

SEQUENTIAL NONLINEAR LEARNING

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

By
Nuri Denizcan Vanlı
August, 2015

Sequential Nonlinear Learning

By Nuri Denizcan Vanlı

August, 2015

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

Assoc. Prof. Dr. S. Serdar Kozat (Advisor)

Prof. Dr. A. Enis Çetin

Assoc. Prof. Dr. Çağatay Candan

Approved for the Graduate School of Engineering and Science:

Prof. Dr. Levent Onural
Director of the Graduate School

ABSTRACT

SEQUENTIAL NONLINEAR LEARNING

Nuri Denizcan Vanlı

M.S. in Electrical and Electronics Engineering

Advisor: Assoc. Prof. Dr. S. Serdar Kozat

August, 2015

We study sequential nonlinear learning in an individual sequence manner, where we provide results that are guaranteed to hold without any statistical assumptions. We address the convergence and undertraining issues of conventional nonlinear regression methods and introduce algorithms that elegantly mitigate these issues using nested tree structures. To this end, in the second chapter, we introduce algorithms that 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. In the third chapter, we propose an incremental decision tree structure and using this model, we introduce an online regression algorithm that partitions the regressor space in a data driven manner. We prove that the proposed algorithm sequentially and asymptotically achieves the performance of the optimal twice differentiable regression function for any data sequence with an unknown and arbitrary length. The computational complexity of the introduced algorithm is only logarithmic in the data length under certain regularity conditions. In the fourth chapter, we construct an online finite state (FS) predictor over hierarchical structures, whose computational complexity is only linear in the hierarchy level. We prove that the introduced algorithm asymptotically achieves the performance of the best linear combination of all FS predictors defined over the hierarchical model in a deterministic manner and in a mean square error sense in the steady-state for certain nonstationary models. In the fifth chapter, we introduce a distributed subgradient based extreme learning machine algorithm to train single hidden layer feedforward neural networks (SLFNs). We show that using the proposed algorithm, each of the individual SLFNs asymptotically achieves the performance of the optimal centralized batch SLFN in a strong deterministic sense.

Keywords: Sequential learning, nonlinear models, big data.

ÖZET

ARDIŞIK DOĞRUSAL OLMAYAN ÖĞRENME

Nuri Denizcan Vanlı
Elektrik Elektronik Mühendisliği, Yüksek Lisans
Tez Danışmanı: Doç. Dr. S. Serdar Kozat
Ağustos, 2015

Ardışık doğrusal olmayan öğrenme problemini bireysel dizi usulünde çalışmaktayız ve herhangi bir istatistiksel varsayım gerekmeksizin sağlanması garanti olan sonuçlar sunmaktayız. Geleneksel doğrusal olmayan bağlanım yöntemlerinin yakınsama ve seyrek öğrenme problemlerini ele almaktayız ve içiçe ağaç yapıları kullanarak bu problemleri zarif bir biçimde çözen algoritmalar sunmaktayız. Bu doğrultuda, ikinci bölümde, sadece bağlanım fonksiyonlarını değil, aynı zamanda bütün ağaç yapısını uyarlayan, çifte üstel sayıdaki bölüntülerin en iyi doğrusal kombinasyonunun performansına ulaşan ve hesaplama karmaşıklığı ağaçtaki düğüm sayısı ile sadece polinomsal olarak artan algoritmalar önermekteyiz. Üçüncü bölümde, artımlı karar ağacı yapısı önermekteyiz ve bu modeli kullanarak değişken uzayını veriye dayalı bir biçimde bölen bir çevrimiçi bağlanım algoritması sunmaktayız. Önerilen algoritmanın ardışık ve asimptotik olarak en iyi iki kez türevlenebilir bağlanım fonksiyonunun performansına uzunluğu bilinmeyen ve gelişigüzel olan tüm veri dizileri için ulaştığını ispatlamaktayız. Önerilen algoritmanın hesaplama karmaşıklığı, bazı düzenlilik koşulları altında, veri uzunluğunda sadece logaritmiktir. Dördüncü bölümde, sıradüzensel yapılar üzerinden, hesaplama karmaşıklığı sıradüzen seviyesiyle doğrusal olarak artan bir çevrimiçi sonlu durumlu (SD) öngörme algoritması oluşturmaktayız. Önerilen algoritmanın sıradüzensel yapı üzerinde tanımlı olan tüm SD öngörücülerin en iyi doğrusal kombinasyonunun performansına asimptotik olarak ulaştığını, belirlenimci çerçevede ve bazı durağan olmayan modeller için yatışkın durumda ortalama karesel hata çerçevesinde ispatlamaktayız. Beşinci bölümde, tek saklı katmanlı ileri beslemeli sinir ağlarını (TKİS) eğitmek için altbayır temelli dağıtılmış uç öğrenim makinesi algoritması önermekteyiz. Önerilen algoritmayı kullanarak her bir TKİS'in, en iyi merkezi toplu TKİS'in performansına asimptotik olarak güçlü belirlenimci çerçevede ulaştığını göstermekteyiz.

Anahtar sözcükler: Ardışık öğrenme, doğrusal olmayan modeller, büyük veri.

Acknowledgement

I would like to express my deepest gratitude to my advisor, Assoc. Prof. S. Serdar Kozat, for his excellent guidance, motivation, and enthusiasm. I attribute the level of this thesis to his continuous support throughout my M.S. study. I could not have imagined a better advisor for my M.S. study.

I would like to thank Assoc. Prof. Sinan Gezici for guiding my research in my undergraduate years. I would also like to thank Prof. Ezhan Karasan for his sincere guidance in the past several years.

I would like to thank TÜBİTAK for supporting me through BİDEB 2228-A and 2210-C Scholarship Programs.

Finally, I would like to thank my parents for supporting me throughout my life.

Contents

1	Introduction	1
2	Online Piecewise Linear Regression via Decision Adaptive Trees	8
2.1	Regression Using Specific Partitions	9
2.2	Regressor Space Partitioning via Hard Separator Functions	13
2.3	Proof of Theorem 2.1 and Construction of Algorithm 1	15
2.4	Regressor Space Partitioning via Adaptive Soft Separator Functions	22
2.4.1	Outline of the Proof of Theorem 2.3 and Construction of Algorithm 2	23
2.4.2	Selection of the Learning Rates	25
2.4.3	Selection of the Depth of the Tree	26
2.5	Simulations	27
2.5.1	Computational Complexities	28
2.5.2	Matched Partitions	30

2.5.3	Mismatched Partitions	33
2.5.4	Mismatched Partitions with Overfitting & Underfitting . .	36
2.5.5	Chaotic Signals	37
2.5.6	Benchmark Real and Synthetic Data	39
3	Predicting Nearly As Well As the Optimal Twice Differentiable Regressor	43
3.1	Nonlinear Regression via Incremental Decision Trees	46
3.1.1	Notation	47
3.1.2	Incremental Decision Trees	47
3.2	Main Results	50
3.3	Construction of the Algorithm and Proofs of the Theorems	53
3.4	Simulations	70
3.4.1	Synthetic Data	72
3.4.2	Chaotic Data	74
3.4.3	Benchmark Sequences	76
3.4.4	Real Data	78
4	Sequential Prediction over Hierarchical Structures	83
4.1	Finite State Predictors	83
4.2	Sequential Combination of FS Predictors	88

4.3	Efficient Sequential Combination of FS Predictors	91
4.3.1	A Recursive Calculation	92
4.3.2	Construction of the Final Predictor	94
4.3.3	A Low Complexity Sequential Update	95
4.3.4	Summary of the Algorithm	96
4.4	Positive and Negative Weights	97
4.5	Implementation of the Algorithm with Forgetting Factor	99
4.6	Simulations	100
4.6.1	Real Life Energy Profile Forecasting	100
4.6.2	Synthetic MSE Analysis	102
4.6.3	SETAR Time Series Prediction	103
5	Sequential Nonlinear Learning for Distributed Multi-Agent Systems via Feedforward Networks	107
5.1	Extreme Learning Machines	108
5.2	The Forward-Backward Splitting Method	111
5.3	Distributed Sequential Splitting Extreme Learning Machine (DSS-ELM)	112
5.4	Proofs of Theorem 5.1 & Corollary 5.2	115
5.5	Simulations	121
5.5.1	Stationary Scenario	121

5.5.2	Nonstationary Scenario	125
5.5.3	Real Data Sets	126
6	Conclusion	130
A	Proofs	141
A.1	Proof of Lemma 3.3	141
A.2	Proof of Lemma 3.5	143
A.3	Proof of Theorem 4.1	145
A.4	Proof of Lemma 4.2	148
A.5	Proof of Lemma 4.3	149
A.6	Proof of Lemma 4.4	149
A.7	Proof of Lemma 5.3	150

List of Figures

2.1	The partitioning of a two dimensional regressor space using a complete tree of depth-2.	9
2.2	All different partitions of the regressor space that can be obtained using a depth-2 tree.	10
2.3	Regression error performances for the second order piecewise linear model in (2.26) averaged over 10 trials.	30
2.4	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.	31
2.5	Regression error performances for the second order piecewise linear model in (2.27).	32
2.6	Changes in the boundaries of the leaf nodes of the depth-2 tree of the DAT algorithm for $t = 0, 1000, 2000, 5000, 20000, 50000$	33
2.7	Progress of the node weights for the piecewise linear model in (2.27) for (a) the DFT algorithm and (b) the DAT algorithm.	34
2.8	Regression error performances for (a) the first order piecewise linear model in (2.28) (b) the third order piecewise linear model in (2.29).	35

2.9	Regression error performances of the proposed algorithms for the chaotic process presented in (2.30).	38
2.10	Regression error performances for the chaotic signal generated from the Lorenz attractor in (2.31),(2.32), and (2.33) with parameters $dt = 0.01$, $\rho = 28$, $\sigma = 10$, and $\beta = 8/3$	39
2.11	Regression error performances for the real data set: California housing - estimation of the median house prices in the California area using California housing database.	40
3.1	The partitioning of a one dimensional regressor space, i.e., $[-A, A]$, using a depth-2 full decision tree, where each node represents a portion of the regressor space.	44
3.2	All different piecewise linear models that can be obtained using a depth-2 full decision tree, where the regressor space is one dimensional.	45
3.3	A sample evolution of the incremental decision tree, where the regressor space is one dimensional. The “ $\times$ ” marks on the regressor space represents the value of the regressor vector at that specific time instant. Light nodes are the ones having an index of 1, whereas the index of the dark nodes is 0.	48
3.4	Normalized accumulated squared error performances for the piecewise linear model in (3.40) averaged over 10 trials.	73
3.5	Evolution of the normalized cumulative node weights at the corresponding depths of the tree for the piecewise linear model in (3.40) averaged over 10 trials.	74
3.6	Normalized accumulated squared error performances for the chaotic data generated by the Duffing map in (3.41).	75

3.7	Normalized accumulated squared error performances for the chaotic data generated by the Tinkerbell map in (3.42) and (3.43).	76
3.8	Normalized accumulated squared error performances for the Mackey-Glass sequence in (3.44).	77
3.9	Normalized accumulated squared error performances for the Chua's circuit sequence in (3.45).	78
3.10	Normalized accumulated squared error performances for the "kinematics" data set.	79
3.11	Normalized accumulated squared error performances for the "pumadyn" data set.	80
4.1	FS Diagram for $l = 3$ and $h = 2$, where all allowable transitions are drawn.	84
4.2	All equivalence classes for the FS diagram with $l = 3$ and $h = 2$.	87
4.3	3 possible FS predictors for the equivalence classes in Figure 4.2 with $h = 3$.	88
4.4	Normalized cumulative square errors of the proposed algorithms for the real life electricity consumption data.	101
4.5	The experimental MSE of the proposed algorithm converges to the theoretical steady-state MSE performance. The results are averaged over 500 independent trials.	102
4.6	Normalized cumulative square errors of the proposed algorithms for the SETAR model in (4.35).	103
4.7	The actual and the predicted time series for the SETAR model in (4.35) over a data length of 1000.	104

4.8	Normalized cumulative square errors of the proposed algorithms for the SETAR models in (4.35) and (4.36). Here, the first 5000 samples of the data are generated using (4.35), whereas the last 5000 are generated using (4.36).	105
5.1	An example multi-agent network. Each agent is connected to and communicate with a set of other agents, which form its neighborhood.	108
5.2	Comparison of the algorithms for the linear regression model in (5.44) with mean square error loss and ℓ_1 norm regularization. . .	122
5.3	Comparison of the algorithms for the linear regression model in (5.44) with mean square error loss and ℓ_2^2 norm regularization. . .	123
5.4	Comparison of the algorithms for the nonlinear regression model in (5.46) with mean square error loss and ℓ_2^2 norm regularization. .	124
5.5	Comparison of the algorithms for the dynamic linear regression model in (5.44) and (5.47) with mean square error loss and ℓ_2^2 norm regularization.	125

List of Tables

2.1	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.	29
2.2	Time accumulated normalized errors of the proposed algorithms. Each dimension of the data sets is normalized between $[-1, 1]$. . .	39
3.1	Comparison of the complexities of the proposed algorithms with the corresponding update rules. In the table, p 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. For the MARS algorithm (particularly, the fast MARS algorithm), b represents the number of basis functions and w represents the window length.	71
3.2	Squared errors of the proposed algorithms for various benchmark data sets, where each dimension of the data sets are scaled between $[-1, 1]$	81
5.1	Specifications of Benchmark Data Sets.	127

5.2	Comparison of the RMSE performance of the algorithms.	128
5.3	RMSE performance of the DSS-ELM algorithm for different network sizes for the protein tertiary data with ℓ_2^2 -regularization. . .	129

Chapter 1

Introduction

Sequential nonlinear learning is extensively investigated in the signal processing [1–7] and machine learning literatures [8–10], especially for applications where linear modeling [11, 12] 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, 13–15], 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 [13, 14, 16]. 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 [17].

Our aim, in this context, is to estimate or model a desired sequence $\{d_t\}_{t \geq 1}$ by using a sequence of regressor vectors $\{\mathbf{x}_t\}_{t \geq 1}$. We seek to find the relationship, if it exists, between these two sequences, which is assumed to be unknown, nonlinear, and possibly time varying. In order to define and find this relationship between the desired sequence and regressor vectors, numerous methods such as neural networks [18, 19], Volterra filters [5], and B-splines [6] are used. However, either these methods are extremely difficult to use in real life applications due

to convergence issues, e.g., Volterra filters and B-splines, or it is quite hard to obtain a consistent performance in different scenarios, cf. [2, 8, 20, 21].

To overcome these difficulties, “tree” based nonlinear adaptive filters or regressors 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, 17, 22–24]. In its most basic form, a regression tree defines a hierarchical or nested partitioning of the regressor space [2]. According to this nested partitioning, 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 [17, 25].

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, 22]. Especially their 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 [17, 25]. 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 [17].

To address these issues, in Chapter 2, we provide a *comprehensive* solution to

nonlinear regression using decision trees. Particularly, 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, 11, 26] 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.

Specifically, 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 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, 27, 28] are artificially constraint to be positive and sum up to 1 [29], 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 Chapter 3, we propose an algorithm that alleviates the aforementioned issues by introducing hierarchical models that recursively and effectively partition the regressor space into subsequent regions in a data driven manner, where a different linear model is learned at each region. Unlike most of the nonlinear models, learning linear structures at each region can be efficiently managed. Hence, using this hierarchical piecewise model, we significantly mitigate the convergence and consistency issues. Furthermore, we prove that the resulting hierarchical piecewise model asymptotically achieves the performance of any twice differentiable regression function that is directly tuned to the underlying observations without any tuning of algorithmic parameters or without any assumptions on the data (other than an upper bound on the magnitude). Since most of the nonlinear modeling functions of the regression algorithms in the literature, such as neural networks and Volterra filters, can be accurately represented by twice differentiable functions [27, 28], our algorithm readily performs asymptotically as well as such nonlinear learning algorithms.

The introduced method sequentially and recursively divides the space of the regressors into disjoint regions according to the amount of the data in each region, instead of committing to a priori selected partition. In this sense, we avoid creating undertrained regions until a sufficient amount of data is observed. The nonlinear modeling power of the introduced algorithm is incremented (by consecutively partitioning the regressor space into smaller regions) as the observed

data length increases. The introduced method adapts itself according to the observed data instead of relying on ad-hoc parameters that are set while initializing the algorithm. Thus, the introduced algorithm provides a significantly stronger modeling power with respect to the state-of-the-art methods in the literature as shown in our experiments.

The main contributions of Chapter 3 are as follows. We introduce a sequential piecewise linear regression algorithm *i)* that provides a significantly improved modeling power by adaptively increasing the number of partitions according to the observed data, *ii)* that is highly efficient in terms of the computational complexity as well as the error performance, and *iii)* whose performance converges to *iii-a)* the performance of the optimal twice differentiable function that is selected in hindsight and *iii-b)* the best piecewise linear model defined on the incremental decision tree, with guaranteed upper bounds without any statistical or structural assumptions on the desired data as well as on the regressor vectors (other than an upper bound on them). Hence, unlike the state-of-the-art approaches whose performances usually depend on the initial construction of the tree, we introduce a method to construct a decision tree, whose depth (and structure) is adaptively incremented (and adjusted) in a data dependent manner, which we call an incremental decision tree. Furthermore, the introduced algorithm achieves this superior performance only with a computational complexity $O(\log(n))$ for any data length n , under certain regularity conditions. Even if these regularity conditions are not met, the introduced algorithm still achieves the performance of any twice differentiable regression function, however with a computational complexity linear in the data length.

In Chapter 4, we introduce truly sequential algorithms over any arbitrary hierarchical structure, with a computational complexity only linear in the hierarchy depth that *i)* asymptotically achieve the performance of the best FS predictor among the doubly exponential number of possible FS predictors in an individual sequence manner without any stochastic assumptions over any data length under a wide range of loss functions; *ii)* asymptotically achieve the performance of the best “linear combination” of all FS predictors define on the hierarchy in an

individual sequence manner over any data length under a wide range of loss functions; *iii*) achieve the mean square error (MSE) of the best linear combination of all FS filters or predictors in the steady-state [30] for certain nonstationary models [16, 31]. We emphasize that our algorithms are truly sequential such that they do need any a prior information on the underlying data sequence such as the sequence length, bounds on the sequence values or the statistical distribution of the data. In this sense, the introduced algorithm is suitable for big data and real life applications under both stationary and nonstationary settings. We also show that the weights of our algorithm converge to the minimum MSE (MMSE) optimal linear combination weights. Our approach is generic such that our algorithm can be applied to a wide range of hierarchical equivalence class definitions.

In Chapter 5, we consider neural network inspired learning structures. Although several neural-adaptive learning methods (e.g., [32–36]) are used for processing data in a centralized manner, the steadily increasing growth of the data sizes (in terms of both dimensionality and length) prohibit centralized processing due to computational complexity, storage and communication issues [37, 38]. To address this problem, several distributed learning algorithms are proposed in the machine learning and signal processing literatures [15, 39–43]. Although these algorithms are shown to achieve certain statistical and deterministic convergence rates, they are usually based on linear models, which significantly limits their performance in real life applications [20, 44, 45].

