar{p} \Leftarrow \acute{p} \in \mathcal{P}(p) \cap \mathcal{P}(p') : l(\acute{p}) = |\mathcal{P}(p) \cap \mathcal{P}(p')| - 1$
9: $\quad\quad\quad \kappa_{t,\nu_j'} \Leftarrow \kappa_{t,\nu_j'} + \frac{\gamma_d(l(\nu_j'))\gamma_{d-l(\bar{p})-1}(l(p)-l(\bar{p})-1)}{\beta_{d-l(\bar{p})-1}} w_{t,p}$
10: $\quad\quad$ **end for**
11: $\quad\quad \hat{d}_t \Leftarrow \hat{d}_t + \kappa_{t,\nu_j'} \hat{d}_{t,\nu_j'}$
12: $\quad$ **end for**
13: $\quad e_t \Leftarrow d_t - \hat{d}_t$
14: $\quad$ **for all** $\nu_j' \in \mathcal{P}(p')$ **do**
15: $\quad\quad \boldsymbol{v}_{t+1,\nu_j'} \Leftarrow \boldsymbol{v}_{t,\nu_j'} + \mu_t e_t \boldsymbol{x}_t$
16: $\quad\quad w_{t+1,\nu_j'} \Leftarrow w_{t,\nu_j'} + \mu_t e_t \hat{d}_{t,\nu_j'}$
17: $\quad$ **end for**
18: **end for**

For a depth-d tree, at time t say $\boldsymbol{x}_t \in \mathcal{R}_{p'}$ for a node $p' \in \mathcal{L}_d$. Then the final estimate of our algorithm is found by

$$\hat{d}_t = \sum_{k=1}^{\beta_d} w_t^{(k)} \hat{d}_t^{(k)}$$
$$= \sum_{k=1}^{\beta_d} \sum_{p \in \mathcal{M}_k} w_{t,p} \hat{d}_{t,p_k}, \qquad (16)$$

where $\mathcal{M}_k$ is the set of all leaf nodes in model k, and $p_k \in \mathcal{P}(p')$ is the longest prefix to the string p' in the kth model, i.e., $p_k \triangleq \mathcal{P}(p') \cap \mathcal{M}_k$. Let $\mathcal{P}(p') = \{\nu_1', \ldots, \nu_{d+1}'\}$ denote the set of all prefixes to string p'. We then observe that the regressors of the nodes $\nu_j' \in \mathcal{P}(p')$ will be sufficient to obtain the final estimate of our algorithm. Therefore, we only consider the estimates of $O(d)$ nodes.

In order to further simplify the final estimate in (16), we first let $\mathcal{S}_d(p) \triangleq \{\acute{p} \in \mathcal{N}_d \mid \mathcal{P}(\acute{p}) = p\}$, i.e., $\mathcal{S}_d(p)$ denotes the set of all nodes of a depth-d tree, whose set of prefixes include the node p. As an example, for a depth-2 tree, we have $\mathcal{S}(0) = \{0, 00, 01\}$. We then define a function $\rho(p, \acute{p})$ for arbitrary two nodes $p, \acute{p} \in \mathcal{N}_d$, as the number of models having both p and $\acute{p}$ as its leaf nodes. Trivially, if $\acute{p} = p$, then $\rho(p, p) = \gamma_d(l(p))$. If $p \neq \acute{p}$, then letting $\bar{p}$ denote the longest prefix to both p and $\acute{p}$, i.e., the longest string in $\mathcal{P}(p) \cap \mathcal{P}(\acute{p})$, we obtain

$$\rho(p, \acute{p})$$
$$\triangleq \begin{cases} \gamma_d(l(p)), & \text{if } p = \acute{p} \\ \frac{\gamma_d(l(p))\gamma_{d-l(\bar{p})-1}(l(\acute{p})-l(\bar{p})-1)}{\beta_{d-l(\bar{p})-1}}, & \text{if } p_1 \notin \mathcal{P}(\acute{p}) \cup \mathcal{S}_d(\acute{p}) \\ 0, & \text{otherwise.} \end{cases}$$
$$(17)$$

Since $l(\bar{p}) + 1 \leq l(p), l(\acute{p})$ from the definition of the tree, we naturally have $\rho(p, \acute{p}) = \rho(\acute{p}, p)$.

Now turning our attention back to (16) and considering the definition in (17), we notice that the number of occurrences of the product $w_{t,p} \hat{d}_{t,p_k}$ in $\hat{d}_t$ is given by $\rho(p, p_k)$. Hence, the combination weight of the estimate of the node p at time t can be calculated as follows

$$\kappa_{t,p} \triangleq \sum_{\acute{p} \in \mathcal{N}_d} \rho(p, \acute{p}) w_{t,\acute{p}}. \qquad (18)$$

Then, the final estimate of our algorithm becomes

$$\hat{d}_t = \sum_{\nu_j' \in \mathcal{P}(p')} \kappa_{t,\nu_j'} \hat{d}_{t,\nu_j'}. \qquad (19)$$

We emphasize that the estimate of our algorithm given in (19) achieves the exact same result with $\hat{d}_t = \boldsymbol{w}_t^T \hat{\boldsymbol{d}}_t$ with a computational complexity of $O(d2^d)$. Hence, the proof is concluded. $\square$

IV. REGRESSOR SPACE PARTITIONING VIA ADAPTIVE SOFT SEPARATOR FUNCTIONS

In this section, the sequential regressors (as described in Section II.A) for all partitions in the doubly exponential tree

class are combined when soft separation functions are used, i.e., $s_t = (1 + e^{\boldsymbol{x}_t^T \boldsymbol{\theta}_t})^{-1}$, where $\boldsymbol{x}_t \in \mathbb{R}^{m+1}$ is the extended regressor vector and $\boldsymbol{\theta}_t$ is the extended direction vector. By using soft separator functions, we train the corresponding region boundaries, i.e., the structure of the tree.

As in Section III, for β_d different models that are embedded within a depth-d tree, we introduce the algorithm (given in Algorithm 2) achieving asymptotically the same cumulative squared regression error as the optimal combination of the best adaptive models. The algorithm is constructed in the proof of the Theorem 3.

The computational complexity of the algorithm of Theorem 3 is $O(m4^d)$ whereas it achieves the performance of the best combination of $O(2^{2^d})$ different "adaptive" regressors that partitions the m-dimensional regressor space. The computational complexity of the first algorithm was $O(d2^d)$, however, it was unable to learn the region boundaries of the regressor space. In this case since we are using soft separator functions, we need to consider the cross-correlation of every node estimate and node weight, whereas in the previous case there we were only considering the cross-correlation of the estimates of the prefixes of the node $p \in \mathcal{L}_d$ such that $\boldsymbol{x}_t \in \mathcal{R}_p$ and the weights of every node. This change transforms the computational complexity from $O(d2^d)$ to $O(4^d)$. Moreover, for all inner nodes a soft separator function is defined. In order to update the region boundaries of the partitions, we have to update the direction vector $\boldsymbol{\theta}_t$ of size m since $\boldsymbol{x}_t \in \mathbb{R}^m$. Therefore, considering the cross-correlation of the final estimates of every node, we get a computational complexity of $O(m4^d)$.

Theorem 3: Let $\{d_t\}_{t \geq 1}$ and $\{\boldsymbol{x}_t\}_{t \geq 1}$ be arbitrary, bounded, and real-valued sequences. The algorithm $\hat{d}_t$ given in Algorithm 2 when applied these sequences yields

$$\sum_{t=1}^n (d_t - \hat{d}_t)^2 - \min_{\boldsymbol{w} \in \mathbb{R}^{\beta_d}} \sum_{t=1}^n (d_t - \boldsymbol{w}^T \hat{\boldsymbol{d}}_t)^2 \leq O(\ln(n)), \quad (20)$$

for all n, when $e_t^2(\boldsymbol{w})$ is strongly convex $\forall t$, where $\hat{\boldsymbol{d}}_t = [\hat{d}_t^{(1)}, \ldots, \hat{d}_t^{(\beta_d)}]^T$ and $\hat{d}_t^{(k)}$ represents the estimate of d_t at time t for the adaptive model $k = 1, \ldots, \beta_d$.

This theorem implies that our algorithm (given in Algorithm 2), asymptotically achieves the performance of the best linear combination of the $O(2^{2^d})$ different adaptive models that can be represented using a depth-d tree with a computational complexity $O(m4^d)$. We emphasize that while constructing the algorithm, we refrain from any statistical assumptions on the underlying data, and our algorithm works for any sequence of $\{d_t\}_{t \geq 1}$ with an arbitrary length of n. Furthermore, one can use this algorithm to learn the region boundaries and then feed this information to the first algorithm to reduce computational complexity.

A. Outline of the Proof of Theorem 3 and Construction of Algorithm 2

The proof of the upper bound in Theorem 3 follows similar lines to the proof of upper bound in Theorem 1, therefore is omitted. In this proof, we provide the detailed algorithmic description and highlight the computational complexity differences.

According to the same labeling operation we presented in Section II, the final estimate of the kth model at time t can be found as follows

$$\hat{d}^{(k)} = \sum_{p \in \mathcal{M}_k} \hat{\delta}_{t,p}.$$

Similarly, the weight of the kth model is given by

$$w_t^{(k)} = \sum_{p \in \mathcal{M}_k} w_{t,p}.$$

Since we use soft separator functions, we have $\hat{\delta}_{t,p} > 0$ and without introducing any approximations, the final estimate of our algorithm is given as follows

$$\hat{d}_t = \sum_{k=1}^{\beta_d} \left\{ \left(\sum_{p \in \mathcal{M}_k} w_{t,p} \right) \left(\sum_{p \in \mathcal{M}_k} \hat{\delta}_{t,p} \right) \right\}.$$

Here, we observe that for arbitrary two nodes $p, \acute{p} \in \mathcal{N}_d$, the product $w_{t,p} \hat{\delta}_{t,\acute{p}}$ appears $\rho(p, \acute{p})$ times in $\hat{d}_t$, where $\rho(p, \acute{p})$ is the number of models having both p and $\acute{p}$ as its leaf nodes (as we previously defined in (17)). Hence, according to the notation derived in (17) and (18), we obtain the final estimate of our algorithm as follows

$$\hat{d}_t = \sum_{p \in \mathcal{N}_d} \kappa_{t,p} \hat{\delta}_{t,p}. \tag{21}$$

Note that (21) is equal to $\hat{d}_t = \boldsymbol{w}_t^T \hat{\boldsymbol{d}}_t$ with a computational complexity of $O(4^d)$.