We resolve these issues by introducing a sequential nonlinear kernel-adaptive learning algorithm with guaranteed convergence bounds without any statistical assumptions. In particular, we provide a novel and scalable approach to nonlinear learning problems by presenting a complete distributed formulation of the learning structure, over which any ELM-based algorithm can be applied. We show that the computational complexity of the proposed algorithm is linear in the number of hidden nodes for each agent over the distributed network, whereas it is quadratic for the original ELM method [33]. Since the introduced algorithm directly decreases *i*) the amount of data to be processed at each agent and *ii*) the computational complexity of the processing algorithms at each agent, it is highly appealing for applications involving big data. Furthermore, our algorithm works

for a wide range of cost functions that are extensively used in signal processing and machine learning literatures including the squared error loss [21, 33, 46] and the absolute difference loss [36]. Our derivations can be extended to various learning problems such as classification [36] and ranking [47]. In this sense, this paper significantly contributes to the existing centralized ELM-based learning methods widely used in the literature by presenting a scalable distributed formulation that can incorporate various neural network based algorithms and nonlinear learning problems.

Our main contributions are as follows. *i)* We introduce a truly sequential nonlinear optimization algorithm over distributed multi-agent learning systems. Here, the multi-agent structure optimizes SFLNs for both additive and radial basis function (RBF) kernels in a fully distributed manner. The proposed algorithm is truly sequential such that it processes each new data pair and update the SLFN model without knowledge of the time horizon. *ii)* We show that by our diffusion scheme, each agent can successfully and uniformly optimize the SFLN weights to minimize the overall network cost (over the entire data) with observing only a portion of the data. We demonstrate this result in a strong mathematical sense without any statistical assumptions on the data such that our results are guaranteed to hold uniformly for all input and output sequences. *iii)* We achieve this performance in a highly efficient manner with a computational complexity only linear in the data length. Thus, our algorithm is suitable for applications involving big data. *iv)* We demonstrate the significant performance gains of our algorithm over numerical examples and benchmark real data sets.

Notation: Throughout the paper, all vectors are column vectors and represented by boldface lowercase letters. Matrices are represented by boldface uppercase letters. For a matrix $\mathbf{H}$, $\|\mathbf{H}\|_F$ is the Frobenius norm. For a vector $\mathbf{x}$ (and matrix $\mathbf{H}$), $\|\mathbf{x}\|$ (and $\|\mathbf{H}\|$) is the ℓ^2 -norm. For two vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^m$, $\langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{x}^T \mathbf{y}$ is the inner product. Here, $\mathbf{0}$ (and $\mathbf{1}$) denotes the vector with all zeros (and ones) and the dimensions can be understood from the context. For a matrix $\mathbf{H}$, H_{jk} represents its entry at the j th row and k th column.

Chapter 2

Online Piecewise Linear Regression via Decision Adaptive Trees

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

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

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., $\mathbf{x}_t \in \mathbb{R}^m$, 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 [17], 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.

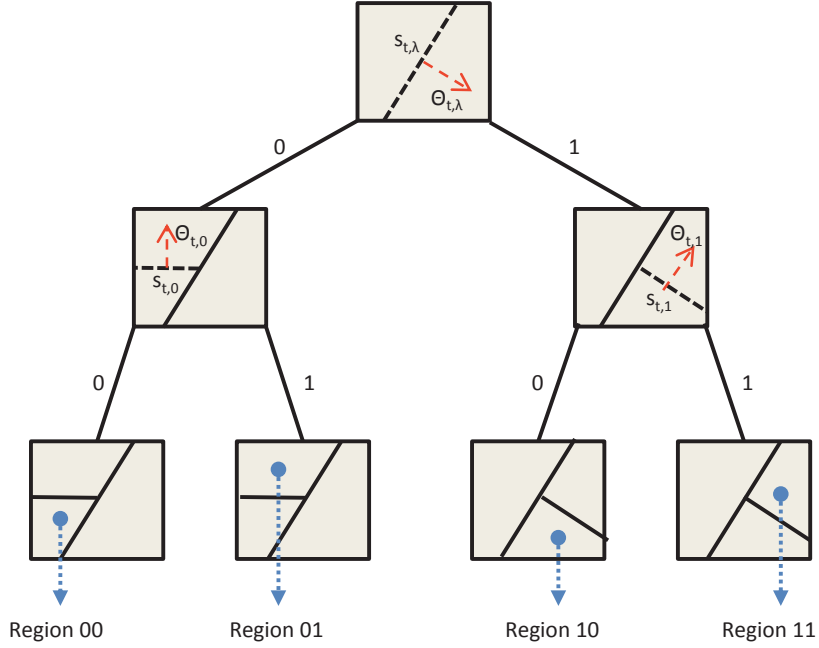


Figure 2.1: The partitioning of a two dimensional regressor space using a complete tree of depth-2.

2.1 Regression Using Specific Partitions

To clarify the framework, suppose the corresponding space of regressor vectors is two dimensional, i.e., $\mathbf{x}_t \in \mathbb{R}^2$, and we partition this regressor space using a depth-2 tree as in Figure 2.1. 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 $\theta_{t,\lambda}$, $\theta_{t,0}$ and $\theta_{t,1}$, respectively (See Figure 2.1). 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 Figure 2.1 such that the partitioning is defined in a hierarchical manner, i.e., $\mathbf{x}_t$ 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 Figure 2.2, 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 [48].

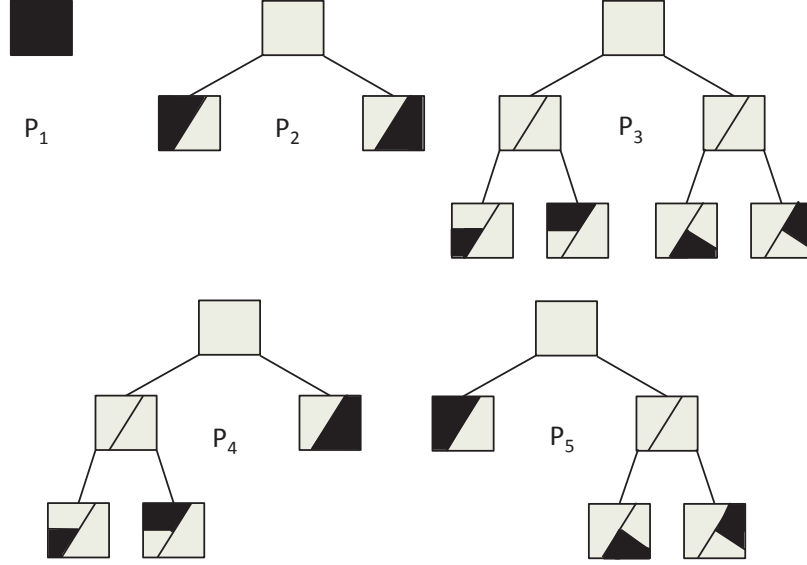


Figure 2.2: All different partitions of the regressor space that can be obtained using a depth-2 tree.

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 Figure 2.1. The separating functions can also be soft. As an example, we use the logistic regression classifier [49]

$$s_t = \frac{1}{1 + e^{\mathbf{x}_t^T \boldsymbol{\theta}_t + b_t}}, \quad (2.1)$$

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 (2.1) can be rewritten as

$$s_t = \frac{1}{1 + e^{\mathbf{x}_t^T \boldsymbol{\theta}_t}}, \quad (2.2)$$

where $\mathbf{x}_t = [\mathbf{x}_t^T, 1]^T$. One can easily use other differentiable soft separating functions in this setup in a straightforward manner as remarked later in this chapter.

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. Throughout this chapter, we assign linear regressors to each region. For instance consider the third model in Figure 2.2, i.e., P_3 , where this partition is the union of 4 regions each corresponding to a leaf of the original complete tree in Figure 2.1, 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

$$\hat{d}_{t,00} = \mathbf{x}_t^T \mathbf{v}_{t,00}, \quad (2.3)$$

where $\mathbf{v}_{t,00} \in \mathbb{R}^m$ 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

$$\hat{d}_t = s_{t,\lambda} s_{t,0} \hat{d}_{t,00} + s_{t,\lambda} (1 - s_{t,0}) \hat{d}_{t,01} + (1 - s_{t,\lambda}) s_{t,1} \hat{d}_{t,10} + (1 - s_{t,\lambda}) (1 - s_{t,1}) \hat{d}_{t,11}, \quad (2.4)$$

for any $\mathbf{x}_t$. We emphasize that any P_i , $i = 1, \dots, 5$ 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 [29, 50–52], we update the regressor of the node “00” as

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

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

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

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

$$\frac{\partial s_{t,0}}{\partial \boldsymbol{\theta}_{t,0}} = \frac{-\mathbf{x}_t e^{\mathbf{x}_t^T \boldsymbol{\theta}_{t,0}}}{(1 + e^{\mathbf{x}_t^T \boldsymbol{\theta}_{t,0}})^2}, \quad (2.5)$$

according to the separator function in (2.2). 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 (2.5).

Until now a specific partition, i.e., P_3 , is used to construct a piecewise linear regressor, although the tree can represent P_i , $i = 1, \dots, 5$. However, since the data structure is unknown, one may not prefer a particular model [4, 27, 28], 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 Figure 2.2) 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 2.2, 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 2.4, 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 $O(m4^d)$.

2.2 Regressor Space Partitioning via Hard Separator Functions

In this section, we consider the regression problem in which the sequential regressors (as described in Section 2.1) 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 [17]. While the k th model (i.e., P_k partition) generates the regression output $\hat{d}_t^{(k)}$ at time t for all $k = 1, \dots, \beta_d$, we linearly combine these estimates using the weighting vector $\mathbf{w}_t \triangleq [w_t^{(1)}, \dots, w_t^{(\beta_d)}]^T$ such that the final estimate of our algorithm at time t is given as

$$\hat{d}_t \triangleq \sum_{k=1}^{\beta_d} w_t^{(k)} \hat{d}_t^{(k)} = \mathbf{w}_t^T \hat{\mathbf{d}}_t, \quad (2.6)$$

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

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

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 the nodes of the tree.

Theorem 2.1 *Let $\{d_t\}_{t \geq 1}$ and $\{\mathbf{x}_t\}_{t \geq 1}$ be arbitrary, bounded, and real-valued sequences. The algorithm $\hat{d}_t$ given in Algorithm 1 when applied to these data sequence yields*

$$\sum_{t=1}^n (d_t - \hat{d}_t)^2 - \min_{\mathbf{w} \in \mathbb{R}^{\beta_d}} \sum_{t=1}^n (d_t - \mathbf{w}^T \hat{\mathbf{d}}_t)^2 \leq O(\ln(n)), \quad (2.7)$$

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

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 separation functions, e.g., [17], 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 [13, 30] such that the expectation of $\hat{d}_t^{(k)}$, $k = 1, \dots, \beta_d$, and $\mathbf{w}_t$ are separable, we have the following theorem.

Theorem 2.2 *Assuming that the partition regressors, i.e., $\hat{d}_t^{(k)}$, $k = 1, \dots, \beta_d$, and d_t converge to zero mean stationary distributions, we have*

$$\lim_{t \rightarrow \infty} E[e_t^2] = J^* + \frac{\mu J^* \text{tr}(\mathbf{D})}{2 - \mu \text{tr}(\mathbf{D})},$$

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

$$J^* \triangleq \min_{\mathbf{w} \in \mathbb{R}^{\beta_d}} \lim_{t \rightarrow \infty} E[(d_t - \mathbf{w}^T \hat{\mathbf{d}}_t)^2],$$

and

$$\mathbf{D} \triangleq \lim_{t \rightarrow \infty} E \left[\hat{\mathbf{d}}_t \hat{\mathbf{d}}_t^T \right],$$

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

Theorem 2.2 directly follows Chapter 6 of [13] since we use a stochastic gradient algorithm to merge the partition regressors [13, 30]. 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.

2.3 Proof of Theorem 2.1 and Construction of Algorithm 1

To construct the final algorithm, we first introduce a “direct” algorithm which achieves the corresponding bound in Theorem 2.1. 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 $\hat{d}_t^{(k)}$, $k = 1, \dots, \beta_d$, are obtained as described in Section 2.1. To achieve the upper bound in (2.7), we use the stochastic gradient descent approach and update the combination weights as

$$\begin{aligned}\mathbf{w}_{t+1} &= \mathbf{w}_t - \frac{1}{2}\mu_t \nabla e_t^2(\mathbf{w}_t) \\ &= \mathbf{w}_t + \mu_t e_t \hat{\mathbf{d}}_t,\end{aligned}\tag{2.8}$$

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

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

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

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

Following [50], using Taylor series approximation, for some point $\mathbf{z}_t$ on the line segment connecting $\mathbf{w}_t$ to $\mathbf{w}_n^*$, we have

$$e_t^2(\mathbf{w}_n^*) = e_t^2(\mathbf{w}_t) + (\nabla e_t^2(\mathbf{w}_t))^T (\mathbf{w}_n^* - \mathbf{w}_t) + \frac{1}{2}(\mathbf{w}_n^* - \mathbf{w}_t)^T \nabla^2 e_t^2(\mathbf{z}_t)(\mathbf{w}_n^* - \mathbf{w}_t).$$

According to the update rule in (2.8), at each iteration the update on weights are

performed as $\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{\mu_t}{2} \nabla e_t^2(\mathbf{w}_t)$. Hence, we have

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

Then we obtain

$$(\nabla e_t^2(\mathbf{w}_t))^T (\mathbf{w}_t - \mathbf{w}_n^*) = \frac{\|\mathbf{w}_t - \mathbf{w}_n^*\|^2 - \|\mathbf{w}_{t+1} - \mathbf{w}_n^*\|^2}{\mu_t} + \mu_t \frac{\|\nabla e_t^2(\mathbf{w}_t)\|^2}{4}. \quad (2.9)$$

Under the mild assumptions that $\|\nabla e_t^2(\mathbf{w}_t)\|^2 \leq A^2$ for some $A > 0$ and $e_t^2(\mathbf{w}_n^*)$ is λ -strong convex for some $\lambda > 0$ [50], we achieve the following upper bound

$$e_t^2(\mathbf{w}_t) - e_t^2(\mathbf{w}_n^*) \leq \frac{\|\mathbf{w}_t - \mathbf{w}_n^*\|^2 - \|\mathbf{w}_{t+1} - \mathbf{w}_n^*\|^2}{\mu_t} - \frac{\lambda}{2} \|\mathbf{w}_t - \mathbf{w}_n^*\|^2 + \mu_t \frac{A^2}{4}. \quad (2.10)$$

By selecting $\mu_t = 2/(\lambda t)$ and summing up the regret terms in (2.10), we get

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

Note that (2.8) 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 (2.8) requires a computational complexity of $O(2^{2^d})$ since the vector $\mathbf{w}_t$ has a size of $O(2^{2^d})$. We next illustrate an algorithm that performs the same adaptation in (2.8) with a complexity of $O(d2^d)$.

We next introduce a labeling for the tree nodes following [48]. 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 $p' = q'_1 \dots q'_{l'}$ is a prefix to string $p = q_1 \dots q_l$ if $l' \leq l$ and $q'_i = q_i$ for all $i = 1, \dots, l'$, and the empty string λ is a prefix to all strings. Let $\mathcal{P}(p)$ represent all prefixes to the string p , i.e., $\mathcal{P}(p) \triangleq \{\nu_1, \dots, \nu_{l+1}\}$, 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 $i = 1, \dots, l + 1$.

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 $\mathbf{x}_t$ has fallen, the final estimate will be calculated with the separator functions. As an example, for the second model in Figure 2.2 (i.e., P_2 partition), say $\mathbf{x}_t \in \mathcal{R}_{00}$, 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 k th 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 k th model at time t as

$$\hat{d}_t^{(k)} = \sum_{p \in \mathcal{M}_k} \left(\hat{d}_{t,p} \prod_{i=1}^{l(p)} s_{t,\nu_i}^{q_i} \right), \quad (2.11)$$

where $\mathcal{M}_k$ is the set of all leaf nodes in the k th 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 i th 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

$$s_{t,\nu_i}^{q_i} \triangleq \begin{cases} s_{t,\nu_i}, & \text{if } q_i = 0 \\ 1 - s_{t,\nu_i}, & \text{otherwise} \end{cases} \quad (2.12)$$

with s_{t,ν_i} defined as in (2.2). 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 Figure 2.2 as the k th 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

$$\begin{aligned}
\hat{d}_t^{(k)} &= \sum_{p \in \mathcal{M}_k} \left(\hat{d}_{t,p} \prod_{i=1}^{l(p)} s_{t,\nu_i}^{q_i} \right) \\
&= \hat{d}_{t,00} s_{t,0}^0 s_{t,\lambda}^0 + \hat{d}_{t,01} s_{t,0}^1 s_{t,\lambda}^0 + \hat{d}_{t,10} s_{t,1}^0 s_{t,\lambda}^1 + \hat{d}_{t,11} s_{t,1}^1 s_{t,\lambda}^1 \\
&= \hat{d}_{t,00} s_{t,0} s_{t,\lambda} + \hat{d}_{t,01} (1 - s_{t,0}) s_{t,\lambda} \\
&\quad + \hat{d}_{t,10} s_{t,1} (1 - s_{t,\lambda}) + \hat{d}_{t,11} (1 - s_{t,1}) (1 - s_{t,\lambda}). \tag{2.13}
\end{aligned}$$

Note that (2.4) and (2.13) are the same special cases of (2.11).

We next denote the product terms in (2.11) as follows

$$\hat{\delta}_{t,p} \triangleq \hat{d}_{t,p} \prod_{i=1}^{l(p)} s_{t,\nu_i}^{q_i}, \tag{2.14}$$

to simplify the notation. Here, $\hat{\delta}_{t,p}$ can be viewed as the estimate of the node (i.e., region) p given that $\mathbf{x}_t \in \mathcal{R}_{p'}$ 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 (2.11) 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 k th 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 $k = 1, \dots, \beta_d$. 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

$$w_{t+1,p} \triangleq w_{t,p} + \mu_t e_t \hat{\delta}_{t,p}, \tag{2.15}$$

where $\hat{\delta}_{t,p}$ is defined as in (2.14).

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 (2.15) 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 $\mathbf{x}_t$ has fallen into the region $\mathcal{R}_{p'}$ specified by the node $p' \in \mathcal{L}_d$. Consider the node regressor defined in (2.14) for some node $p \in \mathcal{N}_d$. Since we are using hard separator functions, we obtain

$$\hat{\delta}_{t,p} = \begin{cases} \hat{d}_{t,p}, & \text{if } p \in \mathcal{P}(p') \\ 0, & \text{otherwise} \end{cases}, \quad (2.16)$$

where $\mathcal{P}(p')$ represents all prefixes to the string p' , i.e., $\mathcal{P}(p') = \{\nu'_1, \dots, \nu'_{d+1}\}$. 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 (2.6) 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 $\gamma_d(0) = 1$.

Algorithm 1 Decision Fixed Tree (DFT) Regressor

```

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

```

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., $\hat{d}_t = \mathbf{w}_t^T \hat{\mathbf{d}}_t$. We emphasize that the sizes of the vectors $\mathbf{w}_t$ and $\hat{\mathbf{d}}_t$ 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.

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

$$\begin{aligned}
\hat{d}_t &= \sum_{k=1}^{\beta_d} w_t^{(k)} \hat{d}_t^{(k)} \\
&= \sum_{k=1}^{\beta_d} \sum_{p \in \mathcal{M}_k} w_{t,p} \hat{d}_{t,p_k},
\end{aligned} \tag{2.17}$$

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 k th model, i.e., $p_k \triangleq \mathcal{P}(p') \cap \mathcal{M}_k$. Let $\mathcal{P}(p') = \{\nu'_1, \dots, \nu'_{d+1}\}$ 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 (2.17), we first let $\mathcal{S}_d(p) \triangleq \{\dot{p} \in \mathcal{N}_d \mid \mathcal{P}(\dot{p}) = p\}$, 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 $\rho(p, \dot{p})$ for arbitrary two nodes $p, \dot{p} \in \mathcal{N}_d$, as the number of models having both p and $\dot{p}$ as its leaf nodes. Trivially, if $\dot{p} = p$, then $\rho(p, p) = \gamma_d(l(p))$. If $p \neq \dot{p}$, then letting $\bar{p}$ denote the longest prefix to both p and $\dot{p}$, i.e., the longest string in $\mathcal{P}(p) \cap \mathcal{P}(\dot{p})$, we obtain

$$\rho(p, \dot{p}) \triangleq \begin{cases} \gamma_d(l(p)), & \text{if } p = \dot{p} \\ \frac{\gamma_d(l(p))\gamma_{d-l(\bar{p})-1}(l(\dot{p})-l(\bar{p})-1)}{\beta_{d-l(\bar{p})-1}}, & \text{if } p_1 \notin \mathcal{P}(\dot{p}) \cup \mathcal{S}_d(\dot{p}). \\ 0, & \text{otherwise} \end{cases} \quad (2.18)$$

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

Now turning our attention back to (2.17) and considering the definition in (2.18), 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

$$\kappa_{t,p} \triangleq \sum_{\dot{p} \in \mathcal{N}_d} \rho(p, \dot{p}) w_{t,\dot{p}}. \quad (2.19)$$

Then, the final estimate of our algorithm becomes

$$\hat{d}_t = \sum_{\nu'_j \in \mathcal{P}(p')} \kappa_{t,\nu'_j} \hat{d}_{t,\nu'_j}. \quad (2.20)$$

We emphasize that the estimate of our algorithm given in (2.20) achieves the exact same result with $\hat{d}_t = \mathbf{w}_t^T \hat{\mathbf{d}}_t$ with a computational complexity of $O(d2^d)$. Hence, the proof is concluded. $\square$

2.4 Regressor Space Partitioning via Adaptive Soft Separator Functions

In this section, the sequential regressors (as described in Section 2.1) for all partitions in the doubly exponential tree class are combined when soft separation functions are used, i.e., $s_t = \left(1 + e^{\mathbf{x}_t^T \boldsymbol{\theta}_t}\right)^{-1}$, where $\mathbf{x}_t \in \mathbb{R}^{m+1}$ 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 2.2, 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 2.3.

The computational complexity of the algorithm of Theorem 2.3 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 $\mathbf{x}_t \in \mathcal{R}_p$ 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 $\mathbf{x}_t \in \mathbb{R}^m$. Therefore, considering the cross-correlation of the final estimates of every node, we get a computational complexity of $O(m4^d)$.

Theorem 2.3 *Let $\{d_t\}_{t \geq 1}$ and $\{\mathbf{x}_t\}_{t \geq 1}$ be arbitrary, bounded, and real-valued sequences. The algorithm $\hat{d}_t$ given in Algorithm 2 when applied these sequences*

yields

$$\sum_{t=1}^n (d_t - \hat{d}_t)^2 - \min_{\mathbf{w} \in \mathbb{R}^{\beta_d}} \sum_{t=1}^n (d_t - \mathbf{w}^T \hat{\mathbf{d}}_t)^2 \leq O(\ln(n)), \quad (2.21)$$

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

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.

2.4.1 Outline of the Proof of Theorem 2.3 and Construction of Algorithm 2

The proof of the upper bound in Theorem 2.3 follows similar lines to the proof of upper bound in Theorem 2.1, 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 2.2, the final estimate of the k th 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 k th 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 (2.18)). Hence, according to the notation derived in (2.18) and (2.19), we obtain the final estimate of our algorithm as follows

$$\hat{d}_t = \sum_{p \in \mathcal{N}_d} \kappa_{t,p} \hat{\delta}_{t,p}. \quad (2.22)$$

Note that (2.22) is equal to $\hat{d}_t = \mathbf{w}_t^T \hat{\mathbf{d}}_t$ with a computational complexity of $O(4^d)$.

Unlike Section 2.2, 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

$$\boldsymbol{\theta}_{t+1,p} = \boldsymbol{\theta}_{t,p} - \frac{1}{2} \eta_t \nabla e_t^2(\boldsymbol{\theta}_{t,p}), \quad (2.23)$$

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

$$\begin{aligned} \boldsymbol{\theta}_{t+1,p} &= \boldsymbol{\theta}_{t,p} + \eta_t e_t \frac{\partial \hat{d}_t}{\partial s_{t,p}} \frac{\partial s_{t,p}}{\partial \boldsymbol{\theta}_{t,p}}, \\ &= \boldsymbol{\theta}_{t,p} + \eta_t e_t \left\{ \sum_{\acute{p} \in \mathcal{N}_d} \kappa_{t,\acute{p}} \frac{\partial \hat{\delta}_{t,\acute{p}}}{\partial s_{t,p}} \right\} \frac{\partial s_{t,p}}{\partial \boldsymbol{\theta}_{t,p}} \\ &= \boldsymbol{\theta}_{t,p} + \eta_t e_t \left\{ \sum_{q=0}^1 \sum_{\acute{p} \in \mathcal{S}_d(pq)} (-1)^q \kappa_{t,\acute{p}} \frac{\hat{\delta}_{t,\acute{p}}}{s_{t,p}^q} \right\} \frac{\partial s_{t,p}}{\partial \boldsymbol{\theta}_{t,p}}, \end{aligned} \quad (2.24)$$

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

$$\begin{aligned} \frac{\partial s_{t,p}}{\partial \boldsymbol{\theta}_{t,p}} &= - (1 + \exp(\mathbf{x}_t^T \boldsymbol{\theta}_{t,p}))^{-2} \exp(\mathbf{x}_t^T \boldsymbol{\theta}_{t,p}) \mathbf{x}_t \\ &= -s_{t,p}(1 - s_{t,p}) \mathbf{x}_t. \end{aligned} \quad (2.25)$$

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 (2.24) and (2.25).

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

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

According to the update rule in (2.24), 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. $\square$

2.4.2 Selection of the Learning Rates

We emphasize that the learning rate μ_t can be set according to the similar studies in the literature [13, 50] 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 $\mathbf{x}_t$ is sufficiently away from the region boundaries $s_{t,p}$, it is either close to s^+ or $1 - s^+$. However, when $\mathbf{x}_t$ 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 $\mathbf{x}_t$ falls on the boundary of multiple region crossings, e.g., say

$\mathbf{x}_t = [0, 0]^T$ 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.

2.4.3 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.

2.5 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 [53–55], elevators [53], kinematics [54], pumadyn [54], and bank [55] (which will be explained in detail in Subsection 2.5.6).

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 $\hat{d}_{t,1}, \dots, \hat{d}_{t,p}$, and a fixed Gaussian mixture weighting (that is selected according to the underlying sequence in hindsight), giving

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

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

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

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

For a fair performance comparison, in the corresponding experiments in Subsections 2.5.5 and 2.5.6, 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 2.5.2, 2.5.3, and 2.5.4, 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 2.5.2, 2.5.3, and 2.5.4, 2 for the experiments in Subsection 2.5.5, and 1 for the experiments in Subsection 2.5.6. 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 $\boldsymbol{\theta}_{t,p} = [\theta_{t,p}^{(1)}, \dots, \theta_{t,p}^{(m)}]^T$ 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 experiments. We emphasize that both these parameters and the learning rates of these algorithms are selected to give equal rate of performance and convergence.

2.5.1 Computational Complexities

As can be observed from Table 2.1, among the tree based algorithms that partition the regressor space, the CTW algorithm has the smallest complexity since at each

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

Table 2.1: 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.

time t , it only associates the regressor vector $\mathbf{x}_t$ with $O(d)$ nodes (the leaf node $\mathbf{x}_t$ 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 r th (and up to r th) 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

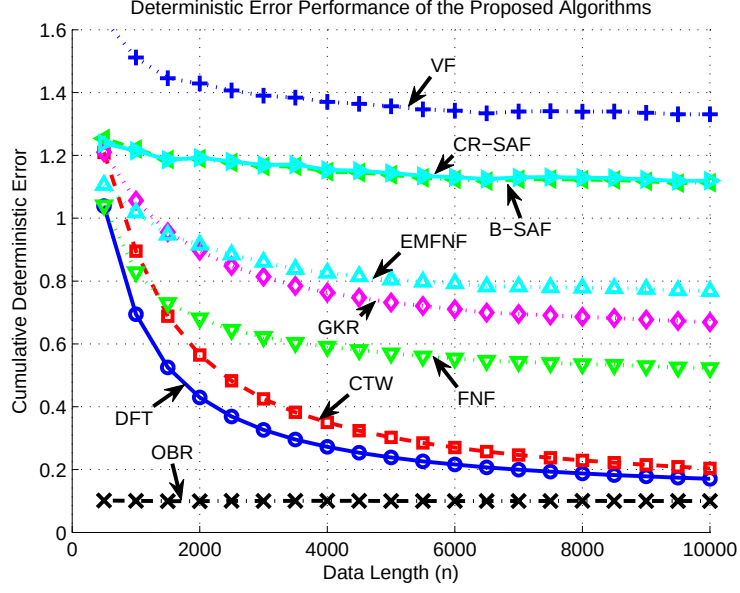


Figure 2.3: Regression error performances for the second order piecewise linear model in (2.26) averaged over 10 trials.

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.

2.5.2 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

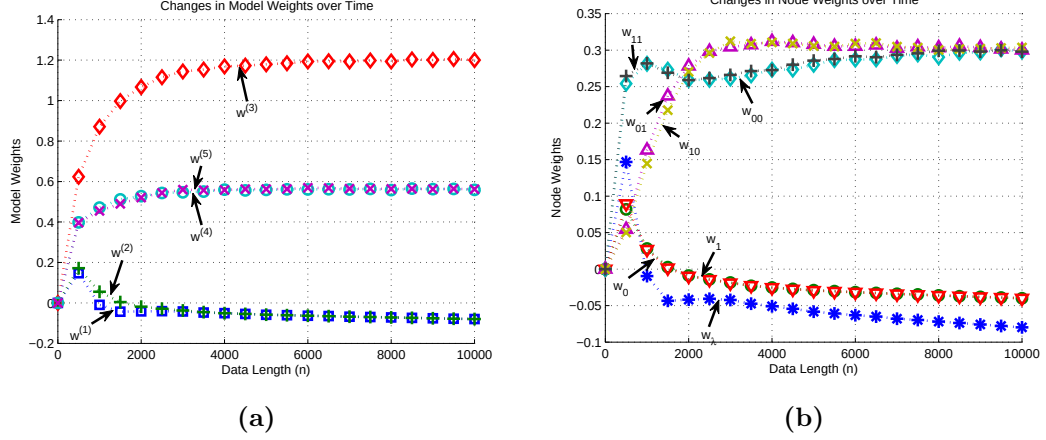


Figure 2.4: Progress of (a) the model weights and (b) the node weights averaged over 10 trials for the DFT algorithm. Note that the model weights do not sum up to 1.

linear model

$$d_t = \begin{cases} \mathbf{w}^T \mathbf{x}_t + \pi_t, & \text{if } \phi_0^T \mathbf{x}_t \geq 0 \text{ and } \phi_1^T \mathbf{x}_t \geq 0 \\ -\mathbf{w}^T \mathbf{x}_t + \pi_t, & \text{if } \phi_0^T \mathbf{x}_t \geq 0 \text{ and } \phi_1^T \mathbf{x}_t < 0 \\ -\mathbf{w}^T \mathbf{x}_t + \pi_t, & \text{if } \phi_0^T \mathbf{x}_t < 0 \text{ and } \phi_1^T \mathbf{x}_t \geq 0 \\ \mathbf{w}^T \mathbf{x}_t + \pi_t, & \text{if } \phi_0^T \mathbf{x}_t < 0 \text{ and } \phi_1^T \mathbf{x}_t < 0 \end{cases}, \quad (2.26)$$

where $\mathbf{w} = [1, 1]^T$, $\phi_0 = [1, 0]^T$, $\phi_1 = [0, 1]^T$, $\mathbf{x}_t = [x_{1,t}, x_{2,t}]^T$, 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 $\mathbf{I}_2$. The desired data at time t is denoted as d_t whereas the extended regressor vector is $\mathbf{x}_t = [x_{1,t}, x_{2,t}, 1]^T$, 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 $\boldsymbol{\Sigma}_i = 1.2 \times \mathbf{I}_2$ for $i = 1, \dots, 4$, are set to exactly match the underlying partitioning that generates the desired data.

In Figure 2.3, 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,

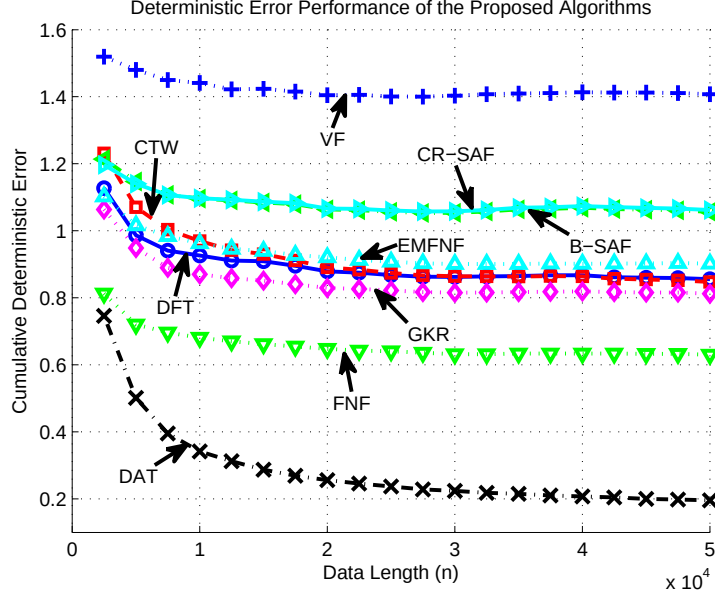


Figure 2.5: Regression error performances for the second order piecewise linear model in (2.27).

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 matches with the underlying partition in (2.26), 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 Figure 2.4a, 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 Figure 2.4a. In Figure 2.4b, 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

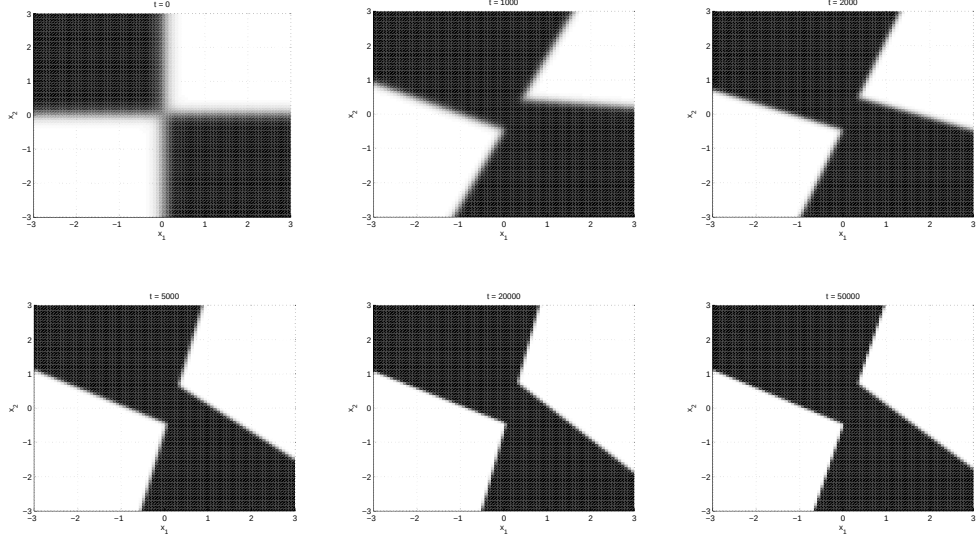


Figure 2.6: Changes in the boundaries of the leaf nodes of the depth-2 tree of the DAT algorithm for $t = 0, 1000, 2000, 5000, 20000, 50000$.

of the other nodes decrease. We also point out that although the tree based algorithms [4, 27, 28] 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.

2.5.3 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

$$d_t = \begin{cases} \mathbf{w}^T \mathbf{x}_t + \pi_t, & \text{if } \phi_0^T \mathbf{x}_t \geq 0.5 \text{ and } \phi_1^T \mathbf{x}_t \geq 1 \\ -\mathbf{w}^T \mathbf{x}_t + \pi_t, & \text{if } \phi_0^T \mathbf{x}_t \geq 0.5 \text{ and } \phi_1^T \mathbf{x}_t < 1 \\ -\mathbf{w}^T \mathbf{x}_t + \pi_t, & \text{if } \phi_0^T \mathbf{x}_t < 0.5 \text{ and } \phi_2^T \mathbf{x}_t \geq -1 \\ \mathbf{w}^T \mathbf{x}_t + \pi_t, & \text{if } \phi_0^T \mathbf{x}_t < 0.5 \text{ and } \phi_2^T \mathbf{x}_t < -1 \end{cases}, \quad (2.27)$$

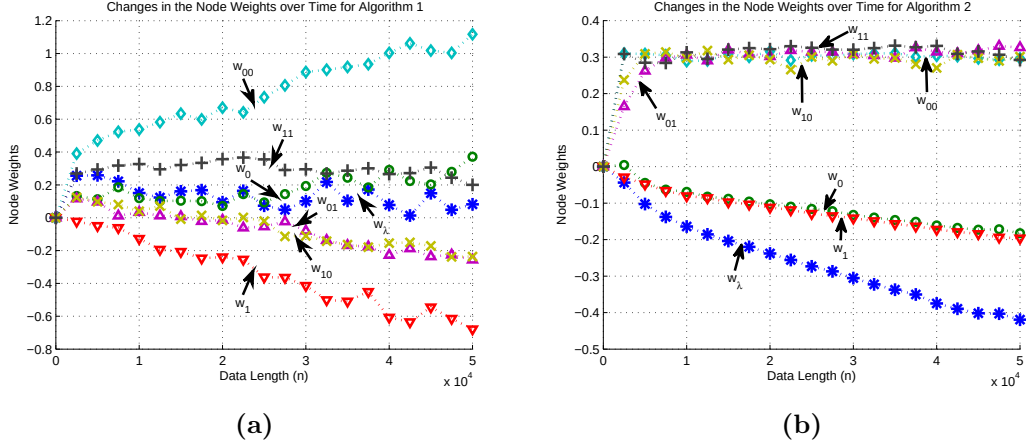


Figure 2.7: Progress of the node weights for the piecewise linear model in (2.27) for (a) the DFT algorithm and (b) the DAT algorithm.

where $\mathbf{w} = [1, 1]^T$, $\phi_0 = [4, -1]^T$, $\phi_1 = [1, 1]^T$, $\phi_2 = [1, 2]^T$, $\mathbf{x}_t = [x_{1,t}, x_{2,t}]^T$, 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 $\mathbf{I}_2$. 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, the mass points of the GKR are set to $\mu_1 = [1.4565, 1.0203]^T$, $\mu_2 = [0.6203, -0.4565]^T$, $\mu_3 = [-0.5013, 0.5903]^T$, and $\mu_4 = [-1.0903, -1.0013]^T$ with the same covariance matrix in the previous example.