Unlike Section III, in which each model has a fixed partitioning of the regressor space, here, we define the regressor models with adaptive partitions. For this, we use a stochastic gradient descent update

$$\boldsymbol{\theta}_{t+1,p} = \boldsymbol{\theta}_{t,p} - \frac{1}{2}\eta_t \nabla e_t^2(\boldsymbol{\theta}_{t,p}), \tag{22}$$

for all nodes $p \in \mathcal{N}_d - \mathcal{L}_d$, where η_t is the learning rate of the region boundaries and $\nabla e_t^2(\boldsymbol{\theta}_{t,p})$ is the derivative of $e_t^2(\boldsymbol{\theta}_{t,p})$ with respect to $\boldsymbol{\theta}_{t,p}$. After some algebra, we obtain

$$\boldsymbol{\theta}_{t+1,p} = \boldsymbol{\theta}_{t,p} + \eta_t e_t \frac{\partial \hat{d}_t}{\partial s_{t,p}} \frac{\partial s_{t,p}}{\partial \boldsymbol{\theta}_{t,p}},$$

$$= \boldsymbol{\theta}_{t,p} + \eta_t e_t \left\{ \sum_{\acute{p} \in \mathcal{N}_d} \kappa_{t,\acute{p}} \frac{\partial \hat{\delta}_{t,\acute{p}}}{\partial s_{t,p}} \right\} \frac{\partial s_{t,p}}{\partial \boldsymbol{\theta}_{t,p}}$$

$$= \boldsymbol{\theta}_{t,p} + \eta_t e_t \left\{ \sum_{q=0}^{1} \sum_{\acute{p} \in \mathcal{S}_d(pq)} (-1)^q \kappa_{t,\acute{p}} \frac{\hat{\delta}_{t,\acute{p}}}{s_{t,p}^q} \right\} \frac{\partial s_{t,p}}{\partial \boldsymbol{\theta}_{t,p}}, \tag{23}$$

where we use the logistic regression classifier as our separator function, i.e., $s_{t,p} = (1 + \exp(\boldsymbol{x}_t^T \boldsymbol{\theta}_{t,p}))^{-1}$. Therefore, we have

$$\frac{\partial s_{t,p}}{\partial \boldsymbol{\theta}_{t,p}} = -\left(1 + \exp(\boldsymbol{x}_t^T \boldsymbol{\theta}_{t,p})\right)^{-2} \exp(\boldsymbol{x}_t^T \boldsymbol{\theta}_{t,p}) \boldsymbol{x}_t$$

$$= -s_{t,p}(1 - s_{t,p})\boldsymbol{x}_t. \tag{24}$$

Algorithm 2: Decision Adaptive Tree (DAT) Regressor

1: **for** $t = 1$ **to** n **do**
2: $\quad \hat{d}_t \Leftarrow 0$
3: $\quad$ **for all** $p \in \mathcal{N}_d - \mathcal{L}_d$ **do**
4: $\quad\quad s_{t,p} \Leftarrow s^+ + (1 - 2s^+)/(1 + e^{\boldsymbol{x}_t^T \boldsymbol{\theta}_{t,p}})$
5: $\quad$ **end for**
6: $\quad$ **for all** $p \in \mathcal{L}_d$ **do**
7: $\quad\quad \hat{d}_{t,p} \Leftarrow \boldsymbol{v}_{t,p}^T \boldsymbol{x}_t$
8: $\quad\quad \alpha_{t,p} \Leftarrow 1$
9: $\quad\quad$ **for** $i = 1$ **to** $l(p)$ **do**
10: $\quad\quad\quad \alpha_{t,p} \Leftarrow \alpha_{t,p} s_{t,\nu_i}^{q_i}$
11: $\quad\quad$ **end for**
12: $\quad\quad \hat{\delta}_{t,p} \Leftarrow \alpha_{t,p} \hat{d}_{t,p}$
13: $\quad\quad \kappa_{t,p} \Leftarrow \gamma_d(l(p)) w_{t,p}$
14: $\quad\quad$ **for all** $\acute{p} \in \mathcal{N}_d - (\mathcal{P}(p) \cup \mathcal{S}_d(p))$ **do**
15: $\quad\quad\quad \bar{p} \Leftarrow \tilde{p} \in \mathcal{P}(p) \cap \mathcal{P}(\acute{p}) : l(\tilde{p}) = |\mathcal{P}(p) \cap \mathcal{P}(\acute{p})| - 1$
16: $\quad\quad\quad \kappa_{t,p} \Leftarrow \kappa_{t,p} + \frac{\gamma_d(l(p))\gamma_{d-l(\bar{p})-1}(l(\acute{p})-l(\bar{p})-1)}{\beta_{d-l(\bar{p})-1}} w_{t,\acute{p}}$
17: $\quad\quad$ **end for**
18: $\quad\quad \hat{d}_t \Leftarrow \hat{d}_t + \kappa_{t,p} \hat{\delta}_{t,p}$
19: $\quad$ **end for**
20: $\quad e_t \Leftarrow d_t - \hat{d}_t$
21: $\quad$ **for all** $p \in \mathcal{L}_d$ **do**
22: $\quad\quad \boldsymbol{v}_{t+1,p} \Leftarrow \boldsymbol{v}_{t,p} + \mu_t e_t \alpha_{t,p} \boldsymbol{x}_t$
23: $\quad\quad w_{t+1,p} \Leftarrow w_{t,p} + \mu_t e_t \hat{\delta}_{t,p}$
24: $\quad$ **end for**
25: $\quad$ **for all** $p \in \mathcal{N}_d - \mathcal{L}_d$ **do**
26: $\quad\quad \sigma_{t,p} \Leftarrow 0$
27: $\quad\quad$ **for all** $\acute{p} \in \mathcal{S}_d(p0)$ **do**
28: $\quad\quad\quad \sigma_{t,p} \Leftarrow \sigma_{t,p} + \kappa_{t,\acute{p}} \frac{\hat{\delta}_{t,\acute{p}}}{s_{t,p}}$
29: $\quad\quad$ **end for**
30: $\quad\quad$ **for all** $\acute{p} \in \mathcal{S}_d(p1)$ **do**
31: $\quad\quad\quad \sigma_{t,p} \Leftarrow \sigma_{t,p} - \kappa_{t,\acute{p}} \frac{\hat{\delta}_{t,\acute{p}}}{1 - s_{t,p}}$
32: $\quad\quad$ **end for**
33: $\quad\quad \boldsymbol{\theta}_{t+1,p} \Leftarrow \boldsymbol{\theta}_{t,p} - \eta_t e_t \sigma_{t,p} s_{t,p}(1 - s_{t,p}) \boldsymbol{x}_t$
34: $\quad$ **end for**
35: **end for**

Note that other separator functions can also be used in a similar way by simply calculating the gradient with respect to the extended direction vector and plugging in (23) and (24).

We emphasize that $\nabla e_t^2(\boldsymbol{\theta}_{t,p})$ includes the product of $s_{t,p}$ and $1 - s_{t,p}$ terms, hence in order not to slow down the learning rate of our algorithm, we may restrict $s^+ \leq |s_t| \leq 1 - s^+$ for some $0 < s^+ < 0.5$. According to this restriction, we define the separator functions as follows

$$s_t = s^+ + \frac{1 - 2s^+}{1 + e^{\boldsymbol{x}_t^T \boldsymbol{\theta}_t}}.$$

According to the update rule in (23), the computational complexity of the introduced algorithm results in $O(m4^d)$. This concludes the outline of the proof and the construction of the algorithm. $\qquad \square$

B. Selection of the Learning Rates

We emphasize that the learning rate μ_t can be set according to the similar studies in the literature [26], [44] or considering

the application requirements. However, for the introduced algorithm to work smoothly, we expect the region boundaries to converge faster than the node weights, therefore, we conventionally choose the learning rate to update the region boundaries as $\eta_t = \mu_t/(s^+(1-s^+))$. Experimentally, we observed that different choices of η_t also yields acceptable performance, however, we note that when updating $\boldsymbol{\theta}_{t,p}$, we have the multiplication term $s_{t,p}(1-s_{t,p})$, which significantly decreases the steps taken at each time t. Therefore, in order to compensate for it, such a selection is reasonable.

On the other hand, for stability purposes, one can consider to put an upper bound on the steps at each time t. When $\boldsymbol{x}_t$ is sufficiently away from the region boundaries $s_{t,p}$, it is either close to s^+ or $1-s^+$. However, when $\boldsymbol{x}_t$ falls right on a region boundary, we have $s_{t,p} = 0.5$, which results in an approximately 25 times greater step than the expected one, when $s^+ = 0.01$. This issue is further exacerbated when $\boldsymbol{x}_t$ falls on the boundary of multiple region crossings, e.g., say $\boldsymbol{x}_t = [0,0]^T$ when we have the four quadrants as the four regions (leaf nodes) of the depth-2 tree. In such a scenario, one can observe a 25^d times greater step than expected, which may significantly perturb the stability of the algorithm. That is why, two alternate solutions can be proposed: *1)* a reasonable threshold (e.g., $10s^+(1-s^+)$)) over the steps can be embedded when s^+ is small (or equivalently, a regularization constant can be embedded), *2)* s^+ can be sufficiently increased according to the depth of the tree. Throughout the experiments, we used the first approach.

C. Selection of the Depth of the Tree

In many real life applications, we do not know how the true data is generated, therefore, the accurate selection of the depth of the decision tree is usually a difficult problem. For instance, if the desired data is generated from a piecewise linear model, then in order for the conventional approaches that use a fixed tree structure (i.e., fixed partitioning of the regressor space) to perfectly estimate the data, they need to perfectly guess the underlying partitions in hindsight. Otherwise, in order to capture the salient characteristics of the desired data, the depth of the tree should be increased to infinity. Hence, the performance of such algorithms significantly varies according to the initial partitioning of the regressor space, which makes it harder to decide how to select the depth of the tree.

On the other hand, the introduced algorithm adapts its region boundaries to minimize the final regression error. Therefore, even if the initial partitioning of the regressor space is not accurate, our algorithm will learn to the locally optimal partitioning of the regressor space for any given depth d. In this sense, one can select the depth of the decision tree by only considering the computational complexity issues of the application.

V. SIMULATIONS

In this section, we illustrate the performance of our algorithms under different scenarios with respect to various methods. We first consider the regression of a signal generated by a piecewise linear model when the underlying partition of the model corresponds to one of the partitions represented by the tree. We then consider the case when the partitioning does not match any partition represented by the tree to demonstrate the region-learning performance of the introduced algorithm.

We also illustrate the performance of our algorithms in underfitting and overfitting (in terms of the depth of the tree) scenarios. We then consider the prediction of two benchmark chaotic processes: the Lorenz attractor and the Henon map. Finally, we illustrate the merits of our algorithm using benchmark data sets (both real and synthetic) such as California housing [48]–[50], elevators [48], kinematics [49], pumadyn [49], and bank [50] (which will be explained in detail in Subsection V.F).

Throughout this section, "DFT" represents the decision fixed tree regressor (i.e., Algorithm 1) and "DAT" represents the decision adaptive tree regressor (i.e., Algorithm 2). Similarly, "CTW" represents the context tree weighting algorithm of [4], "OBR" represents the optimal batch regressor, "VF" represents the truncated Volterra filter [5], "LF" represents the simple linear filter, "B-SAF" and "CR-SAF" represent the Beizer and the Catmul-Rom spline adaptive filter of [6], respectively, "FNF" and "EMFNF" represent the Fourier and even mirror Fourier nonlinear filter of [7], respectively. Finally, "GKR" represents the Gaussian-Kernel regressor and it is constructed using p node regressors, say $\hat{d}_{t,1}, \ldots, \hat{d}_{t,p}$, and a fixed Gaussian mixture weighting (that is selected according to the underlying sequence in hindsight), giving

$$\hat{d}_t = \sum_{i=1}^{p} f(\boldsymbol{x}_t; \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)\hat{d}_{t,i},$$

where $\hat{d}_{t,i} = \boldsymbol{v}_{t,i}^T \boldsymbol{x}_t$ and

$$f(\boldsymbol{x}_t; \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) \triangleq \frac{1}{2\pi\sqrt{|\boldsymbol{\Sigma}_i|}} e^{-\frac{1}{2}(\boldsymbol{x}_t-\boldsymbol{\mu}_i)^T \boldsymbol{\Sigma}_i^{-1}(\boldsymbol{x}_t-\boldsymbol{\mu}_i)},$$

for all $i = 1, \ldots, p$.

For a fair performance comparison, in the corresponding experiments in Subsections V.E and V.F, the desired data and the regressor vectors are normalized between $[-1,1]$ since the satisfactory performance of the several algorithms require the knowledge on the upper bounds (such as the B-SAF and the CR-SAF) and some require these upper bounds to be between $[-1,1]$ (such as the FNF and the EMFNF). Moreover, in the corresponding experiments in Subsections V.B, V.C, and V.D, the desired data and the regressor vectors are normalized between $[-1,1]$ for the VF, the FNF, and the EMFNF due to the aforementioned reason. The regression errors of these algorithms are then scaled back to their original values for a fair comparison.

Considering the illustrated examples in the respective papers [4], [6], [7], the orders of the FNF and the EMFNF are set to 3 for the experiments in Subsections V.B, V.C, and V.D, 2 for the experiments in Subsection V.E, and 1 for the experiments in Subsection V.F. The order of the VF is set to 2 for all experiments, except for the California housing experiment, in which it is set to 3. Similarly, the depth of the tree of the DAT algorithm is set to 2 for all experiments, except for the California housing experiment, in which it is set to 3. The depths of the trees of the DFT and the CTW algorithms are set to 2 for all experiments. For the tree based algorithms, the regressor space is initially partitioned by the direction vectors $\boldsymbol{\theta}_{t,p} = [\theta_{t,p}^{(1)}, \ldots, \theta_{t,p}^{(m)}]^T$ for all nodes $p \in \mathcal{N}_d - \mathcal{L}_d$, where $\theta_{t,p}^{(i)} = -1$ if $i \equiv l(p) \pmod{d}$, e.g., when $d = m = 2$, we have the four quadrants as the four leaf nodes of the tree. Finally, we used cubic B-SAF and CR-SAF algorithms, whose number of knots are set to 21 for all

TABLE I
COMPARISON OF THE COMPUTATIONAL COMPLEXITIES OF THE PROPOSED
ALGORITHMS. IN THE TABLE, m REPRESENTS THE DIMENSIONALITY OF
THE REGRESSOR SPACE, d REPRESENTS THE DEPTH OF THE TREES IN
THE RESPECTIVE ALGORITHMS, AND r REPRESENTS THE ORDER OF THE
CORRESPONDING FILTERS AND ALGORITHMS

Algorithm	Computational Complexity
DFT	$O\left(md2^d\right)$
DAT	$O\left(m4^d\right)$
CTW	$O\left(md\right)$
GKR	$O\left(m2^d\right)$
VF	$O\left(m^r\right)$
B-SAF	$O\left(mr^2\right)$
CR-SAF	$O\left(mr^2\right)$
FNF	$O\left((mr)^r\right)$
EMFNF	$O\left(m^r\right)$

Fig. 3. Regression error performances for the second order piecewise linear model in (25) averaged over 10 trials.

experiments. We emphasize that both these parameters and the learning rates of these algorithms are selected to give equal rate of performance and convergence.

A. Computational Complexities

As can be observed from Table I, among the tree based algorithms that partition the regressor space, the CTW algorithm has the smallest complexity since at each time t, it only associates the regressor vector $\boldsymbol{x}_t$ with $O(d)$ nodes (the leaf node $\boldsymbol{x}_t$ has fallen into and all its prefixes) and their individual weights. The DFT algorithm also considers the same $O(d)$ nodes on the tree, but in addition, it calculates the weight of the each node with respect to the rest of the nodes, i.e., it correlates $O(d)$ nodes with all the $O(2^d)$ nodes. The DAT algorithm, however, estimates the data with respect to the correlation of all the nodes, one another, which results in a computational complexity of $O(4^d)$. In order for the Gaussian-Kernel Regressor (GKR) to achieve a comparable nonlinear modeling power, it should have 2^d mass points, which results in a computational complexity of $O(m2^d)$.

On the other hand, the filters such as the VF, the FNF, and the EMFNF introduce the nonlinearity by directly considering the rth (and up to rth) powers of the entries of the regressor vector. In many practical applications, such methods cannot be applied due to the high dimensionality of the regressor space. Therefore, the algorithms such as the B-SAF and the CR-SAF are introduced to decrease the high computational complexity of such approaches. However, as can be observed from our simulation results, the introduced algorithm significantly outperforms its competitors in various benchmark problems.

The algorithms such as the VF, the FNF, and the EMFNF have more than enough number of basis functions, which result in a significantly slower and parameter dependent convergence performance with respect to the other algorithms. On the other hand, the performances of the algorithms such as the B-SAF, the CR-SAF, and the CTW algorithm are highly dependent on the underlying setting that generates the desired signal. Furthermore, for all these algorithms to yield satisfactory results, prior knowledge on the desired signals and the regressor vectors is needed. The introduced algorithms, on the other hand, do not rely on any prior knowledge, and still outperform their competitors.
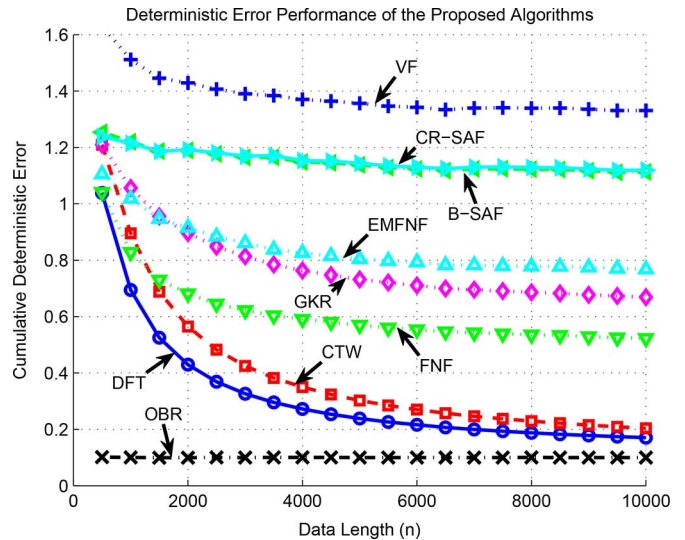
B. Matched Partitions