Figure 2.5 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 Figure 2.3 and Figure 2.5, 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 as its leaf nodes). However that regressor is sub-optimal

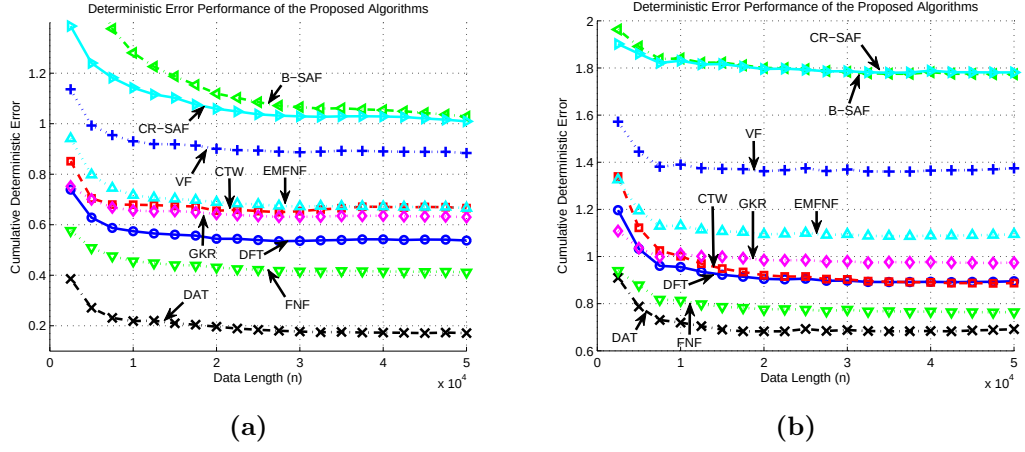


Figure 2.8: Regression error performances for (a) the first order piecewise linear model in (2.28) (b) the third order piecewise linear model in (2.29).

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. Figure 2.6 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 Figure 2.7a and Figure 2.7b, respectively. In Figure 2.7a, 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 Figure 2.7b, 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.

2.5.4 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

$$d_t = \begin{cases} \mathbf{w}^T \mathbf{x}_t + \pi_t, & \text{if } \phi_0^T \mathbf{x}_t \geq 0.5 \\ -\mathbf{w}^T \mathbf{x}_t + \pi_t, & \text{if } \phi_0^T \mathbf{x}_t < 0.5 \end{cases}, \quad (2.28)$$

and the third order piecewise linear model is defined as

$$d_t = \begin{cases} \mathbf{w}^T \mathbf{x}_t + \pi_t, & \text{if } \phi_0^T \mathbf{x}_t \geq 0.5, \phi_1^T \mathbf{x}_t \geq 1, \phi_3^T \mathbf{x}_t \geq 0.5 \\ -\mathbf{w}^T \mathbf{x}_t + \pi_t, & \text{if } \phi_0^T \mathbf{x}_t \geq 0.5, \phi_1^T \mathbf{x}_t \geq 1, \phi_3^T \mathbf{x}_t < 0.5 \\ \mathbf{w}^T \mathbf{x}_t + \pi_t, & \text{if } \phi_0^T \mathbf{x}_t \geq 0.5, \phi_1^T \mathbf{x}_t < 1, \phi_4^T \mathbf{x}_t \geq 0.5 \\ -\mathbf{w}^T \mathbf{x}_t + \pi_t, & \text{if } \phi_0^T \mathbf{x}_t \geq 0.5, \phi_1^T \mathbf{x}_t < 1, \phi_4^T \mathbf{x}_t < 0.5 \\ \mathbf{w}^T \mathbf{x}_t + \pi_t, & \text{if } \phi_0^T \mathbf{x}_t < 0.5, \phi_2^T \mathbf{x}_t < 0.5, \phi_5^T \mathbf{x}_t < 0.5 \\ -\mathbf{w}^T \mathbf{x}_t + \pi_t, & \text{if } \phi_0^T \mathbf{x}_t < 0.5, \phi_2^T \mathbf{x}_t < 0.5, \phi_5^T \mathbf{x}_t \geq 0.5 \\ \mathbf{w}^T \mathbf{x}_t + \pi_t, & \text{if } \phi_0^T \mathbf{x}_t < 0.5, \phi_2^T \mathbf{x}_t \geq 0.5, \phi_6^T \mathbf{x}_t < 0.5 \\ -\mathbf{w}^T \mathbf{x}_t + \pi_t, & \text{if } \phi_0^T \mathbf{x}_t < 0.5, \phi_2^T \mathbf{x}_t \geq 0.5, \phi_6^T \mathbf{x}_t \geq 0.5 \end{cases}, \quad (2.29)$$

where $\mathbf{w} = [1, 1]^T$, $\phi_0 = [4, -1]^T$, $\phi_1 = [1, 1]^T$, $\phi_2 = [-1, -2]^T$, $\phi_3 = [0, 1]^T$, $\phi_4 = [1, 0]^T$, $\phi_5 = [-1, 0]^T$, $\phi_6 = [0, -1]^T$, $\mathbf{x}_t = [x_{1,t}, x_{2,t}]^T$, 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 $\mathbf{I}_2$. 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 Figure 2.8. Figure 2.8a shows the performances of the algorithms in the overfitting scenario, where the desired data is generated by the first order piecewise linear model in (2.28). Similarly, Figure 2.8b shows the performances of the algorithms in the underfitting scenario, where the desired data is generated by the third order piecewise linear model in (2.29). 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 .

2.5.5 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 [56].

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

$$d_t = 1 - \zeta d_{t-1}^2 + \eta d_{t-2}, \quad (2.30)$$

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 $\mathbf{x}_t = [d_{t-1}, d_{t-2}, 1]^T$, 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.

Figure 2.9 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

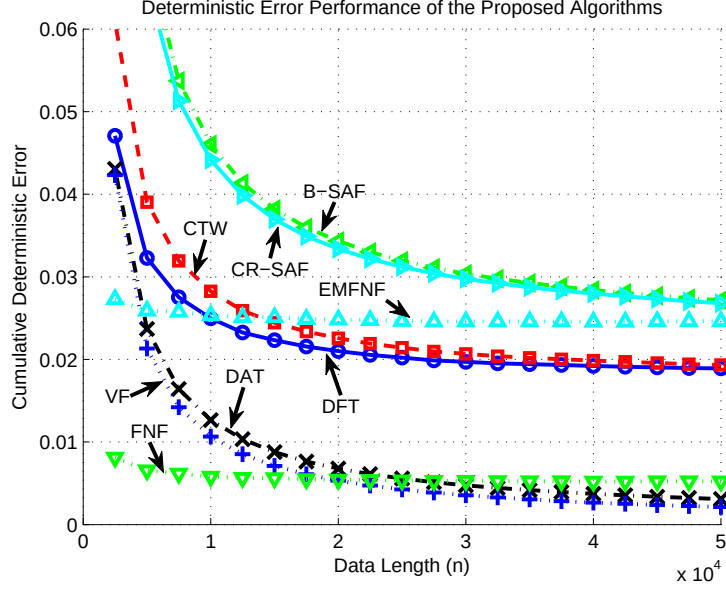


Figure 2.9: Regression error performances of the proposed algorithms for the chaotic process presented in (2.30).

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

Second, we consider the chaotic signal set generated using the Lorenz attractor [56] that is defined by the following three discrete time equations:

$$x_t = x_{t-1} + (\sigma(y - x))dt \quad (2.31)$$

$$y_t = y_{t-1} + (x_{t-1}(\rho - z_{t-1}) - y_{t-1})dt \quad (2.32)$$

$$z_t = z_{t-1} + (x_{t-1}y_{t-1} - \beta z_{t-1})dt, \quad (2.33)$$

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.

Figure 2.10 illustrates the nonlinear modeling power of the DAT algorithm even when estimating a highly nonlinear chaotic signal set. As can be observed

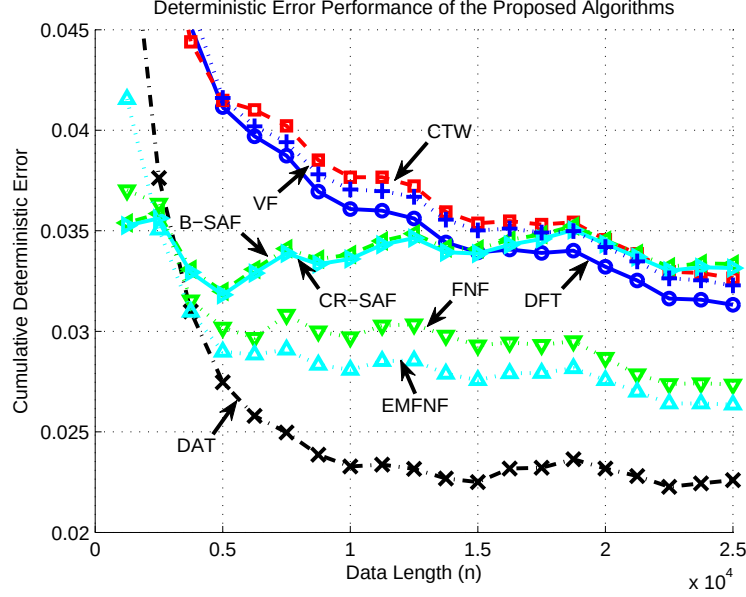


Figure 2.10: Regression error performances for the chaotic signal generated from the Lorenz attractor in (2.31), (2.32), and (2.33) with parameters $dt = 0.01$, $\rho = 28$, $\sigma = 10$, and $\beta = 8/3$.

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

Table 2.2: Time accumulated normalized errors of the proposed algorithms. Each dimension of the data sets is normalized between $[-1, 1]$.

from Figure 2.10, 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.

2.5.6 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 [53–55]: California

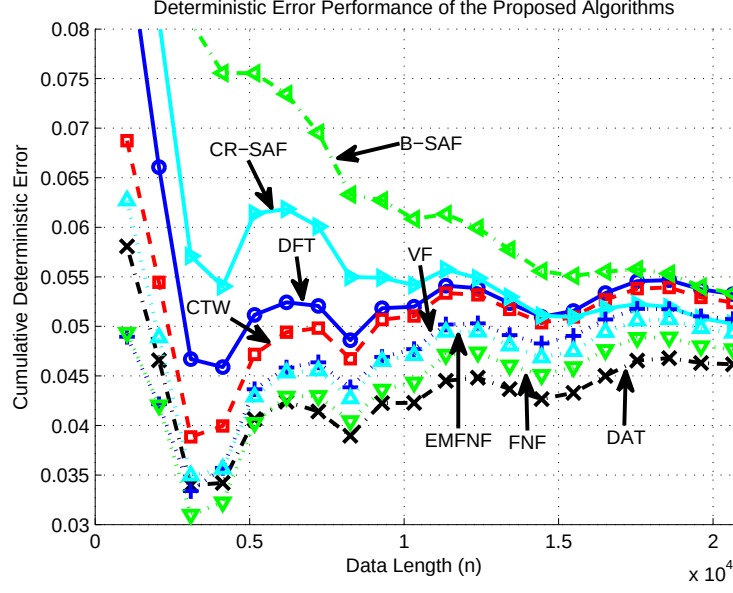


Figure 2.11: Regression error performances for the real data set: California housing - estimation of the median house prices in the California area using California housing database.

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. Figure 2.11 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.

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 [53] ($m = 8$) - 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 [54] ($m = 18$) - 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 [54] ($m = 32$) - 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.
- Bank [55] ($m = 32$) - 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 2.2, 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.

Algorithm 2 Decision Adaptive Tree (DAT) Regressor

```

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

```

Chapter 3

Predicting Nearly As Well As the Optimal Twice Differentiable Regressor

In this chapter, we study sequential nonlinear regression, where the aim is to estimate an unknown desired sequence $\{d[t]\}_{t \geq 1}$ by using a sequence of regressor vectors $\{\mathbf{x}[t]\}_{t \geq 1}$, where the desired sequence and the regressor vectors are real valued and bounded but otherwise arbitrary, i.e., $d[t] \in \mathbb{R}$, $\mathbf{x}[t] \triangleq [x_1[t], \dots, x_p[t]]^T \in \mathbb{R}^p$ for an arbitrary integer p and $|d[t]|, |x_i[t]| \leq A < \infty$ for all t and $i = 1, \dots, p$. We call the regressors “sequential” if in order to estimate the desired data at time t , i.e., $d[t]$, they only use the past information $d[1], \dots, d[t-1]$ and the observed regressor vectors¹ $\mathbf{x}[1], \dots, \mathbf{x}[t]$.

In this framework, a piecewise linear model is constructed by dividing the regressor space into a union of disjoint regions, where in each region a linear model holds. As an example, suppose that the regressor space is parsed into K disjoint regions $\mathcal{R}_1, \dots, \mathcal{R}_K$ such that $\bigcup_{k=1}^K \mathcal{R}_k = [-A, A]^p$. Given such a model,

¹All vectors are column vectors and denoted by boldface lower case letters. Matrices are denoted by boldface upper case letters. For a vector $\mathbf{x}$, $\mathbf{x}^T$ is the ordinary transpose. We denote $d_a^b \triangleq \{d[t]\}_{t=a}^b$.

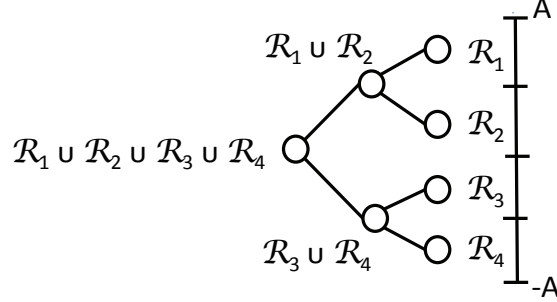


Figure 3.1: The partitioning of a one dimensional regressor space, i.e., $[-A, A]$, using a depth-2 full decision tree, where each node represents a portion of the regressor space.

at each time t , the sequential linear² regressor predicts $d[t]$ as $\hat{d}[t] = \mathbf{v}_k^T[t]\mathbf{x}[t]$ when $\mathbf{x}[t] \in \mathcal{R}_k$, where $\mathbf{v}_k[t] \in \mathbb{R}^p$ for all $k = 1, \dots, K$. These linear models assigned to each region can be trained independently using different adaptive methods such as the gradient descent or the recursive least squares (RLS) algorithms.

However, by directly partitioning the regressor space as $\bigcup_{k=1}^K \mathcal{R}_k = [-A, A]^p$ before the processing starts and optimizing only the internal parameters of the piecewise linear model, i.e., $\mathbf{v}_k[t]$, one significantly limits the performance of the overall regressor since we do not have any prior knowledge on the underlying desired signal. Therefore, instead of committing to a single piecewise linear model with a fixed and given partitioning, and performing optimization only over the internal linear regression parameters of this regressor, one can use a decision tree to partition the regressor space and try to achieve the performance of the best partitioning over the whole doubly exponential number of different models represented by this tree ([57]).

As an example, in Figure 3.1, we partition the one dimensional regressor space $[-A, A]$, using a depth-2 tree, where the regions $\mathcal{R}_1, \dots, \mathcal{R}_4$ correspond to disjoint intervals on the real line and the internal nodes are constructed using union of these regions. In the generic case, for a depth- d full decision tree, there exist 2^d leaf nodes and $2^d - 1$ internal nodes. Each node of the tree represents a

²Note that affine models can also be represented as linear models by appending a 1 to $\mathbf{x}[t]$, where the dimension of the regressor space increases by one.

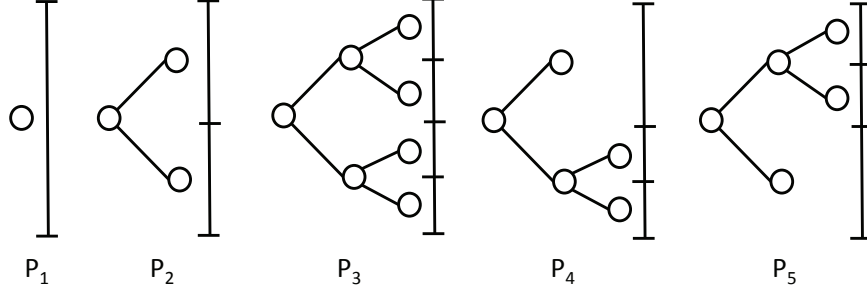


Figure 3.2: All different piecewise linear models that can be obtained using a depth-2 full decision tree, where the regressor space is one dimensional.

portion of the regressor space such that the union of the regions represented by the leaf nodes is equal to the entire regressor space $[-A, A]^p$. Moreover, the region corresponding to each internal node is constructed by the union of the regions of its children. In this way, we obtain $2^{d+1} - 1$ different nodes (regions) on the depth- d decision tree (on the regressor space) and approximately 1.5^{2^d} different piecewise models that can be represented by certain collections of the regions represented by the nodes of the decision tree ([57]). For example, we consider the same scenario as in Figure 3.1, where we partition the one dimensional real space using a depth-2 tree. Then, as shown in Figure 3.1, there are 7 different nodes on the depth-2 decision tree; and as shown in Figure 3.2, a depth-2 tree defines 5 different piecewise partitions or models, where each of these models is constructed using certain unions of the nodes of the full depth decision tree.

We emphasize that given a decision tree of depth- d , the nonlinear modeling power of this tree is fixed and finite since there are only $2^{d+1} - 1$ different regions (one for each node) and approximately 1.5^{2^d} different piecewise models (i.e., partitions) defined on this tree. Instead of introducing such a limitation, we recursively increment the depth of the decision tree as the data length increases. We call such a tree the “incremental decision tree” since the depth of the decision tree is incremented (and potentially goes to infinity) as the data length n increases, hence in a certain sense, we can achieve the modeling power of an infinite depth tree.

Using this incremental structure, we seek to construct a sequential regression algorithm (whose estimate at time t is represented by $\hat{d}_s[t]$), when applied to any sequence of data and regressor vectors, yields the following performance (i.e., regret) guarantee

$$\sum_{t=1}^n \left(d[t] - \hat{d}_s[t] \right)^2 - \inf_{f \in \mathcal{F}} \sum_{t=1}^n \left(d[t] - \hat{d}_f[t] \right)^2 \leq o(n), \quad (3.1)$$

over any n , without the knowledge of n , where $\mathcal{F}$ represents the class of all twice differentiable functions, whose parameters are set in hindsight, i.e., after observing the entire data before processing starts, and $\hat{d}_f[t]$ represents the estimate of the twice differentiable function $f \in \mathcal{F}$ at time t . The relative accumulated error in (3.1) represents the performance difference of the introduced algorithm and the optimal batch twice differentiable regressor. Hence, an upper bound of $o(n)$ in (3.1) implies that the algorithm $\hat{d}_s[t]$ sequentially and asymptotically converges to the performance of the regressor $\hat{d}_f[t]$, for any $f \in \mathcal{F}$.

3.1 Nonlinear Regression via Incremental Decision Trees

In this section, we present our incremental decision tree structure and use it for piecewise linear regression. For the sake of clarity, we first introduce certain notations to efficiently describe our incremental decision tree structure. We next introduce an iterative regressor space partitioning rule and construct an incremental decision tree using the resulting partitions. We then assign separate linear regressors to each node on this incremental decision tree and then introduce a sequential algorithm that achieves the performance of the best piecewise model on this incremental decision tree in Section 3.3.