In this subsection, we consider the case where the desired data is generated by a piecewise linear model that matches with the initial partitioning of the tree based algorithms. Specifically, the desired signal is generated by the following piecewise linear model

$$d_t = \begin{cases} \boldsymbol{w}^T\boldsymbol{x}_t + \pi_t, & \text{if } \boldsymbol{\phi}_0^T\boldsymbol{x}_t \geq 0 \text{ and } \boldsymbol{\phi}_1^T\boldsymbol{x}_t \geq 0 \\ -\boldsymbol{w}^T\boldsymbol{x}_t + \pi_t, & \text{if } \boldsymbol{\phi}_0^T\boldsymbol{x}_t \geq 0 \text{ and } \boldsymbol{\phi}_1^T\boldsymbol{x}_t < 0 \\ -\boldsymbol{w}^T\boldsymbol{x}_t + \pi_t, & \text{if } \boldsymbol{\phi}_0^T\boldsymbol{x}_t < 0 \text{ and } \boldsymbol{\phi}_1^T\boldsymbol{x}_t \geq 0 \\ \boldsymbol{w}^T\boldsymbol{x}_t + \pi_t, & \text{if } \boldsymbol{\phi}_0^T\boldsymbol{x}_t < 0 \text{ and } \boldsymbol{\phi}_1^T\boldsymbol{x}_t < 0, \end{cases} \quad (25)$$

where $\boldsymbol{w} = [1, 1]^T$, $\boldsymbol{\phi}_0 = [1, 0]^T$, $\boldsymbol{\phi}_1 = [0, 1]^T$, $\boldsymbol{x}_t = [x_{1,t}, x_{2,t}]^T$, π_t is a sample function from a zero mean white Gaussian process with variance 0.1, $x_{1,t}$ and $x_{2,t}$ are sample functions of a jointly Gaussian process of mean $[0, 0]^T$ and variance $\boldsymbol{I}_2$. The desired data at time t is denoted as d_t whereas the extended regressor vector is $\boldsymbol{x}_t = [x_{1,t}, x_{2,t}, 1]^T$, i.e., x_1 represents the first dimension and x_2 the second dimension.

For this scenario, the learning rates are set to 0.005 for the DFT algorithm, the FNF, and the CTW algorithm, 0.025 for the B-SAF and the CR-SAF, 0.05 for the VF and the EMFNF, 1 for the GKR. Moreover, for the GKR, $\boldsymbol{\mu}_i = 1.2 \times [(-1)^{\lfloor(i-1)/2\rfloor}, (-1)^i]^T$ and $\boldsymbol{\Sigma}_i = 1.2 \times \boldsymbol{I}_2$ for $i = 1, \ldots, 4$, are set to exactly match the underlying partitioning that generates the desired data.

In Fig. 3, we demonstrate the time accumulated regression error of the proposed algorithms averaged over 10 trials. Since the desired data is generated by a highly nonlinear piecewise model, the algorithms such as the GKR, the FNF, the EMFNF, the B-SAF, and the CR-SAF cannot capture the salient characteristics of the data. These algorithms yield satisfactory results only if the desired data is generated by a smooth nonlinear function of the regressor vector. In this scenario, however, we have high nonlinearity and discontinuity, which makes the algorithms such as the DFT and the CTW appealing.

Comparing the DFT and the CTW algorithms, we can observe that even though the partitioning of the tree perfectly
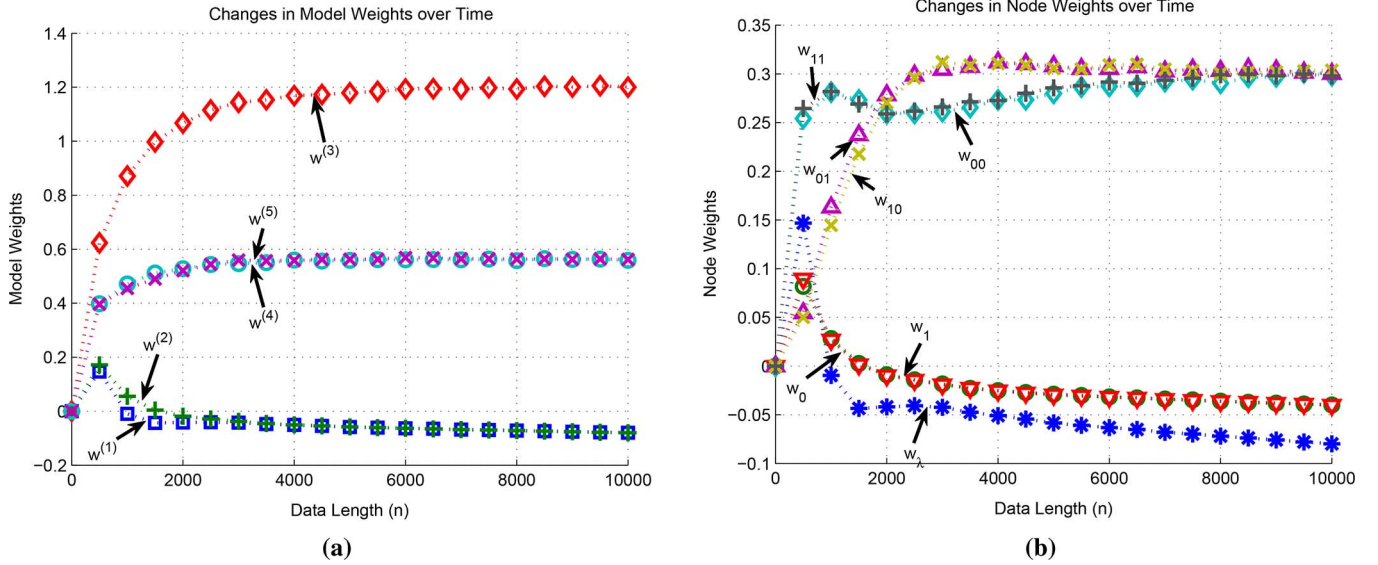
Fig. 4. Progress of (a) the model weights and (b) the node weights averaged over 10 trials for the DFT algorithm. Note that the model weights do not sum up to 1.

matches with the underlying partition in (25), the learning performance of the DFT algorithm significantly outperforms the CTW algorithm especially for short data records. As commented in the text, this is expected since the context-tree weighting method enforces the sum of the model weights to be 1, however, the introduced algorithms have no such restrictions. As seen in Fig. 4(a), the model weights sum up to 2.1604 instead of 1. Moreover, in the CTW algorithm all model weights are "forced" to be nonnegative whereas in our algorithm model weights can also be negative as seen in Fig. 4(a). In Fig. 4(b), the individual node weights are presented. We observe that the nodes (i.e., regions) that directly match with the underlying partition that generates the desired data have higher weights whereas the weights of the other nodes decrease. We also point out that although the tree based algorithms [4], [35], [36] need a priori information, such as an upper bound on the desired data, for a successful operation, whereas the introduced algorithm has no such requirements.

C. Mismatched Partitions

In this subsection, we consider the case where the desired data is generated by a piecewise linear model that mismatches with the initial partitioning of the tree based algorithms. Specifically, the desired signal is generated by the following piecewise linear model

$$d_t = \begin{cases} \boldsymbol{w}^T \boldsymbol{x}_t + \pi_t, & \text{if } \boldsymbol{\phi}_0^T \boldsymbol{x}_t \geq 0.5 \text{ and } \boldsymbol{\phi}_1^T \boldsymbol{x}_t \geq 1 \\ -\boldsymbol{w}^T \boldsymbol{x}_t + \pi_t, & \text{if } \boldsymbol{\phi}_0^T \boldsymbol{x}_t \geq 0.5 \text{ and } \boldsymbol{\phi}_1^T \boldsymbol{x}_t < 1 \\ -\boldsymbol{w}^T \boldsymbol{x}_t + \pi_t, & \text{if } \boldsymbol{\phi}_0^T \boldsymbol{x}_t < 0.5 \text{ and } \boldsymbol{\phi}_2^T \boldsymbol{x}_t \geq -1 \\ \boldsymbol{w}^T \boldsymbol{x}_t + \pi_t, & \text{if } \boldsymbol{\phi}_0^T \boldsymbol{x}_t < 0.5 \text{ and } \boldsymbol{\phi}_2^T \boldsymbol{x}_t < -1, \end{cases} \quad (26)$$

where $\boldsymbol{w} = [1, 1]^T$, $\boldsymbol{\phi}_0 = [4, -1]^T$, $\boldsymbol{\phi}_1 = [1, 1]^T$, $\boldsymbol{\phi}_2 = [1, 2]^T$, $\boldsymbol{x}_t = [x_{1,t}, x_{2,t}]^T$, π_t is a sample function from a zero mean white Gaussian process with variance 0.1, $x_{1,t}$ and $x_{2,t}$ are sample functions of a jointly Gaussian process of mean $[0, 0]^T$ and variance $\boldsymbol{I}_2$. The learning rates are set to 0.005 for the DFT, the DAT, and the CTW algorithms, 0.1 for the FNF, 0.025 for the B-SAF and the CR-SAF, 0.05 for the EMFNF and the VF. Moreover, in order to match the underlying partition,
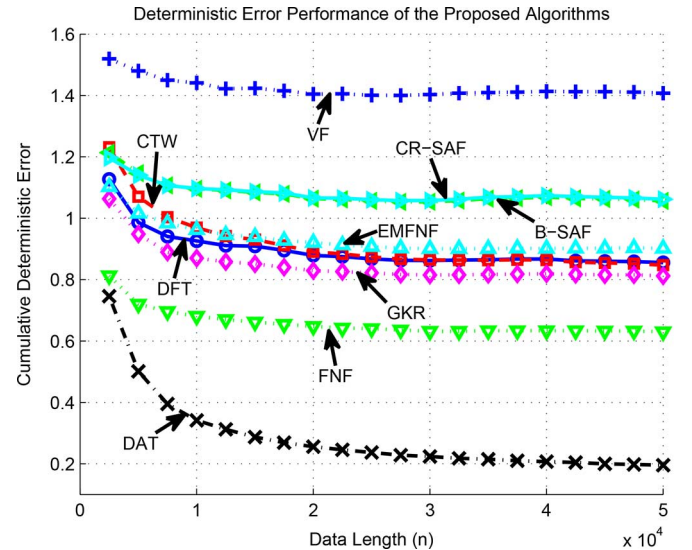

Fig. 5. Regression error performances for the second order piecewise linear model in (26).

the mass points of the GKR are set to $\boldsymbol{\mu}_1 = [1.4565, 1.0203]^T$, $\boldsymbol{\mu}_2 = [0.6203, -0.4565]^T$, $\boldsymbol{\mu}_3 = [-0.5013, 0.5903]^T$, and $\boldsymbol{\mu}_4 = [-1.0903, -1.0013]^T$ with the same covariance matrix in the previous example.

Fig. 5 shows the normalized time accumulated regression error of the proposed algorithms. We emphasize that the DAT algorithm achieves a better error performance compared to its competitors. Comparing Figs. 3 and 5, one can observe the degradation in the performances of the DFT and the CTW algorithms. This shows the importance of the initial partitioning of the regressor space for tree based algorithms to yield a satisfactory performance. Comparing the same figures, one can also observe that the rest of the algorithms performs almost similar to the previous scenario.

The DFT and the CTW algorithms converge to the best batch regressor having the predetermined leaf nodes (i.e., the best regressor having the four quadrants of two dimensional space
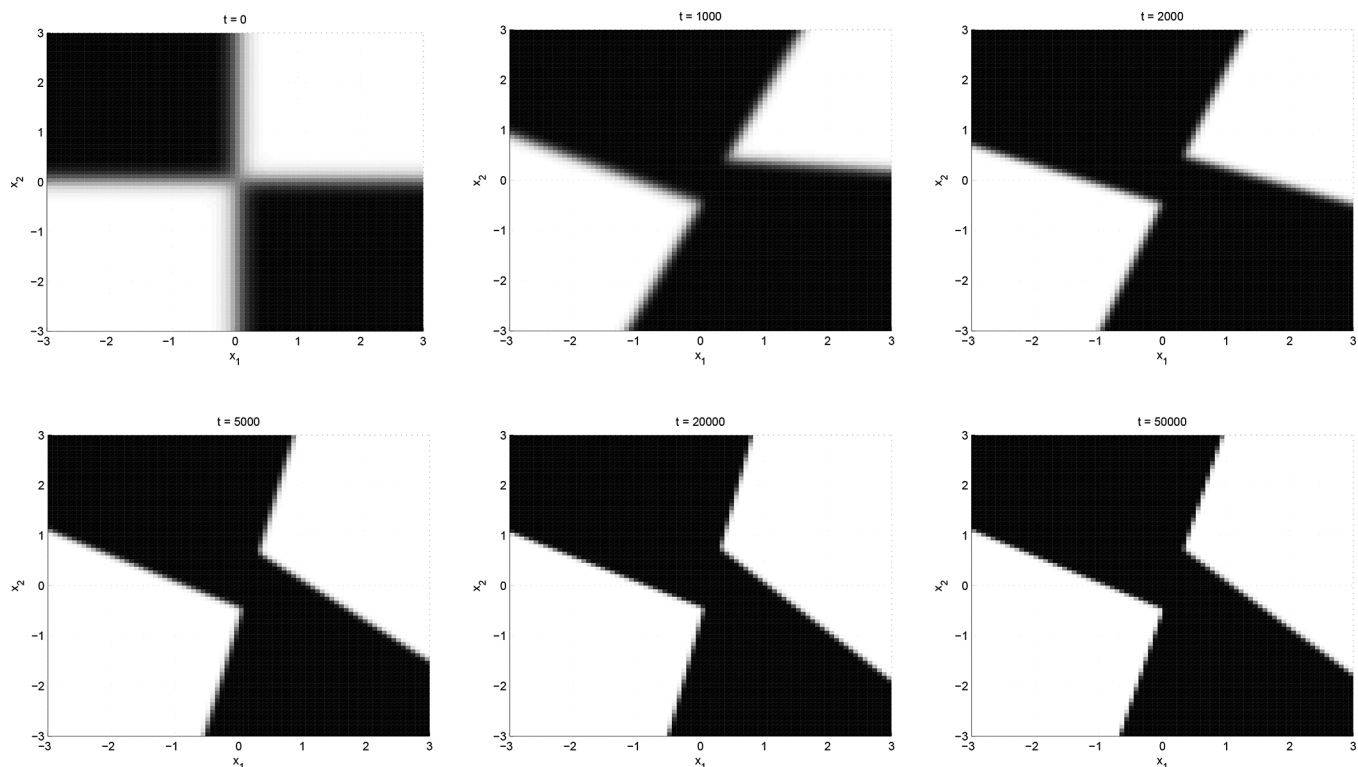
Fig. 6.   Changes in the boundaries of the leaf nodes of the depth-2 tree of the DAT algorithm for $t = 0, 1000, 2000, 5000, 20000, 50000$. The separator functions adaptively learn the boundaries of the piecewise linear model in (26).
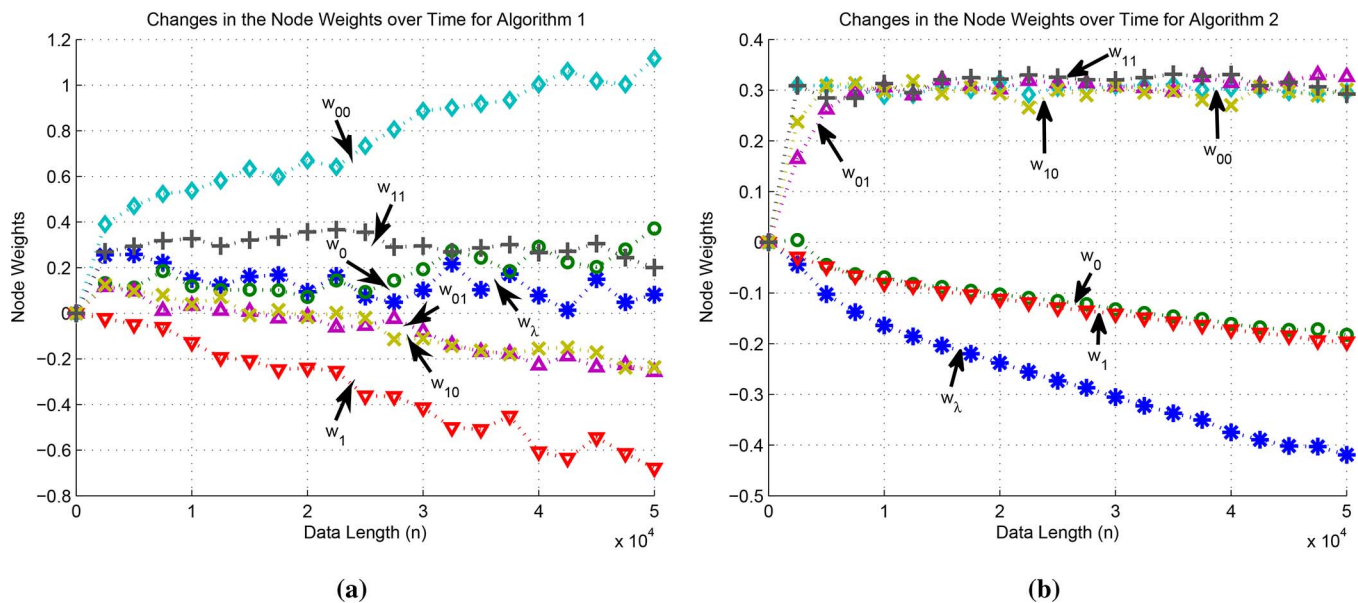

Fig. 7.   Progress of the node weights for the piecewise linear model in (26) for (a) the DFT algorithm and (b) the DAT algorithm.

as its leaf nodes). However that regressor is sub-optimal since the underlying data is generated using another constellation, hence their time accumulated regression error is always lower bounded by $O(1)$ compared to the global optimal regressor. The DAT algorithm, on the other hand, adapts its region boundaries and captures the underlying unevenly rotated and shifted regressor space partitioning, perfectly. Fig. 6 shows how our algorithm updates its separator functions and illustrates the nonlinear modeling power of the introduced DAT algorithm.

We also present the node weights for the DFT and the DAT algorithms in Fig. 7(a) and (b), respectively. In Fig. 7(a), we can observe that the DFT algorithm cannot estimate the underlying data accurately, hence its node weights show unstable behavior. On the other hand, as can be observed from Fig. 7(b), the DAT algorithm learns the optimal node weights as the region boundaries are learned. In this manner, the DAT algorithm achieves a significantly superior performance with respect to its competitors.
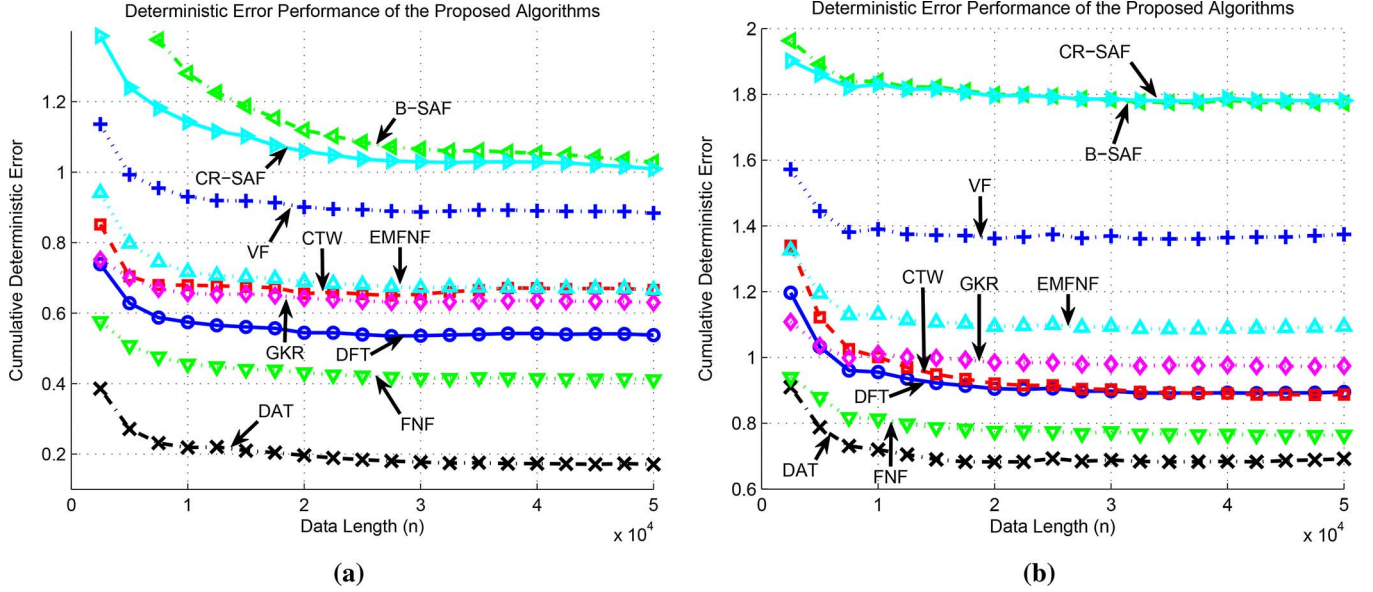
Fig. 8. Regression error performances for (a) the first order piecewise linear model in (27) (b) the third order piecewise linear model in (28).