3.1.1 Notation

We introduce a labeling for the nodes of the tree following [48]. The root node is labeled with an empty binary string λ and assuming that a node has a label κ , where $\kappa = \nu_1 \dots \nu_l$ is a binary string of length l formed from letters $\nu_1, \dots, \nu_l$, we label its upper and lower children as $\kappa 1$ and $\kappa 0$, respectively. Here, we emphasize that a string can only take its letters from the binary alphabet, i.e., $\nu \in \{0, 1\}$, where 0 refers to the lower child, and 1 refers to the upper child of a node. According to this notation, we say that a string $\kappa' = \nu'_1 \dots \nu'_{l'}$ is a prefix to string $\kappa = \nu_1 \dots \nu_l$ if $l' \leq l$ and $\nu'_i = \nu_i$ for all $i = 1, \dots, l'$, where the empty string λ is a prefix to all strings. We let $l(\kappa)$ represent the length of the string κ and $\mathcal{P}(\kappa)$ represent the set of all prefixes to the string κ , i.e., $\mathcal{P}(\kappa) \triangleq \{\kappa_0, \dots, \kappa_l\}$, where $l(\kappa) = l$ is the length of the string κ , κ_i is the string with length $l(\kappa_i) = i$, and $\kappa_0 = \lambda$ is the empty string, such that the first i letters of the string κ forms the string κ_i for all $i = 0, \dots, l$.

We let $\mathcal{L}_t$ and $\mathcal{N}_t$ represent the set of all leaf nodes and the set of all nodes on the incremental decision tree at time t , respectively. For each leaf node on the incremental decision tree at each time t , i.e., $\forall \kappa \in \mathcal{L}_t$, we assign a specific index $\alpha_\kappa \in \{0, \dots, M-1\}$ representing the number of regressor vectors that has fallen into $\mathcal{R}_\kappa$. This parameter is used in our design of the incremental decision tree and its use will be clearer in the following section. For presentation purposes, we consider $M = 2$ throughout this chapter.

3.1.2 Incremental Decision Trees

Before the processing starts, i.e., at time $t = 0$, we begin with a single node, i.e., the root node λ , having index $\alpha_\lambda = 0$. Then, we recursively construct the decision tree according to the following principle. For every time instant $t > 0$, we find the leaf node of the tree $\kappa \in \mathcal{L}_t$ such that $\mathbf{x}[t] \in \mathcal{R}_\kappa$. For this node, if we have $\alpha_\kappa = 0$, we do not modify the tree but only increment this index by 1. On the other hand, if $\alpha_\kappa = 1$, then we generate two children nodes $\kappa 0, \kappa 1$ for

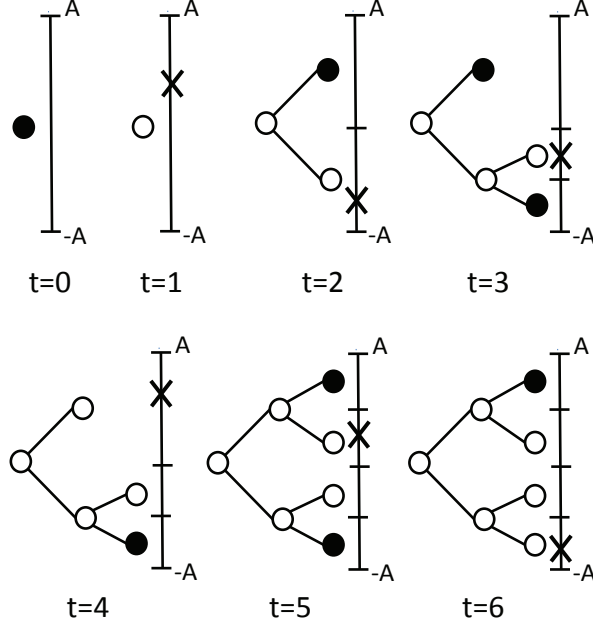


Figure 3.3: A sample evolution of the incremental decision tree, where the regressor space is one dimensional. The “ $\times$ ” marks on the regressor space represents the value of the regressor vector at that specific time instant. Light nodes are the ones having an index of 1, whereas the index of the dark nodes is 0.

this node by dividing the region $\mathcal{R}_\kappa$ into two disjoint regions $\mathcal{R}_{\kappa 0}, \mathcal{R}_{\kappa 1}$, using the plane $x_i = c$, where $i - 1 \equiv l(\kappa) \pmod{p}$ and c is the midpoint of the region $\mathcal{R}_\kappa$ along the i th dimension. For node $\kappa\nu$ with $\mathbf{x}[t] \in \mathcal{R}_{\kappa\nu}$ (i.e., the children node containing the current regressor vector), we set $\alpha_{\kappa\nu} = 1$ and the index of the other child is set to 0. The accumulated regressor vectors and the data in node κ are also transferred to its children to train a linear regressor in these child nodes.

As an example, in Figure 3.3, we consider that the regressor space is one dimensional, i.e., $[-A, A]$, and present a sample evolution of the tree. In the figure, the nodes having an index of 0 are shown as dark nodes, whereas the ones having an index of 1 are shown as light nodes, and the regressor vectors are marked with $\times$ ’s in the one dimensional regressor space. For instance at time $t = 2$, we have a depth-1 tree, where we have two nodes 0 and 1 with corresponding regions $\mathcal{R}_0 = [-A, 0]$, $\mathcal{R}_1 = [0, A]$, and $\alpha_0 = 1$, $\alpha_1 = 0$. Then, at time $t = 3$, we observe a regressor vector $\mathbf{x}[3] \in \mathcal{R}_0$ and divide this region into two disjoint regions using $x_1 = -A/2$ line. We then find that $\mathbf{x}[3] \in \mathcal{R}_{01}$, hence set $\alpha_{01} = 1$, whereas $\alpha_{00} = 0$.

We assign an independent linear regressor to each node on the incremental decision tree. Each linear regressor is trained using only the information contained in its corresponding node. Hence, we can obtain different piecewise models by using a certain collection of this node regressors according to the hierarchical structure. Using this incremental hierarchical structure with linear regressors at each region, the incremental decision tree can represent up to 1.5^n different piecewise linear models after observing a data of length n . For example, in Figure 3.3, at time $t = 6$, we have 5 different piecewise linear models (see Figure 3.2), whereas at time $t = 4$, we have 3 different piecewise linear models. Each of these piecewise linear models can be used to perform the estimation task. However, we use a mixture of experts approach to combine the outputs of all piecewise linear models, instead of choosing a single one among them.

To this end, one can assign a performance dependent weight to each piecewise linear model defined on the incremental decision tree and combine their weighted outputs to obtain the final estimate of the algorithm ([13]). In a conventional setting, such a mixture of expert approach is guaranteed to asymptotically achieve the performance of the best piecewise linear model defined on the tree ([30]). However, in our incremental decision tree framework, as t increases (i.e., as we observe new data), the total number of different piecewise linear models can increase exponentially with t . Thus, we have a highly dynamic optimization framework. For example, in Figure 3.3, at time $t = 4$, we have 3 different piecewise linear models, hence calculate the final output of our algorithm as $\hat{d}[t] = w_1[t]\hat{d}_1[t] + w_2[t]\hat{d}_2[t] + w_3[t]\hat{d}_3[t]$, where $\hat{d}_i[t]$ represents the output of the i th piecewise linear model and $w_i[t]$ represents its weight. However, at time $t = 6$, we have 5 different piecewise linear models, i.e., $\hat{d}[t] = \sum_{i=1}^5 w_i[t]\hat{d}_i[t]$, therefore the number of experts increases. Hence, not only such a combination approach requires the processing of the entire observed data at each time t (i.e., it results in a brute-force batch-to-online conversion), but also it cannot be practically implemented even for a considerably short data sequences such as $n = 100$.

To elegantly solve this problem, we assign a weight to each node on the incremental decision tree, instead of using a conventional mixture of experts approach.

Algorithm 3 Incremental Decision Tree (IDT)

- 1: Find the leaf node containing $\mathbf{x}[t]$, denote it by κ .
 - 2: **if** $\alpha_\kappa = 1$ **then**
 - 3: incrementTree(κ)
 - 4: Find the new leaf node containing $\mathbf{x}[t]$ on the incremented tree, denote it by κ .
 - 5: **end if**
 - 6: $\alpha_\kappa = 1$.
 - 7: $\mathcal{T}_{\kappa_i} = \mathcal{T}_{\kappa_i} + t, \forall \kappa_i \in \mathcal{P}(\kappa)$.
 - 8: predict($\mathbf{x}[t], \mathcal{P}(\kappa)$)
 - 9: update($d[t], \mathbf{x}[t], \mathcal{P}(\kappa)$)
-

In this way, we illustrate a method to calculate the original highly dynamic combination weights in an efficient manner, i.e., without requiring the processing of the entire data for each new sample, and with a significantly reduced computational complexity. The main structure of the proposed algorithm is provided in Algorithm 3. Before describing our algorithm in detail, we first provide the theoretical guarantees of our algorithm in the following section.

3.2 Main Results

In this section, we introduce the main results of this chapter. Particularly, we first show that the introduced sequential piecewise linear regression algorithm asymptotically achieves the performance of the best piecewise linear model defined on the incremental decision tree (with possibly infinite depth) with the optimal regression parameters at each region that minimizes the accumulated loss. We then use this result to prove that the introduced algorithm asymptotically achieves the performance of any twice differentiable regression function. We provide the algorithmic details and the construction of the algorithm in Section 3.3.

Theorem 3.1 *Let $\{d[t]\}_{t \geq 1}$ and $\{\mathbf{x}[t]\}_{t \geq 1}$ be arbitrary, bounded, and real-valued sequences of data and regressor vectors, respectively. Then Algorithm 3, whose*

prediction at time t is $\hat{d}[t]$, when applied to these data sequences yields

$$\sum_{t=1}^n \left(d[t] - \hat{d}[t] \right)^2 - \inf_{m \in \mathcal{M}'_n} \left[\inf_{\mathbf{v}^{(m)} \in \mathbb{R}^{pK_m}} \left\{ \sum_{t=1}^n \left(d[t] - \hat{d}_{batch}^{(m)}[t] \right)^2 + \delta \|\mathbf{v}^{(m)}\|^2 \right\} \right] \leq O(p \log^2(n)),$$

for any n , with a computational complexity upper bounded by $O(t)$ at each time instance t , where $\mathcal{M}'_n$ represents the set of all hierarchical models with at most $O(\log(n))$ leaves defined on the incremental decision tree at time n , $\hat{d}_{batch}^{(m)}[t]$ is the prediction of the m th model in the set $\mathcal{M}'_n$ whose parameter vectors at each node are chosen non-causally, K_m is the number of partitions in the m th model, i.e., $K_m \leq O(\log(n))$, $\forall m \in \mathcal{M}'_n$, and $\mathbf{v}^{(m)}$ is the vector constructed by concatenating the parameter vectors at each node on the m th model.

This theorem indicates that the introduced algorithm can asymptotically and sequentially achieve the performance of any piecewise model in the set $\mathcal{M}'_n$, i.e., the piecewise models having at most $O(\log(n))$ leaves defined on the tree. In particular, over any unknown data length n , the performance of the piecewise models with $O(\log(n))$ leaves can be sequentially achieved by the introduced algorithm with a regret upper bounded by $O(p \log^2(n))$. In this sense, we do not compare the performance of the introduced algorithm with a fixed class of regressors, over any data length n . Instead, the regret of the introduced algorithm is defined with respect to a set of piecewise linear regressors, whose number of partitions are upper bounded by $O(\log(n))$, i.e., the competition class grows as n increases. In the conventional tree based regression methods, the depth of the tree is set before processing starts and the performance of the regressor is highly sensitive with respect to the unknown data length. For example, if the depth of the tree is large whereas there are not enough data samples, then the piecewise model will be undertrained and yield an unsatisfactory performance. Similarly, if the depth of the tree is small whereas huge number of data samples are available, then trees (and regressors) with higher depths (and finer regions) can be better trained. As shown in Theorem 3.1, the introduced algorithm elegantly and intrinsically makes such decisions and performs asymptotically as well as any piecewise regressor in

the competition class that grows exponentially in n . Such a significant performance is achieved with a computational complexity upper bounded by $O(n)$, i.e., only linear in the data length, whereas the number of different piecewise models defined on the incremental decision tree can be in the order of 1.5^n ([57]). Moreover, under certain regularity conditions the computational complexity of the algorithm is $O(\log(n))$ as will be discussed in Remark 2. This theorem is an intermediate step to show that the introduced algorithm yields the desired performance guarantee in (3.1), and will be used to prove the next theorem.

Using Theorem 3.1, we introduce another theorem presenting the main result of the chapter, where we define the performance of the introduced algorithm with respect to the class of twice differentiable functions as in (3.1).

Theorem 3.2 *Let $\{d[t]\}_{t \geq 1}$ and $\{\mathbf{x}[t]\}_{t \geq 1}$ be arbitrary, bounded, and real-valued sequences of data and regressor vectors, respectively. Let $\mathcal{F}$ be the class of all twice differentiable functions such that for any $f \in \mathcal{F}$, $\frac{\partial^2 f(\mathbf{x})}{\partial x_i \partial x_j} \leq D < \infty$, $i, j = 1, \dots, p$ and we denote $\hat{d}_f[t] = f(\mathbf{x}[t])$. Then Algorithm 3, whose prediction at time t is $\hat{d}[t]$, when applied to these data sequences yields*

$$\sum_{t=1}^n \left(d[t] - \hat{d}[t] \right)^2 - \inf_{f \in \mathcal{F}} \sum_{t=1}^n \left(d[t] - \hat{d}_f[t] \right)^2 \leq o(p^2 n),$$

for any n , with a computational complexity upper bounded by $O(t)$ at each time instance t .

This theorem presents the nonlinear modeling power of the introduced algorithm. Specifically, it states that the introduced algorithm can asymptotically achieve the performance of the optimal twice differentiable function that is selected after observing the entire data in hindsight. Note that there are several kernel and neural network based sequential nonlinear regression algorithms ([18, 19]) (which can be modeled via twice differentiable functions) whose computational complexities are similar to the introduced algorithm. However, the performances of such nonlinear models are only comparable with respect to their batch variants. On the other hand, we demonstrate the performance of the introduced

algorithm with respect to a extremely large class of regressors without any statistical assumptions. In this sense, the performance of any regression algorithm that can be modeled by twice differentiable functions is asymptotically achievable by the introduced algorithm. Hence, the introduced algorithm yields a significantly more robust performance with respect to the such conventional approaches in the literature as also illustrated in different experiments in Section 3.4.

The proofs of Theorem 3.1, Theorem 3.2, and the construction of the algorithm are given in the following section.

3.3 Construction of the Algorithm and Proofs of the Theorems

In this section, we first introduce several lemmas before proving the theorems. In particular, we first introduce a weighting procedure over the incremental decision tree at time n (i.e., the final decision tree) and construct a regressor using this weighting. The resulting regressor is non-causal since the final decision tree needs to be known in advance of the processing. We then derive a regret upper bound on the performance of this non-causal regression algorithm. We next introduce a weighting procedure, whose values at time t are calculated using the incremental decision tree at time t . Using this new weights, we introduce a causal regression algorithm and show that it achieves the same performance as the aforementioned non-causal regressor. Following this procedure, we construct our algorithm and prove Theorem 3.1 and Theorem 3.2.

Let $\hat{d}_\kappa[t]$ denote the prediction of node κ at time t , where this predictor can be chosen arbitrarily. According to these prediction values, we assign a performance dependent weight to each node on the incremental decision tree at time n as

follows

$$P_\kappa(n) \triangleq \begin{cases} \exp \left\{ -\frac{1}{2a} \sum_{\mathbf{x}[t] \in \mathcal{R}_\kappa}^{t \leq n} (d[t] - \delta_\kappa[t])^2 \right\} & , \text{ if } \kappa \in \mathcal{L}_n \\ \frac{1}{2} P_{\kappa 0}(n) P_{\kappa 1}(n) + \frac{1}{2} \exp \left\{ -\frac{1}{2a} \sum_{\mathbf{x}[t] \in \mathcal{R}_\kappa}^{t \leq n} (d[t] - \delta_\kappa[t])^2 \right\} & , \text{ otherwise,} \end{cases} \quad (3.2)$$

where we set

$$\delta_\kappa[t] \triangleq \begin{cases} \hat{d}_{\kappa_t}[t] & , \text{ if } \kappa \notin \mathcal{L}_t, \\ \hat{d}_\kappa[t] & , \text{ otherwise,} \end{cases} \quad (3.3)$$

with $\kappa_t = \mathcal{L}_t \cap \mathcal{P}(\kappa)$ representing the closest descendant of κ that is available on the incremental tree at time t . In (3.3), for any node that is on the final decision tree but not on the incremental decision tree at time t , we set its prediction to be equal to the prediction of its closest prefix that is on the incremental decision tree at time t . In this sense, $\delta_\kappa[t]$ can be considered as a pseudo-predictor of the original predictor $\hat{d}_\kappa[t]$.

We use the performance dependent weights in (3.2) to obtain performance guarantees for the models defined on the incremental decision tree. To this end, letting $\mathcal{L}(m)$ represent the set of all partitions (i.e., nodes) in the m th model, we introduce the following lemmas.

Lemma 3.3 *According to the definition in (3.2), the weight of the root node λ can be constructed as follows*

$$P_\lambda(n) = \sum_{m \in \mathcal{M}_n} 2^{-B_m} \exp \left\{ -\frac{1}{2a} \sum_{t=1}^n (d[t] - \delta^{(m)}[t])^2 \right\}, \quad (3.4)$$

where $\delta^{(m)}[t] = \delta_\kappa[t]$ for $\kappa \in \mathcal{L}(m)$ such that $\mathbf{x}[t] \in \mathcal{R}_\kappa$, and B_m represents the number of bits required to represent the model m on the binary tree using a universal code (e.g., [58]).

The proof of Lemma 3.3 is provided in Appendix A.1. Using Lemma 3.3, we introduce the following lemma, by which we relate the performance of the original regressors to the weighting function in (3.2).