D. Mismatched Partitions With Overfitting & Underfitting

In this subsection, we consider two cases (and perform two experiments), where the desired data is generated by a piecewise linear model that mismatches with the initial partitioning of the tree based algorithms, where the depth of the tree overfits or underfits the underlying piecewise model. In the first set of experiments, we consider that the data is generated from a first order piecewise linear model, for which using a depth-1 tree is sufficient to capture the salient characteristics of the data. In the second set of experiments, we consider that the data is generated from a third order piecewise linear model, for which it is necessary to use a depth-3 tree to perfectly estimate the data.

The first order piecewise linear model is defined as

$$d_t = \begin{cases} \boldsymbol{w}^T \boldsymbol{x}_t + \pi_t, & \text{if } \boldsymbol{\phi}_0^T \boldsymbol{x}_t \geq 0.5 \\ -\boldsymbol{w}^T \boldsymbol{x}_t + \pi_t, & \text{if } \boldsymbol{\phi}_0^T \boldsymbol{x}_t < 0.5 \end{cases}, \qquad (27)$$

and the third order piecewise linear model is defined as

$$d_t = \begin{cases} \boldsymbol{w}^T \boldsymbol{x}_t + \pi_t, & \text{if } \boldsymbol{\phi}_0^T \boldsymbol{x}_t \geq 0.5, \boldsymbol{\phi}_1^T \boldsymbol{x}_t \geq 1, \boldsymbol{\phi}_3^T \boldsymbol{x}_t \geq 0.5 \\ -\boldsymbol{w}^T \boldsymbol{x}_t + \pi_t, & \text{if } \boldsymbol{\phi}_0^T \boldsymbol{x}_t \geq 0.5, \boldsymbol{\phi}_1^T \boldsymbol{x}_t \geq 1, \boldsymbol{\phi}_3^T \boldsymbol{x}_t < 0.5 \\ \boldsymbol{w}^T \boldsymbol{x}_t + \pi_t, & \text{if } \boldsymbol{\phi}_0^T \boldsymbol{x}_t \geq 0.5, \boldsymbol{\phi}_1^T \boldsymbol{x}_t < 1, \boldsymbol{\phi}_4^T \boldsymbol{x}_t \geq 0.5 \\ -\boldsymbol{w}^T \boldsymbol{x}_t + \pi_t, & \text{if } \boldsymbol{\phi}_0^T \boldsymbol{x}_t \geq 0.5, \boldsymbol{\phi}_1^T \boldsymbol{x}_t < 1, \boldsymbol{\phi}_4^T \boldsymbol{x}_t < 0.5 \\ \boldsymbol{w}^T \boldsymbol{x}_t + \pi_t, & \text{if } \boldsymbol{\phi}_0^T \boldsymbol{x}_t < 0.5, \boldsymbol{\phi}_2^T \boldsymbol{x}_t < 0.5, \boldsymbol{\phi}_5^T \boldsymbol{x}_t < 0.5 \\ -\boldsymbol{w}^T \boldsymbol{x}_t + \pi_t, & \text{if } \boldsymbol{\phi}_0^T \boldsymbol{x}_t < 0.5, \boldsymbol{\phi}_2^T \boldsymbol{x}_t < 0.5, \boldsymbol{\phi}_5^T \boldsymbol{x}_t \geq 0.5 \\ \boldsymbol{w}^T \boldsymbol{x}_t + \pi_t, & \text{if } \boldsymbol{\phi}_0^T \boldsymbol{x}_t < 0.5, \boldsymbol{\phi}_2^T \boldsymbol{x}_t \geq 0.5, \boldsymbol{\phi}_6^T \boldsymbol{x}_t < 0.5 \\ -\boldsymbol{w}^T \boldsymbol{x}_t + \pi_t, & \text{if } \boldsymbol{\phi}_0^T \boldsymbol{x}_t < 0.5, \boldsymbol{\phi}_2^T \boldsymbol{x}_t \geq 0.5, \boldsymbol{\phi}_6^T \boldsymbol{x}_t \geq 0.5 \end{cases} \qquad (28)$$

where $\boldsymbol{w} = [1, 1]^T$, $\boldsymbol{\phi}_0 = [4, -1]^T$, $\boldsymbol{\phi}_1 = [1, 1]^T$, $\boldsymbol{\phi}_2 = [-1, -2]^T$, $\boldsymbol{\phi}_3 = [0, 1]^T$, $\boldsymbol{\phi}_4 = [1, 0]^T$, $\boldsymbol{\phi}_5 = [-1, 0]^T$, $\boldsymbol{\phi}_6 = [0, -1]^T$, $\boldsymbol{x}_t = [x_{1,t}, x_{2,t}]^T$, π_t is a sample function from a zero mean white Gaussian process with variance 0.1, $x_{1,t}$ and $x_{2,t}$ are sample functions of a jointly Gaussian process of mean $[0, 0]^T$ and variance $\boldsymbol{I}_2$. The learning rates are set to 0.005 for the DFT, the DAT, and the CTW algorithms, 0.05 for the EMFNF, 0.01 for the B-SAF, the CR-SAF, and the FNF, 0.5 for the VF, and 1 for the GKR, where the parameters of the GKR are set to the same values in the previous example.

We present the normalized regression errors of the proposed algorithms in Fig. 8. Fig. 8(a) shows the performances of the algorithms in the overfitting scenario, where the desired data is generated by the first order piecewise linear model in (27). Similarly, Fig. 8(b) shows the performances of the algorithms in the underfitting scenario, where the desired data is generated by the third order piecewise linear model in (28). From the figures, it is observed that the DAT algorithm outperforms its competitors by learning the optimal partitioning for the given depth, which illustrates the power of the introduced algorithm under possible mismatches in terms of d.

E. Chaotic Signals

In this subsection, we illustrate the performance of our algorithm when estimating a chaotic data generated by *i*) the Henon map and *ii*) the Lorenz attractor [51].

First, we consider a zero-mean sequence generated by the Henon map, a chaotic process given by

$$d_t = 1 - \zeta d_{t-1}^2 + \eta d_{t-2}, \qquad (29)$$

and known to exhibit chaotic behavior for the values of $\zeta = 1.4$ and $\eta = 0.3$. The desired data at time t is denoted as d_t whereas the extended regressor vector is $\boldsymbol{x}_t = [d_{t-1}, d_{t-2}, 1]^T$, i.e., we consider a prediction framework. The learning rates are set to 0.025 for the B-SAF and the CR-SAF algorithms, whereas it is 0.05 for the rest.

Fig. 9 shows the normalized regression error performance of the proposed algorithms. One can observe that the algorithms whose basis functions do not include the necessary quadratic terms and the algorithms that rely on a fixed regressor space partitioning yield unsatisfactory performance. On the other hand, we emphasize that the VF can capture the salient characteristics of this chaotic process since its order is set to 2. Similarly, the FNF can also learn the desired data since its basis functions can well approximate the chaotic process. The DAT algorithm, however, uses a piecewise linear modeling and still achieves
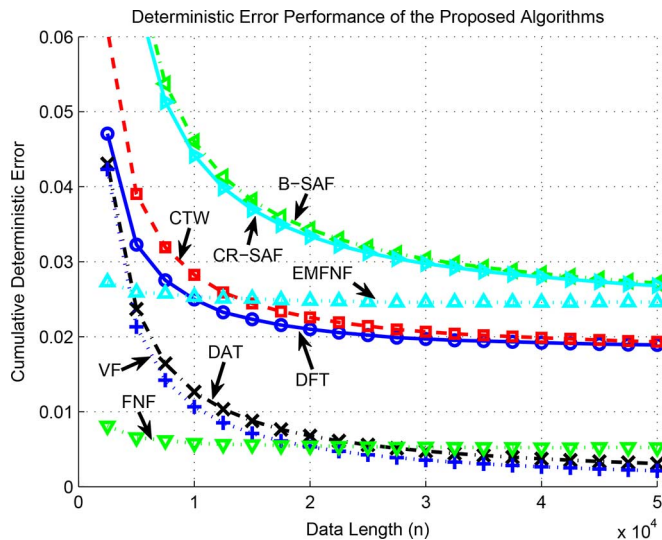
Fig. 9. Regression error performances of the proposed algorithms for the chaotic process presented in (29).
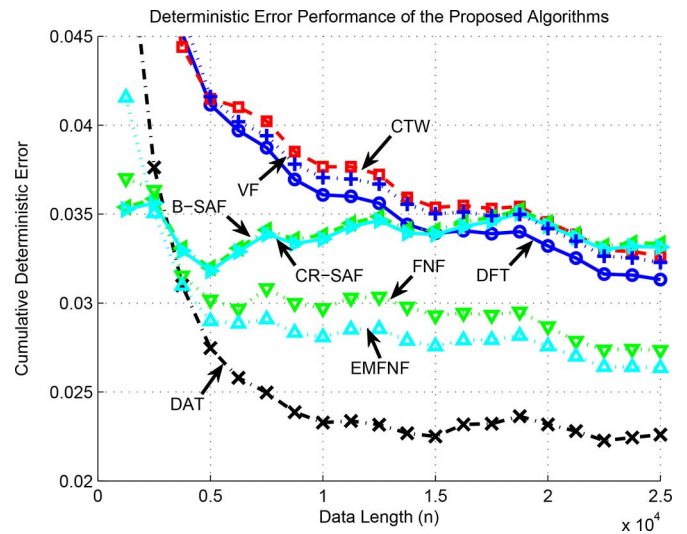

Fig. 10. Regression error performances for the chaotic signal generated from the Lorenz attractor in (30), (31), and (32) with parameters $dt = 0.01$, $\rho = 28$, $\sigma = 10$, and $\beta = 8/3$.

the asymptotically same performance as the VF, while outperforming the FNF algorithm.

Second, we consider the chaotic signal set generated using the Lorenz attractor [51] that is defined by the following three discrete time equations:

$$x_t = x_{t-1} + (\sigma(y - x))dt \tag{30}$$

$$y_t = y_{t-1} + (x_{t-1}(\rho - z_{t-1}) - y_{t-1})dt \tag{31}$$

$$z_t = z_{t-1} + (x_{t-1}y_{t-1} - \beta z_{t-1})dt, \tag{32}$$

where we set $dt = 0.01$, $\rho = 28$, $\sigma = 10$, and $\beta = 8/3$ to generate the well-known chaotic solution of the Lorenz attractor. In the experiment, x_t is selected as the desired data and the two dimensional region represented by y_t, z_t is set as the regressor space, that is, we try to estimate x_t with respect to y_t and z_t. The learning rates are set to 0.01 for all the algorithms.

Fig. 10 illustrates the nonlinear modeling power of the DAT algorithm even when estimating a highly nonlinear chaotic signal set. As can be observed from Fig. 10, the DAT algorithm significantly outperforms its competitors and achieves a superior error performance since it tunes its region boundaries to the optimal partitioning of the regressor space, whereas the performances of the other algorithms directly rely on the initial selection of the basis functions and/or tree structures and partitioning.

F. Benchmark Real and Synthetic Data

In this subsection, we first consider the regression of a benchmark real-life problem that can be found in many data set repositories such as [48]–[50]: California housing—estimation of the median house prices in the California area using California housing database. In this experiment, the learning rates are set to 0.01 for all the algorithms. Fig. 11 provides the normalized regression errors of the proposed algorithms, where it is observed that the DAT algorithm outperforms its competitors and can achieve a much higher nonlinear modeling power with respect to the rest of the algorithms.
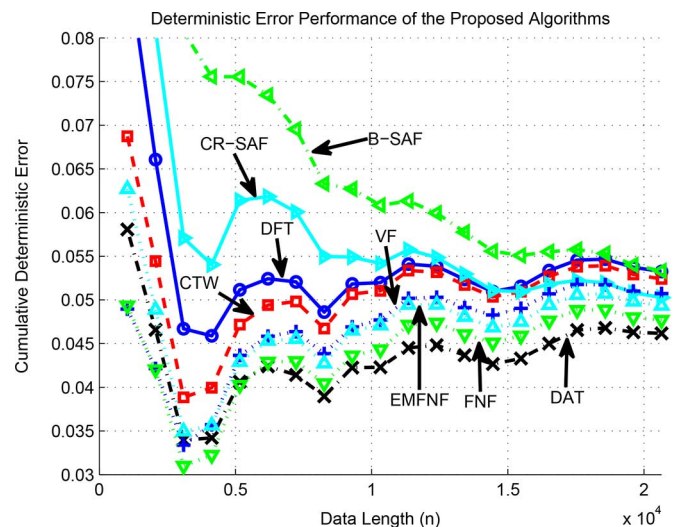

Fig. 11. Regression error performances for the real data set: California housing—estimation of the median house prices in the California area using California housing database [48]–[50].

Aside from the California housing data set, we also consider the regression of several benchmark real life and synthetic data from the corresponding data set repositories:

- Kinematics [48] ($m = 8$)—a realistic simulation of the forward dynamics of an 8 link all-revolute robot arm. The task in all data sets is to predict the distance of the end-effector from a target. (among the existent variants of this data set, we used the variant with $m = 8$, which is known to be highly nonlinear and medium noisy).
- Elevators [49] ($m = 18$)—obtained from the task of controlling a F16 aircraft. In this case the goal variable is related to an action taken on the elevators of the aircraft.
- Pumadyn [49] ($m = 32$)—a realistic simulation of the dynamics of Unimation Puma 560 robot arm. The task in the data set is to predict the angular acceleration of one of the robot arm's links.

TABLE II
TIME ACCUMULATED NORMALIZED ERRORS OF THE PROPOSED ALGORITHMS. EACH DIMENSION OF THE DATA SETS IS NORMALIZED BETWEEN $[-1, 1]$

Algorithms / Data Sets	DAT	LF	VF	FNF	EMFNF	B-SAF	CR-SAF
Kinematics	0.0639	0.0835	0.0746	0.0956	0.0808	0.1108	0.1029
Elevators	0.0091	0.0193	0.0194	0.0112	0.0149	0.0222	0.0225
Pumadyn	0.0780	0.0817	0.0910	0.0904	0.0781	0.0947	0.0936
Bank	0.0511	0.0739	0.0804	0.0544	0.0533	0.0764	0.0891

- Bank [50] ($m = 32$)—generated from a simplistic simulator, which simulates the queues in a series of banks. Tasks are based on predicting the fraction of bank customers who leave the bank because of full queues (among the existent variants of this data set, we used the variant with $m = 32$).

The learning rates of the LF, the VF, the FNF, the EMFNF, and the DAT algorithm are set to μ, whereas it is set to 10μ for the B-SAF and the CR-SAF algorithms, where $\mu = 0.01$ for the kinematics, the elevators, and the bank data sets and $\mu = 0.005$ for the pumadyn data set. In Table II, it is observed that the performance of the DAT algorithm is superior to its competitors since it achieves a much higher nonlinear modeling power with respect to the rest of the algorithms. Furthermore, the DAT algorithm achieves this superior performance with a computational complexity that is only linear in the regressor space dimensionality. Hence, the introduced algorithm can be used in real life big data problems.

VI. CONCLUDING REMARKS

We study nonlinear regression of deterministic signals using trees, where the space of regressors is partitioned using a nested tree structure where separate regressors are assigned to each region. In this framework, we introduce tree based regressors that both adapt their regressors in each region as well as their tree structure to best match to the underlying data while asymptotically achieving the performance of the best linear combination of a doubly exponential number of piecewise regressors represented on a tree. As shown in the text, we achieve this performance with a computational complexity only linear in the number of nodes of the tree. Furthermore, the introduced algorithms do not require a priori information on the data such as upper bounds or the length of the signal. Since these algorithms directly minimize the final regression error and avoid using any artificial weighting coefficients, they readily outperform different tree based regressors in our examples. The introduced algorithms are generic such that one can easily use different regressor or separation functions or incorporate partitioning methods such as the RP trees in their framework as explained in the paper.

REFERENCES

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

[2] O. J. J. Michel, A. O. Hero, and A.-E. Badel, "Tree-structured nonlinear signal modeling and prediction," *IEEE Trans. Signal Process.*, vol. 47, no. 11, pp. 3027–3041, 1999.

[3] R. J. Drost and A. C. Singer, "Constrained complexity generalized context-tree algorithms," in *Proc. IEEE/SP 14th Workshop Statist. Signal Process.*, 2007, pp. 131–135.

[4] S. S. Kozat, A. C. Singer, and G. C. Zeitler, "Universal piecewise linear prediction via context trees," *IEEE Trans. Signal Process.*, vol. 55, no. 7, pp. 3730–3745, 2007.

[5] M. Schetzen, *The Volterra and Wiener Theories of Nonlinear Systems*. New York, NY, USA: Wiley, 1980.

[6] M. Scarpiniti, D. Comminiello, R. Parisi, and A. Uncini, "Nonlinear spline adaptive filtering," *Signal Process.*, vol. 93, no. 4, pp. 772–783, 2013.

[7] A. Carini and G. L. Sicuranza, "Fourier nonlinear filters," *Signal Process.*, vol. 94, no. 0, pp. 183–194, 2014.

[8] V. Kekatos and G. Giannakis, "Sparse Volterra and polynomial regression models: Recoverability and estimation," *IEEE Trans. Signal Process.*, vol. 59, no. 12, pp. 5907–5920, 2011.

[9] L. Montefusco, D. Lazzaro, and S. Papi, "Fast sparse image reconstruction using adaptive nonlinear filtering," *IEEE Trans. Image Process.*, vol. 20, no. 2, pp. 534–544, 2011.

[10] Q. Zhu, Z. Zhang, Z. Song, Y. Xie, and L. Wang, "A novel nonlinear regression approach for efficient and accurate image matting," *IEEE Signal Process. Lett.*, vol. 20, no. 11, pp. 1078–1081, 2013.

[11] R. Mittelman and E. Miller, "Nonlinear filtering using a new proposal distribution and the improved fast Gauss transform with tighter performance bounds," *IEEE Trans. Signal Process.*, vol. 56, no. 12, pp. 5746–5757, 2008.

[12] L. Montefusco, D. Lazzaro, and S. Papi, "Nonlinear filtering for sparse signal recovery from incomplete measurements," *IEEE Trans. Signal Process.*, vol. 57, no. 7, pp. 2494–2502, 2009.