Lemma 3.4 According to the definitions in (3.2) and (3.3), we have

$$-2a \ln(P_\lambda(n)) \leq \min_{m \in \mathcal{M}'_n} \left\{ \sum_{t=1}^n \left(d[t] - \hat{d}^{(m)}[t] \right)^2 \right\} + (2a \ln(2) + 4A^2) O(\log(n)). \quad (3.5)$$

Proof of Lemma 3.4: From Lemma 3.3, we have

$$P_\lambda(n) \geq 2^{-B_m} \exp \left\{ -\frac{1}{2a} \sum_{t=1}^n \left(d[t] - \delta^{(m)}[t] \right)^2 \right\},$$

$\forall m \in \mathcal{M}_n$. Taking the logarithm of both sides, we obtain

$$-2a \ln(P_\lambda(n)) \leq \min_{m \in \mathcal{M}_n} \left\{ \sum_{t=1}^n \left(d[t] - \delta^{(m)}[t] \right)^2 \right\} + 2a \ln(2) B_m. \quad (3.6)$$

We then relate the loss of the perturbed predictor $\delta^{(m)}[t]$ in (3.6) to the loss of the original predictor $\hat{d}^{(m)}[t]$ as follows

$$\begin{aligned} \sum_{t=1}^n \left(d[t] - \delta^{(m)}[t] \right)^2 &= \sum_{\kappa \in \mathcal{L}(m)} \sum_{\substack{t \leq n \\ \mathbf{x}[t] \in \mathcal{R}_\kappa}} \left(d[t] - \delta_\kappa[t] \right)^2 \\ &= \sum_{\kappa \in \mathcal{L}(m)} \left\{ \sum_{\substack{t \leq n \\ \mathbf{x}[t] \in \mathcal{R}_\kappa \wedge \kappa \in \mathcal{N}_t}} \left(d[t] - \hat{d}_\kappa[t] \right)^2 + \sum_{\substack{t \leq n \\ \mathbf{x}[t] \in \mathcal{R}_\kappa \wedge \kappa \notin \mathcal{N}_t}} \left(d[t] - \hat{d}_{\kappa_t}[t] \right)^2 \right\} \\ &= \sum_{\kappa \in \mathcal{L}(m)} \left\{ \sum_{\substack{t \leq n \\ \mathbf{x}[t] \in \mathcal{R}_\kappa}} \left(d[t] - \hat{d}_\kappa[t] \right)^2 - \sum_{\substack{t \leq n \\ \mathbf{x}[t] \in \mathcal{R}_\kappa \wedge \kappa \notin \mathcal{N}_t}} \left(d[t] - \hat{d}_\kappa[t] \right)^2 \right. \\ &\quad \left. + \sum_{\substack{t \leq n \\ \mathbf{x}[t] \in \mathcal{R}_\kappa \wedge \kappa \notin \mathcal{N}_t}} \left(d[t] - \hat{d}_{\kappa_t}[t] \right)^2 \right\} \\ &\leq \sum_{\kappa \in \mathcal{L}(m)} \left\{ \sum_{\substack{t \leq n \\ \mathbf{x}[t] \in \mathcal{R}_\kappa}} \left(d[t] - \hat{d}_\kappa[t] \right)^2 + \sum_{\substack{t \leq n \\ \mathbf{x}[t] \in \mathcal{R}_\kappa \wedge \kappa \notin \mathcal{N}_t}} \left(d[t] - \hat{d}_{\kappa_t}[t] \right)^2 \right\} \\ &= \sum_{t=1}^n \left(d[t] - \hat{d}^{(m)}[t] \right)^2 + \sum_{\kappa \in \mathcal{L}(m)} \sum_{\substack{t \leq n \\ \mathbf{x}[t] \in \mathcal{R}_\kappa \wedge \kappa \notin \mathcal{N}_t}} \left(d[t] - \hat{d}_{\kappa_t}[t] \right)^2 \\ &\leq \sum_{t=1}^n \left(d[t] - \hat{d}^{(m)}[t] \right)^2 + \sum_{\kappa \in \mathcal{L}(m)} \sum_{\substack{t \leq n \\ \mathbf{x}[t] \in \mathcal{R}_\kappa \wedge \kappa \notin \mathcal{N}_t}} 4A^2, \end{aligned} \quad (3.7)$$

where the first line follows from the definition of the perturbed predictor in Lemma 3.3, the second line follows from (3.3), the fourth line follows when we omit the second sum in the third line, the fifth line follows from the definition of the predictor of the m th model, and the last line follows since $-A \leq d[t], \hat{d}_{\kappa_t}[t] \leq A$, $\forall t \in \{1, \dots, n\}$.

The double sum in the second term in (3.7) represents the number of time instances required to create the m th pruning. Here, we note that each node is partitioned into child nodes after it observes a regressor vector that has fallen into its corresponding region. This procedure is similar to the universal coding ([58]), where one bit for each inner node is used to represent a pruning. Therefore, we have

$$\sum_{\kappa \in \mathcal{L}(m)} \sum_{\substack{t \leq n \\ \mathbf{x}[t] \in \mathcal{R}_\kappa \wedge \kappa \notin \mathcal{N}_t}} 4A^2 \leq 4A^2 B_m, \quad (3.8)$$

where the inequality follows since a leaf node on a pruning can be an inner node on the incremental decision tree. Putting (3.7) and (3.8) back in (3.6) and noting that $B_m \leq O(\log(n))$, $\forall m \in \mathcal{M}'_n$, we obtain the desired inequality in (3.5). $\square$

Hence, as shown in Lemma 3.4, we have obtained a weighting assignment achieving the performance of the optimal piecewise linear model. We next introduce a low complexity sequential algorithm achieving $P_\lambda(n)$. To this end, we introduce the following lemma.

Lemma 3.5 *Assume that $\mathbf{x}[t] \in \mathcal{R}_\kappa$ for some $\kappa \in \mathcal{L}_n$. Then, we can write*

$$P_\lambda(t-1) = \sum_{\kappa_i \in \mathcal{P}(\kappa)} \pi_{\kappa_i}[t-1] \exp \left\{ -\frac{1}{2a} \sum_{\substack{t' \leq t \\ \mathbf{x}[t'] \in \mathcal{R}_{\kappa_i}}} (d[t'] - \delta_{\kappa_i}[t'])^2 \right\}, \quad (3.9)$$

where $\kappa_i \in \mathcal{P}(\kappa)$ is the string formed from the first i letters of $\kappa = \nu_1 \dots \nu_l$ and

$$\pi_{\kappa_i}[t] \triangleq \begin{cases} \frac{1}{2} & , \text{ if } i = 0 \\ \frac{1}{2} P_{\kappa_{i-1}\nu_i^c}(t-1) \pi_{\kappa_{i-1}}[t] & , \text{ if } 1 \leq i \leq l-1 \\ P_{\kappa_{i-1}\nu_i^c}(t-1) \pi_{\kappa_{i-1}}[t] & , \text{ if } i = l \end{cases} \quad (3.10)$$

The proof of Lemma 3.5 is provided in Appendix A.2. We use this lemma to construct a sequential algorithm achieving the regret bound in Lemma 3.4. To this end, we define the following predictor

$$\hat{d}[t] \triangleq \sum_{\kappa_i \in \mathcal{P}(\kappa)} \mu_{\kappa_i}[t-1] \delta_{\kappa_i}[t], \quad (3.11)$$

where

$$\mu_{\kappa_i}[t-1] \triangleq \frac{\pi_{\kappa_i}[t-1] \exp \left\{ -\frac{1}{2a} \sum_{\substack{t' < t \\ \mathbf{x}[t'] \in \mathcal{R}_{\kappa_i}}} (d[t'] - \delta_{\kappa_i}[t'])^2 \right\}}{P_\lambda(t-1)}. \quad (3.12)$$

In the next lemma, we relate the performance of this predictor in (3.11) to the weight of the root node. In this way, we relate the performance of the sequential predictor in (3.11) to the performance of the best piecewise model defined on the incremental decision tree using Lemma 3.4.

Lemma 3.6 *For any $a \geq 4A^2$, the sequential predictor in (3.11) achieves the following performance guarantee*

$$\sum_{t=1}^n \left(d[t] - \hat{d}[t] \right)^2 \leq -2a \ln(P_\lambda(n)). \quad (3.13)$$

Proof of Lemma 3.6: Without loss of generality, assume that at time t , we observe a regressor vector $\mathbf{x}[t] \in \mathcal{R}_\kappa$ for some $\kappa \in \mathcal{L}_n$, i.e., the regressor vector at time t has fallen into the node κ . Then, according to Lemma 3.5, we have

$$P_\lambda(t) = \sum_{\kappa_i \in \mathcal{P}(\kappa)} \pi_{\kappa_i}[t] \exp \left\{ -\frac{1}{2a} \sum_{\substack{t' \leq t \\ \mathbf{x}[t'] \in \mathcal{R}_{\kappa_i}}} (d[t'] - \delta_{\kappa_i}[t'])^2 \right\}. \quad (3.14)$$

Since $\mathbf{x}[t] \in \mathcal{R}_\kappa$ and $\pi_{\kappa_i}[t]$ is defined over the sibling nodes of the prefix nodes in $\mathcal{P}(\kappa)$, we then have $\pi_{\kappa_i}[t] = \pi_{\kappa_i}[t-1]$. Thus, we can write (3.14) as follows

$$\begin{aligned} P_\lambda(t) &= \sum_{\kappa_i \in \mathcal{P}(\kappa)} \pi_{\kappa_i}[t-1] \exp \left\{ -\frac{1}{2a} \sum_{\substack{t' \leq t \\ \mathbf{x}[t'] \in \mathcal{R}_{\kappa_i}}} (d[t'] - \delta_{\kappa_i}[t'])^2 \right\} \\ &= \sum_{\kappa_i \in \mathcal{P}(\kappa)} \pi_{\kappa_i}[t-1] \exp \left\{ -\frac{1}{2a} \sum_{\substack{t' < t \\ \mathbf{x}[t'] \in \mathcal{R}_{\kappa_i}}} (d[t'] - \delta_{\kappa_i}[t'])^2 \right\} \\ &\quad \times \exp \left\{ -\frac{1}{2a} (d[t] - \delta_{\kappa_i}[t])^2 \right\}, \end{aligned} \quad (3.15)$$

Then, dividing both sides of (3.15) by $P_\lambda(t-1)$, we obtain

$$\frac{P_\lambda(t)}{P_\lambda(t-1)} = \sum_{\kappa_i \in \mathcal{P}(\kappa)} \mu_{\kappa_i}[t-1] \exp \left\{ -\frac{1}{2a} (d[t] - \delta_{\kappa_i}[t])^2 \right\}, \quad (3.16)$$

according to the definition in (3.12).

We then focus on (3.16) and observe that $\sum_{\kappa_i \in \mathcal{P}(\kappa)} \mu_{\kappa_i}[t-1] = 1$ due to Lemma 3.5. We also observe that the second term in (3.16), i.e.,

$$f(\delta_{\kappa_i}[t]) \triangleq \exp \left\{ -\frac{1}{2a} (d[t] - \delta_{\kappa_i}[t])^2 \right\},$$

is concave since we have $|d[t]| \leq A$ and $|\delta_{\kappa_i}[t]| \leq A$, which indicates that $(d[t] - \delta_{\kappa_i}[t])^2 \leq 4A^2 \leq a$. Then, using Jensen's inequality, (3.16) can be written as follows

$$\exp \left\{ -\frac{1}{2a} \left(d[t] - \sum_{\kappa_i \in \mathcal{P}(\kappa)} \mu_{\kappa_i}[t-1] \delta_{\kappa_i}[t] \right)^2 \right\} \geq \frac{P_\lambda(t)}{P_\lambda(t-1)}. \quad (3.17)$$

We then note that we have

$$\begin{aligned} P_\lambda(n) &= \prod_{t=1}^n \frac{P_\lambda(t)}{P_\lambda(t-1)} \\ &\leq \prod_{t=1}^n \exp \left\{ -\frac{1}{2a} (d[t] - \hat{d}[t])^2 \right\} \\ &= \exp \left\{ -\frac{1}{2a} \sum_{t=1}^n (d[t] - \hat{d}[t])^2 \right\}, \end{aligned} \quad (3.18)$$

where the second line follows from (3.17). Taking natural logarithm of both sides and the multiplication with $-2a$ yields the desired result in (3.13). $\square$

Although in Lemma 3.6 we presented a performance guarantee to the sequential predictor in (3.11), this predictor still needs to know the final decision tree in advance since we have assumed that $\kappa \in \mathcal{L}_n$. In particular, the summation in (3.11) is over the final decision tree at time n , whereas we only have access to the nodes on the incremental decision tree at time t . To lift this assumption, we use the definition of the predictors $\delta_{\kappa_i}[t]$ given in (3.3). We first let $\kappa_r = \mathcal{P}(\kappa) \cap \mathcal{L}_t$

denote the leaf node (with depth r) on the incremental decision tree at time t that contains the current regressor vector. Then, we introduce the following weighting

$$\tilde{P}_\kappa(t) \triangleq \begin{cases} \exp \left\{ -\frac{1}{2a} \sum_{\mathbf{x}[t'] \in \mathcal{R}_\kappa}^{t' \leq t} (d[t'] - \delta_\kappa[t'])^2 \right\} & , \text{ if } \kappa \in \mathcal{L}_t \\ \frac{1}{2} \tilde{P}_{\kappa 0}(t) \tilde{P}_{\kappa 1}(t) + \frac{1}{2} \exp \left\{ -\frac{1}{2a} \sum_{\mathbf{x}[t'] \in \mathcal{R}_\kappa}^{t' \leq t} (d[t'] - \delta_\kappa[t'])^2 \right\} & , \text{ otherwise} \end{cases}, \quad (3.19)$$

$\forall \kappa \in \mathcal{N}_t$. Note that this weighting is over the incremental decision tree that is available at time t . Using this new weighting over the incremental decision tree, our aim is to design a sequential algorithm that achieves the performance of the predictor in (3.11), without the knowledge of the final incremental decision tree at time n . To this end, we first introduce the following lemma.

Lemma 3.7 *For all nodes that are on the final incremental decision tree at time n but not on the incremental decision tree at time t , i.e., $\forall \kappa \in \mathcal{L}_t \cup (\mathcal{N}_n - \mathcal{N}_t)$, we have*

$$P_\kappa(t) = \exp \left\{ -\frac{1}{2a} \sum_{\mathbf{x}[t'] \in \mathcal{R}_\kappa}^{t' \leq t} (d[t'] - \delta_\kappa[t'])^2 \right\}. \quad (3.20)$$

Proof of Lemma 3.7: We prove this claim by induction. If κ has the largest possible depth, i.e., $\kappa \in \mathcal{L}_n$, then the equality directly follows from the definition of $P_\kappa(t)$ in (3.2). Let us assume that the equality holds for nodes at an arbitrary

depth, say for nodes κ_0 and κ_1 . Then, we have the following recursion

$$P_\kappa(t) = \frac{1}{2}P_{\kappa_0}(t)P_{\kappa_1}(t) + \frac{1}{2}\exp\left\{-\frac{1}{2a}\sum_{\substack{t' \leq t \\ \mathbf{x}[t'] \in \mathcal{R}_\kappa}}(d[t'] - \delta_\kappa[t'])^2\right\} \quad (3.21)$$

$$\begin{aligned} &= \frac{1}{2}\exp\left\{-\frac{1}{2a}\sum_{\substack{t' \leq t \\ \mathbf{x}[t'] \in \mathcal{R}_{\kappa_0}}}(d[t'] - \delta_{\kappa_0}[t'])^2\right\}\exp\left\{-\frac{1}{2a}\sum_{\substack{t' \leq t \\ \mathbf{x}[t'] \in \mathcal{R}_{\kappa_1}}}(d[t'] - \delta_{\kappa_1}[t'])^2\right\} \\ &\quad + \frac{1}{2}\exp\left\{-\frac{1}{2a}\sum_{\substack{t' \leq t \\ \mathbf{x}[t'] \in \mathcal{R}_\kappa}}(d[t'] - \delta_\kappa[t'])^2\right\} \end{aligned} \quad (3.22)$$

$$= \exp\left\{-\frac{1}{2a}\sum_{\substack{t' \leq t \\ \mathbf{x}[t'] \in \mathcal{R}_\kappa}}(d[t'] - \delta_\kappa[t'])^2\right\}, \quad (3.23)$$

where the second line follows from the induction hypotheses and the last line follows since $\delta_\kappa[t] = \delta_{\kappa_0}[t] = \delta_{\kappa_1}[t]$, $\forall t' \leq t$ and $\mathcal{R}_{\kappa'} = \mathcal{R}_{\kappa'_0} \cup \mathcal{R}_{\kappa'_1}$ as well as $\mathcal{R}_{\kappa'_0} \cap \mathcal{R}_{\kappa'_1} = \emptyset$. This concludes the proof of the claim. $\square$

Using this lemma, we introduce the following corollary, which illustrates that the weights $\tilde{P}_\kappa(t)$ are the same with the weights $P_\kappa(t)$ over the incremental decision tree at time t .

Corollary 3.8 *The weights in (3.2) and (3.19) satisfy $P_\kappa(t) = \tilde{P}_\kappa(t)$, $\forall \kappa \in \mathcal{N}_t$.*

Proof of Corollary 3.8: This corollary directly follows from the definitions in (3.2) and (3.19) as well as Lemma 3.7, hence is omitted. $\square$

Using this new weighting over the incremental decision tree at time t , our aim is to introduce a sequential algorithm over this incremental decision tree at time t . To this end, we then turn our attention back to (3.11) and observe that it can be written as follows

$$\hat{d}[t] = \sum_{\kappa_i \in \mathcal{P}(\kappa_r)} \tilde{\mu}_{\kappa_i}[t-1] \hat{d}_{\kappa_i}[t], \quad (3.24)$$

where κ_r is the root node of length r on the incremental decision tree at time t containing the regressor vector at that time instance, i.e., $\kappa_r \in \mathcal{L}_t$ such that

$\mathbf{x}[t] \in \mathcal{R}_{\kappa_r}$, and

$$\tilde{\mu}_{\kappa_i}[t] \triangleq \begin{cases} \mu_{\kappa_i}[t] & , \text{ if } i < r \\ \sum_{j=r}^l \mu_{\kappa_j}[t] & , \text{ if } i = r \end{cases}. \quad (3.25)$$

Here, we emphasize that the summation in (3.24) is over the incremental decision tree at time t , whereas $\tilde{\mu}_{\kappa_i}$'s are still defined using the parameters over the incremental decision tree at time n . In order to construct $\tilde{\mu}_{\kappa_i}$'s with the parameters over the incremental decision tree at time t , we introduce the following lemma.

Lemma 3.9 *Letting*

$$\tilde{\pi}_{\kappa_i}[t] \triangleq \begin{cases} \frac{1}{2} & , \text{ if } i = 0 \\ \frac{1}{2} \tilde{P}_{\kappa_{i-1}\nu_i^c}(t-1) \tilde{\pi}_{\kappa_{i-1}}[t] & , \text{ if } 1 \leq i \leq r-1, \\ \tilde{P}_{\kappa_{i-1}\nu_i^c}(t-1) \tilde{\pi}_{\kappa_{i-1}}[t] & , \text{ if } i = r \end{cases} \quad (3.26)$$

we obtain

$$\tilde{\mu}_{\kappa_i}[t] = \frac{\tilde{\pi}_{\kappa_i}[t-1] \exp \left\{ -\frac{1}{2a} \sum_{\substack{t' < t \\ \mathbf{x}[t'] \in \mathcal{R}_{\kappa_i}}} (d[t'] - \delta_{\kappa_i}[t'])^2 \right\}}{\tilde{P}_{\lambda}(t-1)}, \quad (3.27)$$

$\forall i \leq r$.