[13] W. Zhang, B.-S. Chen, and C.-S. Tseng, "Robust H_∞ filtering for nonlinear stochastic systems," *IEEE Trans. Signal Process.*, vol. 53, no. 2, pp. 589–598, 2005.

[14] H. Zhao and J. Zhang, "A novel adaptive nonlinear filter-based pipelined feedforward second-order Volterra architecture," *IEEE Trans. Signal Process.*, vol. 57, no. 1, pp. 237–246, 2009.

[15] L. Ma, Z. Wang, J. Hu, Y. Bo, and Z. Guo, "Robust variance-constrained filtering for a class of nonlinear stochastic systems with missing measurements," *Signal Process.*, vol. 90, no. 6, pp. 2060–2071, 2010.

[16] W. Yang, M. Liu, and P. Shi, "H_∞ filtering for nonlinear stochastic systems with sensor saturation, quantization and random packet losses," *Signal Process.*, vol. 92, no. 6, pp. 1387–1396, 2012.

[17] W. Li and Y. Jia, "H-infinity filtering for a class of nonlinear discrete-time systems based on unscented transform," *Signal Process.*, vol. 90, no. 12, pp. 3301–3307, 2010.

[18] S. Wen, Z. Zeng, and T. Huang, "Reliable H_∞ filtering for neutral systems with mixed delays and multiplicative noises," *Signal Process.*, vol. 94, no. 0, pp. 23–32, 2014.

[19] M. F. Huber, "Chebyshev polynomial Kalman filter," *Digit. Signal Process.*, vol. 23, no. 5, pp. 1620–1629, 2013.

[20] D. P. Helmbold and R. E. Schapire, "Predicting nearly as well as the best pruning of a decision tree," *Mach. Learn.*, vol. 27, no. 1, pp. 51–68, 1997.

[21] O.-A. Maillard and R. Munos, "Linear regression with random projections," *J. Mach. Learn. Res.*, vol. 13, pp. 2735–2772, 2012.

[22] R. Rosipal and L. J. Trejo, "Kernel partial least squares regression in reproducing Kernel Hilbert Space," *J. Mach. Learn. Res.*, vol. 2, pp. 97–123, 2001.

[23] O.-A. Maillard and R. Munos, "Some greedy learning algorithms for sparse regression and classification with Mercer kernels," *J. Mach. Learn. Res.*, vol. 13, pp. 2735–2772, 2012.

[24] A. C. Singer and M. Feder, "Universal linear prediction by model order weighting," *IEEE Trans. Signal Process.*, vol. 47, no. 10, pp. 2685–2699, 1999.

[25] T. Moon and T. Weissman, "Universal FIR MMSE filtering," *IEEE Trans. Signal Process.*, vol. 57, no. 3, pp. 1068–1083, 2009.

[26] A. H. Sayed, *Fundamentals of Adaptive Filtering*. Hoboken, NJ, USA: Wiley, 2003.

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

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

[29] J. Arenas-Garcia, A. R. Figueiras-Vidal, and A. H. Sayed, "Mean-square performance of a convex combination of two adaptive filters," *IEEE Trans. Signal Process.*, vol. 54, no. 3, pp. 1078–1090, 2006.

[30] S. Dasgupta and Y. Freund, "Random projection trees for vector quantization," *IEEE Trans. Inf. Theory*, vol. 55, no. 7, pp. 3229–3242, 2009.

[31] Y. Yilmaz and S. S. Kozat, "Competitive randomized nonlinear prediction under additive noise," *IEEE Signal Process. Lett.*, vol. 17, no. 4, pp. 335–339, 2010.

[32] G. David and A. Averbuch, "Hierarchical data organization, clustering and denoising via localized diffusion folders," *Appl. Comput. Harmon. Anal.*, vol. 33, no. 1, pp. 1–23, 2012.

[33] N. B. Lee, A. B. , and L. Wasserman, "Treelets—An adaptive multiscale basis for sparse unordered data," *Ann. Appl. Statist.*, vol. 2, no. 2, pp. 435–471, 2008.

[34] J. L. Bentley, "Multidimensional binary search trees in database applications," *IEEE Trans. Softw. Eng.*, vol. SE-5, no. 4, pp. 333–340, 1979.

[35] E. Takimoto, A. Maruoka, and V. Vovk, "Predicting nearly as well as the best pruning of a decision tree through dyanamic programming scheme," *Theoretic. Comput. Sci.*, vol. 261, pp. 179–209, 2001.

[36] E. Takimoto and M. K. Warmuth, "Predicting nearly as well as the best pruning of a planar decision graph," *Theoretic. Comput. Sci.*, vol. 288, pp. 217–235, 2002.

[37] A. V. Aho and N. J. A. Sloane, "Some doubly exponential sequences," *Fibonacci Quart.*, vol. 11, pp. 429–437, 1970.

[38] F. M. J. Willems, Y. M. Shtarkov, and T. J. Tjalkens, "The context-tree weighting method: Basic properties," *IEEE Trans. Inf. Theory*, vol. 41, no. 3, pp. 653–664, 1995.

[39] A. C. Singer, S. S. Kozat, and M. Feder, "Universal linear least squares prediction: Upper and lower bounds," *IEEE Trans. Inf. Theory*, vol. 48, no. 8, pp. 2354–2362, 2002.

[40] T. Linder and G. Lagosi, "A zero-delay sequential scheme for lossy coding of individual sequences," *IEEE Trans. Inf. Theory*, vol. 47, no. 6, pp. 2533–2538, 2001.

[41] A. Gyorgy, T. Linder, and G. Lugosi, "Efficient adaptive algorithms and minimax bounds for zero-delay lossy source coding," *IEEE Trans. Signal Process.*, vol. 52, no. 8, pp. 2337–2347, 2004.

[42] J. Arenas-Garcia, V. Gomez-Verdejo, and A. R. Figueiras-Vidal, "New algorithms for improved adaptive convex combination of LMS transversal filters," *IEEE Trans. Instrum. Meas.*, vol. 54, no. 6, pp. 2239–2249, 2005.

[43] D. W. Hosmer, S. Lemeshow, and R. X. Sturdivant, *Applied Logistic Regression*. Hoboken, NJ, USA: Wiley, 2013.

[44] E. Hazan, A. Agarwal, and S. Kale, "Logarithmic regret algorithms for online convex optimization," *Mach. Learn.*, vol. 69, no. 2–3, pp. 169–192, 2007.

[45] E. Eweda, "Comparison of RLS, LMS, and sign algorithms for tracking randomly time-varying channels," *IEEE Trans. Signal Process.*, vol. 42, no. 11, pp. 2937–2944, 1994.

[46] J. Arenas-Garcia, M. Martinez-Ramon, V. Gomez-Verdejo, and A. R. Figueiras-Vidal, "Multiple plant identifier via adaptive LMS convex combination," in *Proc. IEEE Int. Symp. Intell. Signal Process.*, 2003, pp. 137–142.

[47] 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 Trans. Signal Process.*, vol. 58, no. 8, pp. 4050–4063, Aug. 2010.

[48] C. E. Rasmussen, R. M. Neal, G. Hinton, D. Camp, M. Revow, Z. Ghahramani, R. Kustra, and R. Tibshirani, Delve Data Sets [Online]. Available: [Online]. Available: http://www.cs.toronto.edu/delve/data/datasets.html

[49] J. Alcala-Fdez, A. Fernandez, J. Luengo, J. Derrac, S. Garca, L. Snchez, and F. Herrera, "KEEL data-mining software tool: Data set repository, integration of algorithms and experimental analysis framework," *J. Multiple-Valued Logic Soft Comput.*, vol. 17, no. 2–3, pp. 255–287, 2011.

[50] L. Torgo, Regression Data Sets [Online]. Available: [Online]. Available: http://www.dcc.fc.up.pt/ltorgo/Regression/DataSets.html

[51] E. N. Lorenz, "Deterministic nonperiodic flow," *J. Atmosph. Sci.*, vol. 20, no. 2, pp. 130–141, 1963.

N. Denizcan Vanli was born in Nigde, Turkey, in 1990. He received the B.S. degree with high honors in electrical and electronics engineering from Bilkent University, Ankara, Turkey, in 2013.

He is currently working toward the M.S. degree in the Department of Electrical and Electronics Engineering at Bilkent University. His research interests include sequential learning, adaptive filtering, machine learning, and statistical signal processing.

Suleyman Serdar Kozat (A'10–M'11–SM'11) received the B.S. degree with full scholarship and high honors from Bilkent University, Turkey. He received the M.S. and Ph.D. degrees in electrical and computer engineering from University of Illinois at Urbana Champaign, Urbana, IL. Dr. Kozat is a graduate of Ankara Fen Lisesi.

After graduation, Dr. Kozat joined IBM Research, T. J. Watson Research Lab, Yorktown, New York, US as a Research Staff Member in the Pervasive Speech Technologies Group. While doing his Ph.D., he was also working as a Research Associate at Microsoft Research, Redmond, Washington, US in the Cryptography and Anti-Piracy Group. He holds several patent inventions due to his research accomplishments at IBM Research and Microsoft Research. After serving as an Assistant Professor at Koc University, Dr. Kozat is currently an Assistant Professor (with the Associate Professor degree) at the Electrical And Electronics Department of Bilkent University.

Dr. Kozat is the President of the IEEE Signal Processing Society, Turkey Chapter. He has been elected to the IEEE Signal Processing Theory and Methods Technical Committee and IEEE Machine Learning for Signal Processing Technical Committee, 2013. He has been awarded IBM Faculty Award by IBM Research in 2011, Outstanding Faculty Award by Koc University in 2011 (granted the first time in 16 years), Outstanding Young Researcher Award by the Turkish National Academy of Sciences in 2010, ODTU Prof. Dr. Mustafa N. Parlar Research Encouragement Award in 2011, Outstanding Faculty Award by Bilim Kahramanlari, 2013 and holds Career Award by the Scientific Research Council of Turkey, 2009.