Proof of Lemma 3.9: According to Corollary 3.8, we have $P_{\kappa}(t) = \tilde{P}_{\kappa}(t)$, $\forall \kappa \in \mathcal{N}_t$. This indicates that $\pi_{\kappa_i}[t] = \tilde{\pi}_{\kappa_i}[t]$, $\forall \kappa \in \mathcal{N}_t$. Thus, the equality in (3.27) satisfied for all $i < r$ due to Corollary 3.8. On the other hand, for $i = r$, if we also have $r = l$, then the equality trivially follows again due to Corollary 3.8. For $i = r$ and $r < l$, we have

$$\begin{aligned} \tilde{\mu}_{\kappa_r}[t] &= \sum_{j=r}^l \mu_{\kappa_j}[t] \\ &= \frac{1}{P_{\lambda}(t-1)} \sum_{j=r}^l \pi_{\kappa_j}[t-1] \exp \left\{ -\frac{1}{2a} \sum_{\substack{t' < t \\ \mathbf{x}[t'] \in \mathcal{R}_{\kappa_j}}} (d[t'] - \hat{d}_{\kappa_r}[t'])^2 \right\} \\ &= \frac{1}{P_{\lambda}(t-1)} \sum_{j=r}^l P_{\kappa_j}(t-1) \pi_{\kappa_j}[t-1], \end{aligned} \quad (3.28)$$

Algorithm 4 $\text{predict}(\mathbf{x}[t], \mathcal{P}(\kappa))$

- 1: **for all** $\kappa_i \in \mathcal{P}(\kappa)$ **do**
 - 2: Use (3.26) to find π_{κ_i} .
 - 3: $\mu_{\kappa_i} = \pi_{\kappa_i} L_{\kappa_i} / P_\lambda$
 - 4: $\hat{d}_{\kappa_i} = \mathbf{w}_{\kappa_i}^T \mathbf{x}[t]$
 - 5: **end for**
 - 6: $\hat{d} = \sum_{\kappa_i \in \mathcal{P}(\kappa)} \mu_{\kappa_i} \hat{d}_{\kappa_i}$
-

where the last line follows due to Corollary 3.8 since $\kappa_j \in \mathcal{L}_t \cup (\mathcal{N}_n - \mathcal{N}_t)$, $\forall j \in \{r, \dots, l\}$. We then separate the last term of the sum, i.e., $j = l$, and write

$$\begin{aligned} \tilde{\mu}_{\kappa_r}[t] &= \frac{1}{P_\lambda(t-1)} \left[P_{\kappa_l}(t-1) P_{\kappa_{l-1}\nu_l^c}(t-1) \pi_{\kappa_{l-1}}[t] + \sum_{j=r}^{l-1} P_{\kappa_j}(t-1) \pi_{\kappa_j}[t-1] \right] \\ &= \frac{1}{P_\lambda(t-1)} \left[P_{\kappa_{l-1}}(t-1) \pi_{\kappa_{l-1}}[t] + \sum_{j=r}^{l-1} P_{\kappa_j}(t-1) \pi_{\kappa_j}[t-1] \right], \end{aligned}$$

where the last line follows from the above claim. By separating another term from the sum we obtain

$$\begin{aligned} \tilde{\mu}_{\kappa_r}[t] &= \frac{1}{P_\lambda(t-1)} \left[P_{\kappa_{l-1}}(t-1) P_{\kappa_{l-1}\nu_l^c}(t-1) \pi_{\kappa_{l-2}}[t] + \sum_{j=r}^{l-2} P_{\kappa_j}(t-1) \pi_{\kappa_j}[t-1] \right] \\ &= \frac{1}{P_\lambda(t-1)} \left[P_{\kappa_{l-2}}(t-1) \pi_{\kappa_{l-2}}[t] + \sum_{j=r}^{l-2} P_{\kappa_j}(t-1) \pi_{\kappa_j}[t-1] \right]. \end{aligned}$$

By continuing this procedure until $j = r$, we obtain

$$\tilde{\mu}_{\kappa_r}[t] = \frac{2P_{\kappa_r}(t-1) \pi_{\kappa_r}[t-1]}{P_\lambda(t-1)}. \quad (3.29)$$

Since $r < l$, we have $\pi_{\kappa_r}[t] = \frac{1}{2} P_{\kappa_{r-1}\nu_r^c}(t-1) \pi_{\kappa_{r-1}}[t]$. Then according to the definition in (3.26), we have $\pi_{\kappa_r}[t] = \frac{1}{2} \tilde{\pi}_{\kappa_r}[t]$. Therefore, using Lemma 3.7 and Corollary 3.8, the equality in (3.27) follows for $i = r$. This concludes the proof of the lemma. $\square$

This lemma illustrates that we can obtain both $\tilde{\mu}_{\kappa_i}[t-1]$ and $\hat{d}_{\kappa_i}[t]$, $\forall i \leq r$ using the incremental decision tree at time t to construct the predictor in (3.24). Thus, our algorithm does not require any knowledge on the final incremental decision tree at time n and a description of this prediction subroutine is provided

in Algorithm 4. Before proving Theorem 3.1, we finally introduce the following universal piecewise linear regressor. We let

$$\hat{d}^{(m)}[t] = \mathbf{v}_\kappa^T[t] \mathbf{x}[t], \quad (3.30)$$

where

$$\mathbf{v}_\kappa[t] = (\mathbf{R}_\kappa[t] + \delta \mathbf{I})^{-1} \mathbf{p}_\kappa[t],$$

with $\mathbf{I}$ representing the appropriate sized identity matrix,

$$\mathbf{R}_\kappa[t] \triangleq \sum_{\substack{t' \leq t \\ \mathbf{x}[t'] \in \mathcal{R}_\kappa}} \mathbf{x}[t'] \mathbf{x}^T[t'],$$

and

$$\mathbf{p}_\kappa[t] \triangleq \sum_{\substack{t' \leq t \\ \mathbf{x}[t'] \in \mathcal{R}_\kappa}} d[t'] \mathbf{x}[t'].$$

A regret bound on the performance of this universal piecewise linear regressor is given in the following lemma.

Lemma 3.10 *For any $m \in \mathcal{M}'_n$ having $K_m = |\mathcal{L}(m)|$ disjoint regions, the piecewise linear regressor in (3.30) achieves the following performance guarantee*

$$\begin{aligned} \sum_{t=1}^n \left(d[t] - \hat{d}^{(m)}[t] \right)^2 - \min_{\mathbf{v}^{(m)} \in \mathbb{R}^{pK_m}} \left\{ \sum_{t=1}^n \left(d[t] - \hat{d}_{batch}^{(m)}[t] \right)^2 + \delta \|\mathbf{v}^{(m)}\|^2 \right\} \\ \leq A^2 K_m p \ln(n/K_m) + O(1). \end{aligned} \quad (3.31)$$

where $\hat{d}_{batch}^{(m)}[t] = \mathbf{v}_\kappa^T \mathbf{x}[t]$ such that $\kappa \in \mathcal{L}(m)$ with $\mathbf{x}[t] \in \mathcal{R}_\kappa$, and $\mathbf{v}^{(m)}$ is the vector constructed by concatenating the parameter vectors at each node on the m th model (i.e., letting $\mathcal{L}(m) = \{\kappa^{(1)}, \dots, \kappa^{(K_m)}\}$, we have $\mathbf{v}^{(m)} = [\mathbf{v}_{\kappa^{(1)}}^T, \dots, \mathbf{v}_{\kappa^{(K_m)}}^T]^T$).

Proof of Lemma 3.10: The proof of this lemma can be found following similar lines to [26], hence is omitted. $\square$

We emphasize that in each region of a piecewise model, different learning algorithms (not necessarily the above universal piecewise linear regressor), e.g., different linear regressors or nonlinear ones, from the broad literature can be

Algorithm 5 incrementTree(κ)

```
1: for all  $z \in \mathcal{T}_\kappa$  do
2:    $\nu = \{0, 1 : \mathbf{x}[z] \in \mathcal{R}_{\kappa\nu}\}$ 
3:    $\mathcal{T}_{\kappa\nu} = \mathcal{T}_\kappa + z$ 
4:    $L_{\kappa\nu} = L_{\kappa\nu} \exp(-(d[z] - \mathbf{w}_{\kappa\nu}^T \mathbf{x}[z])^2 / 2a)$ 
5:    $P_{\kappa\nu} = L_{\kappa\nu}$ 
6:    $\mathbf{R}_{\kappa\nu} = \mathbf{R}_{\kappa\nu} + \mathbf{x}[z] \mathbf{x}[z]^T$ 
7:    $\mathbf{w}_{\kappa\nu} = \mathbf{w}_{\kappa\nu} + \mathbf{R}_{\kappa\nu} \setminus (\mathbf{x}[z](d[z] - \mathbf{w}_{\kappa\nu}^T \mathbf{x}[z]))$ 
8: end for
9: for all  $\kappa_i \in \mathcal{P}(\kappa)$  do
10:   $P_{\kappa_i} = (P_{\kappa_i 0} P_{\kappa_i 1} + L_{\kappa_i}) / 2$ 
11: end for
```

Algorithm 6 update($d[t], \mathbf{x}[t], \mathcal{P}(\kappa)$)

```
1: for all  $\kappa_i \in \mathcal{P}(\kappa)$  do
2:    $L_{\kappa_i} = L_{\kappa_i} \exp(-(d[t] - \hat{d}_{\kappa_i})^2 / (2a))$ 
3:    $P_{\kappa_i} = \begin{cases} L_{\kappa_i} & , \text{ if } \kappa_i = \kappa \\ (P_{\kappa_i 0} P_{\kappa_i 1} + L_{\kappa_i}) / 2 & , \text{ o.w.} \end{cases}$ 
4:    $R_{\kappa_i} = R_{\kappa_i} + \mathbf{x}[t] \mathbf{x}[t]^T$ 
5:    $w_{\kappa_i} = w_{\kappa_i} + R_{\kappa_i} \setminus (\mathbf{x}[t](d[t] - \hat{d}_{\kappa_i}))$ 
6: end for
```

used. Note that although the main contribution in this chapter is the hierarchical organization and efficient management of these piecewise models, we also discuss the implementation of the universal piecewise linear model of [26] into our framework for completeness in Algorithms 5 and 6.

Using these lemmas, we can now prove Theorem 3.1 as follows.

Proof of Theorem 3.1: We have introduced a sequential predictor over the incremental decision tree in Lemma 3.9. The performance of this predictor can be obtained by combining the results of Lemma 3.4 and Lemma 3.6 as follows

$$\sum_{t=1}^n \left(d[t] - \hat{d}[t] \right)^2 - \min_{m \in \mathcal{M}'_n} \left\{ \sum_{t=1}^n \left(d[t] - \hat{d}^{(m)}[t] \right)^2 \right\} \leq (2a \ln(2) + 4A^2) O(\log(n)). \quad (3.32)$$

Then, using the universal piecewise linear regressor in (3.30), according to Lemma

3.10, we can rewrite (3.32) as follows

$$\begin{aligned}
\sum_{t=1}^n \left(d[t] - \hat{d}[t] \right)^2 &= \inf_{m \in \mathcal{M}'_n} \left[\inf_{\mathbf{v}^{(m)} \in \mathbb{R}^{pK_m}} \left\{ \sum_{t=1}^n \left(d[t] - \hat{d}_{\text{batch}}^{(m)}[t] \right)^2 + \delta \|\mathbf{v}^{(m)}\|^2 \right\} \right] \\
&\leq A^2 K_m (p \ln(n/K_m) + 4 \log(n)) + 2a \ln(2) \log(n) + O(1) \\
&\leq O(p \log^2(n)),
\end{aligned}$$

where the last line follows since $K_m \leq O(\log(n))$, $\forall m \in \mathcal{M}'_n$. This proves the regret upper bound in Theorem 3.1. Before concluding the proof, we finally discuss the computational complexity of the introduced algorithm in detail.

At any time instance t , our algorithm may add child nodes to some leaf node of the incremental decision tree at time $t - 1$ according to the regressor vector $\mathbf{x}[t]$. Without loss of generality assume that $\kappa \in \mathcal{L}_{t-1}$ and $\mathbf{x}[t] \in \mathcal{R}_\kappa$ as well as $\alpha_\kappa = 1$. To create child nodes to κ , our algorithm processes the past regressor vectors observed by this node. Since each node is partitioned into child nodes as soon as its index is one, the number of past regressor vectors observed by this node is upper bounded by the depth of the node κ . Letting $l(\kappa)$ represent the depth of node κ , the child nodes $\kappa 0$ and $\kappa 1$ are created in $O(l(\kappa))$ time.³ After these nodes are created, our algorithm updates the parameters $\tilde{P}$ and $\tilde{\pi}$ for all prefixes to node κ , which also requires $O(l(\kappa))$ time.

After the tree is incremented, our algorithm generates its prediction as in (3.24). Clearly, it requires $O(l(\kappa))$ time to calculate the final estimate of our algorithm. After the prediction is performed and the true data is observed, our algorithm updates its internal parameters. Since these updates are only over the prefixes to node κ , again it requires $O(l(\kappa))$ time to perform all the updates.

Since the depth of a node on the incremental decision tree at time t cannot be larger than t , the computational complexity of the algorithm at each time instance t is upper bounded by $O(t)$. Therefore, although theoretically the resulting computational complexity of the algorithm is upper bounded by $O(n)$, in many

³Note that here we do not consider the computational complexity of the predictors at each node. As an example, for the universal piecewise linear predictor of [26], this computational complexity becomes $O(p^2 l(\kappa))$. Yet, for first-order gradient based algorithms, it is $O(p l(\kappa))$.

real life applications, the regressor vectors converge to stationary distributions ([13]). Hence, in such practical applications, the computational complexity of the algorithm can be upper bounded by $O(\log(n))$ as discussed in Remark 2. We emphasize that in order to achieve this logarithmic computational complexity, we do not even require any statistical assumptions, instead it is sufficient that the regressor vectors are evenly distributed (to a certain degree) in the regressor space. This concludes the proof of Theorem 3.1. $\square$

We next prove Theorem 3.2 using Theorem 3.1 as follows.

Proof of Theorem 3.2: In Theorem 3.1, we have proven that our algorithm achieves the following regret bound

$$\sum_{t=1}^n \left(d[t] - \hat{d}[t] \right)^2 - \inf_{\mathbf{v}^{(m)} \in \mathbb{R}^{pK_m}} \left\{ \sum_{t=1}^n \left(d[t] - \hat{d}_{\text{batch}}^{(m)}[t] \right)^2 + \delta \|\mathbf{v}^{(m)}\|^2 \right\} \leq O(p \log^2(n)), \quad (3.33)$$

$\forall m \in \mathcal{M}'_n$, where $\mathcal{M}'_n$ represents the set of all hierarchical models with at most $O(\log(n))$ leaves defined on the incremental decision tree at time n and $\mathbf{v}^{(m)}$ is the vector constructed by concatenating the parameter vectors at each node on the m th model. Then, letting $\mathbf{v}_i^{(m)}$ denote the i th entry of vector $\mathbf{v}^{(m)}$, for any finite parameter vector $\mathbf{v}^{(m)}$, we define $G \triangleq \max_i |\mathbf{v}_i^{(m)}|$. Here, we note that such a maximum exists since $\sum_{t=1}^n \left(d[t] - \hat{d}_{\text{batch}}^{(m)}[t] \right)^2$ is finite due to the boundedness assumption, and thus picking a finite parameter vector directly minimizes the regularized cost in (3.33), which conflicts with the infimum operator. Since $\mathbf{v}^{(m)} \in \mathbb{R}^{pK_m}$, we then have $\|\mathbf{v}^{(m)}\| \leq GpK_m$. Considering (3.33) and noting that $K_m \leq O(\log(n))$, $\forall m \in \mathcal{M}'_n$, when we put this regularization term in the right hand side of the inequality and upper bound this regularization term as $\delta \|\mathbf{v}_m\|^2 \leq \delta G^2 p^2 O(\log^2(n))$, we obtain

$$\begin{aligned} \sum_{t=1}^n \left(d[t] - \hat{d}[t] \right)^2 - \inf_{\mathbf{v}^{(m)} \in \mathbb{R}^{pK_m}} \sum_{t=1}^n \left(d[t] - \hat{d}_{\text{batch}}^{(m)}[t] \right)^2 \\ \leq O(p \log^2(n)) + \delta G^2 p^2 O(\log^2(n)) \\ \leq O(p^2 \log^2(n)). \end{aligned} \quad (3.34)$$

Thus, for any piecewise model with at most $O(\log(n))$ partitions, our algorithm

achieves the regret bound in (3.34). For an arbitrary pruning m , we then define a suboptimal regressor by applying Taylor's theorem to a twice differentiable function $f \in \mathcal{F}$ about the midpoint of each region. We denote the prediction of this suboptimal regressor at time t by $\hat{d}_s[t]$ (where we drop the notational dependency on m for simplicity) and note that this is a suboptimal predictor since each linear model in every partition of this predictor should pass from the midpoint of each region, whereas the optimal predictor corresponding to this partitioning scheme does not have such a constraint. Therefore, we have

$$\inf_{\mathbf{v}^{(m)} \in \mathbb{R}^{pK_m}} \sum_{t=1}^n \left(d[t] - \hat{d}_{\text{batch}}^{(m)}[t] \right)^2 \leq \sum_{t=1}^n \left(d[t] - \hat{d}_s[t] \right)^2, \quad (3.35)$$

$\forall m \in \mathcal{M}'_n$. Putting (3.35) back in (3.34), we obtain

$$\sum_{t=1}^n \left(d[t] - \hat{d}[t] \right)^2 - \sum_{t=1}^n \left(d[t] - \hat{d}_s[t] \right)^2 \leq O(p^2 \log^2(n)). \quad (3.36)$$

We then consider the performance of this suboptimal piecewise linear regressor and apply the mean value theorem with the Lagrange form of the remainder, which yields

$$\begin{aligned} \sum_{t=1}^n \left(d[t] - \hat{d}_s[t] \right)^2 - \sum_{t=1}^n \left(d[t] - \hat{d}_f[t] \right)^2 \\ \leq 2A \sum_{t=1}^n \left\{ \sum_{i=1}^p \sum_{j=1}^p \frac{\partial^2 f(\mathbf{x})}{\partial x_i \partial x_j} \Big|_{\mathbf{x}=\mathbf{b}} (x_i[t] - a_{\kappa,i})(x_j[t] - a_{\kappa,j}) \right\}, \end{aligned} \quad (3.37)$$

for some $m \in \mathcal{M}'_n$ and $\mathbf{b} \in \mathcal{R}_\kappa$, where $\mathbf{a}_\kappa \triangleq [a_{\kappa,1}, \dots, a_{\kappa,p}]^T$ is the midpoint of the region $\mathcal{R}_\kappa$. In order to maximize this upper bound, we consider the depth- $O(\log(\log(n)))$ complete tree, which has $O(\log(n))$ leaf nodes. Since each leaf node on this tree has depth $O(\log(\log(n)))$, the regions corresponding to these leaf nodes are constructed by partitioning the regressor space $O(\log(\log(n)))$ times from the midpoint of each dimension in an incremental manner. Since each dimension of this regressor space has a radius of $2A$, each dimension of the partitions corresponding to leaf nodes has a radius of

$$\frac{2A}{2^{O(\log(\log(n)))/p}} = \frac{2A}{2^{O(\log(\log^{1/p}(n)))}} = \frac{2A}{O(\log^{1/p}(n))}. \quad (3.38)$$

Thus, by putting (3.38) back in (3.37), we obtain

$$\begin{aligned}
& \sum_{t=1}^n \left(d[t] - \hat{d}_s[t] \right)^2 - \sum_{t=1}^n \left(d[t] - \hat{d}_f[t] \right)^2 \\
& \leq 2Anp^2 \left[\max_{\mathbf{x}} \frac{\partial^2 f(\mathbf{x})}{\partial x_i \partial x_j} \right] \left[\max_{\mathbf{x}} (x_i - a_{\kappa,i})(x_j - a_{\kappa,j}) \right], \\
& \leq 2Anp^2 D \left(\frac{2A}{O(\log^{1/p}(n))} \right)^2 \\
& \leq 8A^3 D \frac{p^2 n}{O(\log^{2/p}(n))} \\
& \leq o(p^2 n). \tag{3.39}
\end{aligned}$$

Here, we note that if the depth- $O(\log(\log(n)))$ complete tree does not exist on our incremental decision tree, then we achieve an upper bound that is tighter than the above bound. This follows since such a case indicates that the nodes with depths less than $O(\log(\log(n)))$ has observed a number of data samples less than $O(\log(\log(n)))$. Thus, their contribution to the upper bound in (3.39) is negligible.

To conclude our proof, we add (3.36) and (3.39) to obtain

$$\begin{aligned}
\sum_{t=1}^n \left(d[t] - \hat{d}[t] \right)^2 - \sum_{t=1}^n \left(d[t] - \hat{d}_f[t] \right)^2 & \leq O(p^2 \log^2(n)) + o(p^2 n) \\
& \leq o(p^2 n).
\end{aligned}$$

This concludes the proof of Theorem 3.2. $\square$

Remark 3.1 *Note that the algorithm in Algorithm 3 achieves the performance of the best piecewise linear model having $O(\log(n))$ partitions with a regret of $O(p \log^2(n))$. In the most generic case, i.e., for an arbitrary piecewise model m having $O(K_m)$ partitions, the introduced algorithm still achieves a regret of $O(pK_m \log(n/K_m))$. This indicates that for models having $O(n)$ partitions, the introduced algorithm achieves a regret of $O(pn)$, hence the performance of the piecewise model cannot be asymptotically achieved. However, we emphasize that no other algorithm can achieve a smaller regret than $O(pn)$ as shown by [4], i.e., the introduced algorithm is optimal in a strong minimax sense. Intuitively, this*

lower bound can be justified by considering the case, in which the regressor vector at time t falls into the t th region of the piecewise model.

Remark 3.2 *As mentioned in Remark 1 (and also can be observed in (3.31)), no algorithm can converge to the performance of the piecewise linear models having $O(n)$ disjoint regions. Therefore, we can limit the maximum depth of the tree by $O(\log(t))$ at each time t to achieve a low complexity implementation. With this limitation and according to the update rule of the tree, we can observe that while dividing a region into two disjoint regions, we may be forced to perform $O(t)$ computations due to the accumulated regressor vectors (since their number can be as large as t). However, since a regressor vector is processed by at most $O(\log(t))$ nodes for any t , the average computational complexity of the update rule of the tree remains to be upper bounded by $O(\log(n))$. Furthermore, the performance of this low complexity implementation will be asymptotically the same as the exact implementation provided that the regressor vectors are evenly distributed in the regressor space, i.e., they are not gathered around a considerably small neighborhood. This result follows when we multiply the tree construction regret in (3.8) by the total number of accumulated regressor vectors, whose order, according to the above condition, is upper bounded by $o(n/\log(n))$.*

Remark 3.3 *We emphasize that the node indexes, i.e., α_κ 's, determines when to create finer regions. According to the described procedure, if a node at depth l is partitioned into smaller regions, then its i th predecessor, i.e., $\kappa_i \in \mathcal{P}(\kappa)$, has observed at least $l - i$ different regressor vectors. Hence, a child node is created when coarser regions (i.e., predecessor nodes) are sufficiently trained. In this sense, we introduce new nodes to the tree according to the current status of the tree as well as the most recent data. We also point out that, we divide each region from its midpoint (see Figure 3.3) to maintain universality. However, this process can also be performed in a data dependent manner, e.g., one can partition each region using the hyperplane that is perpendicular to the line joining two regressor vectors in that region. If there are more than two accumulated regressor vectors, then more advanced methods such as support vectors and anomaly detectors can*

be used to define a separator hyperplane. All these methods can be straightforwardly incorporated into our framework to produce different algorithms depending on the regression task and their performance guarantees can be straightforwardly calculated according to the following corollary.

Corollary 3.11 *Let $\{d[t]\}_{t \geq 1}$ and $\{\mathbf{x}[t]\}_{t \geq 1}$ be arbitrary, bounded, and real-valued sequences of data and regressor vectors, respectively. Let $\mathcal{F}$ be the class of all twice differentiable functions such that for any $f \in \mathcal{F}$, $\frac{\partial^2 f(\mathbf{x})}{\partial x_i \partial x_j} \leq D < \infty$, $i, j = 1, \dots, p$ and we denote $\hat{d}_f[t] = f(\mathbf{x}[t])$. Consider an arbitrary partitioning rule satisfying $|x_i - y_i| \leq \frac{2A}{r(d)}$, $\forall i \in \{1, \dots, p\}$ and $\forall \mathbf{x}, \mathbf{y} \in \mathcal{R}_\kappa$, where κ is any depth- d node on the incremental decision tree. Then Algorithm 3 with this partitioning rule, whose prediction at time t is $\hat{d}[t]$, when applied to these data sequences yields*

$$\sum_{t=1}^n \left(d[t] - \hat{d}[t] \right)^2 - \inf_{f \in \mathcal{F}} \sum_{t=1}^n \left(d[t] - \hat{d}_f[t] \right)^2 \leq O \left(\frac{p^2 n}{r^2(\log(n))} \right),$$

for any n , with a computational complexity upper bounded by $O(t)$ at each time instance t .

Proof of Corollary 3.11: This corollary follows after following similar lines to the proof of Theorem 3.2 by replacing the denominator of (3.38) with $r(\log(n))$, hence is omitted. $\square$

Note that the inequality $|x_i - y_i| \leq \frac{2A}{r(d)}$ in Corollary 3.11 illustrates that the diameter of the regions of the nodes with depth- d is upper bounded by $\frac{2A}{r(d)}$. For any partitioning rule, such a function $r(d)$ always exists since we can always pick $r(d) = 1, \forall d$ (i.e., to partition a space cannot increase the diameter of the space).

3.4 Simulations

In this section, we investigate the performance of the introduced algorithm with respect to various methods under several benchmark scenarios. Throughout the

Algorithm	Computational Complexity	Space Complexity
IDT	$O(p^2 \log(n))$	$O(p^2 n)$
CTW	$O(p^2 d)$	$O(p^2 2^d)$
LR	$O(p^2)$	$O(p^2)$
VSR	$O(p^{2r})$	$O(p^{2r})$
MARS	$O(rbw^3)$	$O(rbw)$
FNR	$O((pr)^{2r})$	$O((pr)^{2r})$

Table 3.1: Comparison of the complexities of the proposed algorithms with the corresponding update rules. In the table, p 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. For the MARS algorithm (particularly, the fast MARS algorithm), b represents the number of basis functions and w represents the window length.

experiments, we denote the incremental decision tree algorithm of Algorithm 3 by “IDT”, the context tree weighting algorithm of [4] by “CTW”, the linear regressor by “LR”, the Volterra series regressor by “VSR” [5], the sliding window Multivariate Adaptive Regression Splines of [59, 60] by “MARS”, and the Fourier nonlinear regressor of [7] by “FNR”. The combination weights of the LR, VSR, and FNR are updated using the recursive least squares (RLS) algorithm ([13]). Unless otherwise stated, the CTW algorithm has depth 2, the VSR, FNR, and MARS algorithms are second order, and the MARS algorithm uses 21 knots with a window length of 500 that shifts in every 200 samples.

In Table 3.1, we provide the computational and space complexities of the proposed algorithms. We emphasize that although the computational complexity to create and run the incremental decision tree is $O(\log(n))$, the overall computational complexity of the algorithm is $O(p^2 \log(n))$ due to the universal linear regressors at each region. Particularly, since the universal linear regressor at each region has a computational complexity of $O(p^2)$, the overall computational complexity of $O(p^2 \log(n))$ follows. However, this universal linear regressor can be straightforwardly replaced with any linear (or nonlinear) regressor in the literature. For example, if we use the LMS algorithm to update the parameters of the linear regressor instead of using the universal algorithm for this update, the computational complexity of the overall structure becomes $O(p \log(n))$. Hence, although the computational complexity of the original IDT algorithm is $O(\log(n))$,

this computational complexity may increase according to the computational complexity of the node regressors.

In this section, we first illustrate the performances of the proposed algorithms for a synthetic piecewise linear model that do not match the modeling structure of any of the above algorithms. We then consider the prediction of chaotic signals (generated from Duffing and Tinkerbell maps) and well-known data sequences such as Mackey-Glass sequence and Chua’s circuit ([2]). Finally, we consider the prediction of real life examples that can be found in various benchmark data set repositories (e.g., [53, 61]).

3.4.1 Synthetic Data

In this subsection, we consider the scenario where the desired data is generated by the following piecewise linear model

$$d[t] = \begin{cases} x_1[t] + x_2[t] + n[t] & , \text{ if } ||\mathbf{x}[t]||^2 \in [0, 0.1] \cup [0.5, 1] \\ -x_1[t] - x_2[t] + n[t] & , \text{ otherwise} \end{cases}, \quad (3.40)$$

and $\mathbf{x}[t] = [x_1[t], x_2[t]]^T$ are sample functions of a jointly Gaussian process of mean $[0, 0]^T$ and covariance matrix $\mathbf{I}$, and $n[t]$ is a sample function from a zero mean white Gaussian process with variance 0.1. Note that the piecewise model in (3.40) has circular regions, which cannot be represented by hyperplanes or twice differentiable functions. Hence, the underlying relationship between the desired data and the regressor vectors cannot be exactly modeled using any of the proposed algorithms.

In Figure 3.4, we present the normalized accumulated squared errors of the proposed algorithms averaged over 10 trials. For this experiment, “CTW-2” and “CTW-6” show the performances of the CTW algorithm with depths 2 and 6, respectively. Since the performances of the LR and FNR algorithms are incomparable with the rest of the algorithms, they are not included in the figure for this experiment. Figure 3.4 illustrates that even for a highly nonlinear system

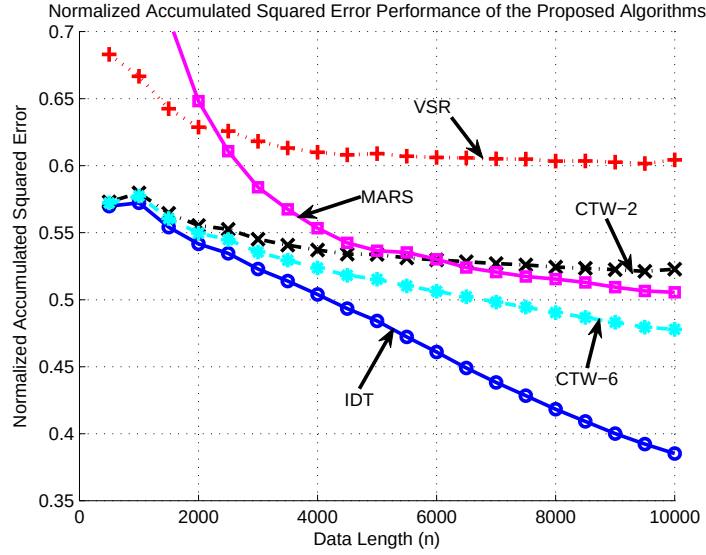


Figure 3.4: Normalized accumulated squared error performances for the piecewise linear model in (3.40) averaged over 10 trials.

(3.40), our algorithm significantly outperforms the other algorithms. The normalized accumulated error of the introduced algorithm goes to the variance of the noise signal as n increases, unlike the rest of the algorithms, whose performances converge to the performance of their optimal batch variants as n increases. This observation can be seen in Figure 3.4, where the normalized cumulative error of the IDT algorithm steadily decreases since the IDT algorithm creates finer regions as the observed data length increases. Hence, even for a highly nonlinear model such as the circular piecewise linear model in (3.40), which cannot be represented via hyperplanes, the IDT algorithm can well approximate this highly nonlinear relationship by incrementally introducing finer partitions as the observed data length increases.

Furthermore, even though the depth of the introduced algorithm is comparable with the CTW-6 algorithm over short data sequences, the performance of our algorithm is superior with respect to the CTW-6 algorithm. This results since the IDT algorithm intrinsically eliminates the extremely finer models at the early processing stages and introduces them whenever they are needed, unlike the CTW-6 algorithm. This procedure can be observed in Figure 3.5, where the IDT algorithm introduces finer regions (i.e., nodes with higher depths) to the

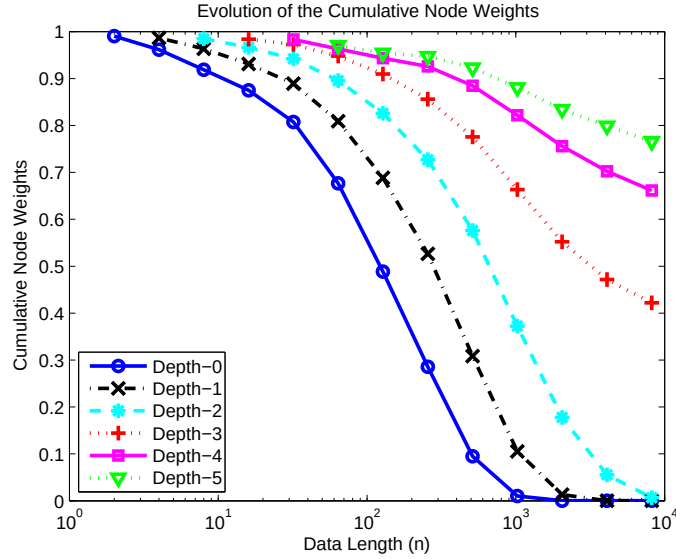


Figure 3.5: Evolution of the normalized cumulative node weights at the corresponding depths of the tree for the piecewise linear model in (3.40) averaged over 10 trials.

hierarchical model as the coarser regions becomes unsatisfactory. Since the universal algorithms such as CTW distribute a “budget” into numerous experts, as the number of experts increases, the performance of such algorithms deteriorate. On the other hand, the introduced algorithm intrinsically limits the number of experts according to the unknown data length at each iteration, hence we avoid such possible performance degradations as can be observed in Figure 3.5.

3.4.2 Chaotic Data

In this subsection, we consider prediction of the chaotic signals generated from the Duffing and Tinkerbell maps. The Duffing map is generated by the following discrete time equation

$$x[t + 1] = ax[t] - (x[t])^3 - bx[t - 1], \quad (3.41)$$

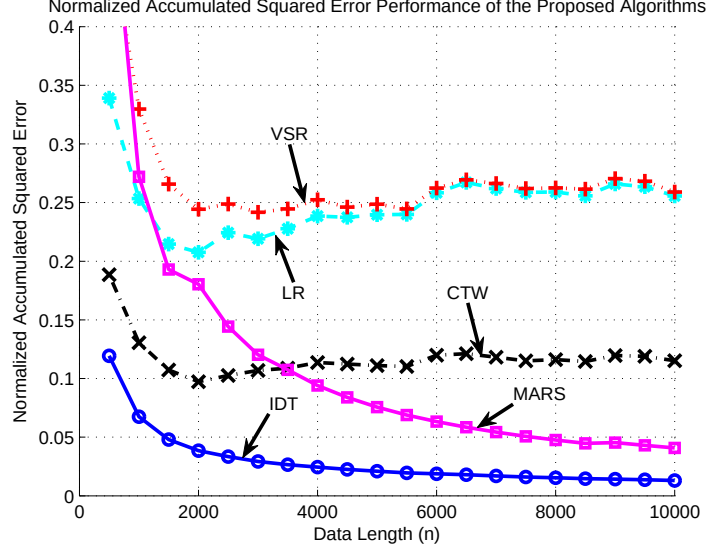


Figure 3.6: Normalized accumulated squared error performances for the chaotic data generated by the Duffing map in (3.41).

where we set $a = 2.75$ and $b = 0.2$ to produce the chaotic behavior ([62]). The Tinkerbell map is generated by the following discrete time equations

$$x[t + 1] = (x[t])^2 - (y[t])^2 + ax[t] + by[t] \quad (3.42)$$

$$y[t + 1] = 2x[t]y[t] + cx[t] + dy[t], \quad (3.43)$$

where we set $a = 0.9$, $b = -0.6013$, $c = 2$, and $d = 0.5$ ([4, 62]). We emphasize that these values are selected to generate the well-known chaotic behaviors of these attractors.

Figure 3.6 and Figure 3.7 shows the normalized accumulated squared error performances of the proposed algorithms. We emphasize that due to the chaotic nature of the signals, we observe non-uniform curves in Figure 3.6 and Figure 3.7. Since the conventional nonlinear and piecewise linear regression algorithms commit to a priori partitioning and/or basis functions, their performances are limited by the performances of the optimal batch regressors using these prior partitioning and/or basis functions as can be observed in Figure 3.6 and Figure 3.7. Hence, such prior selections result in fundamental performance limitations for these algorithms. For example, in the CTW algorithm, the partitioning of the regressor space is set before the processing starts. If this partitioning does not match with

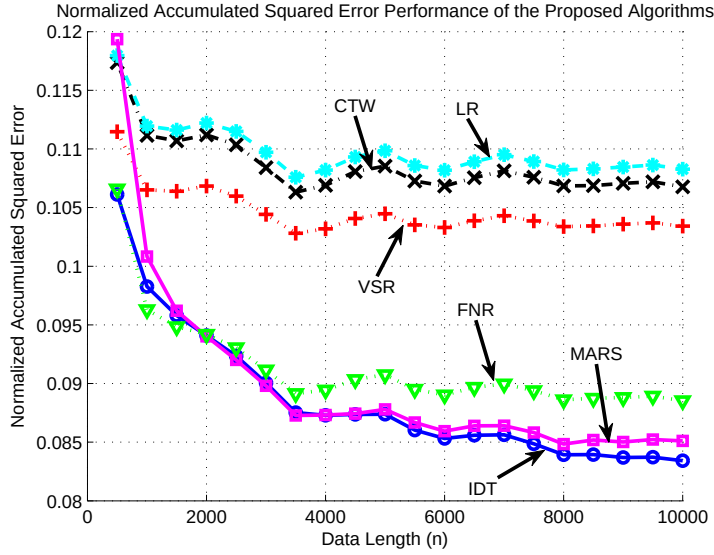


Figure 3.7: Normalized accumulated squared error performances for the chaotic data generated by the Tinkerbell map in (3.42) and (3.43).

the underlying partitioning of the regressor space, then the performance of the CTW algorithm becomes highly unsatisfactory as seen in Figure 3.6 and Figure 3.7. On the other hand, the introduced algorithm illustrates a robust and superior performance while learning the underlying chaotic relationships, whereas the rest of the algorithms yields an inconsistent performance due to the chaotic nature of the signals and the limited modeling power of these algorithms.

3.4.3 Benchmark Sequences

In this subsection, we consider the prediction of the Mackey-Glass and Chua's circuit sequences. The Mackey-Glass sequence is defined by the following differential equation

$$\frac{dx[t]}{dt} = \frac{\beta x[t - \tau]}{1 + (x[t - \tau])^n} - \gamma x[t], \quad (3.44)$$

where we set $\beta = 2$, $\gamma = 1$, $\tau = 2$, and $n = 10$ with the initial condition $x[t] = 0.5$ for $t < 0$. We generate the time series using the fourth order Runge-Kutta method. The Chua's circuit is generated according to the following differential

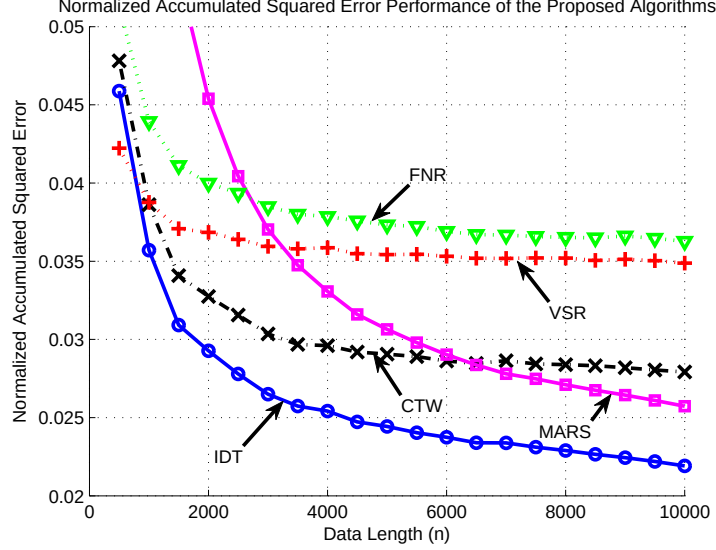


Figure 3.8: Normalized accumulated squared error performances for the Mackey-Glass sequence in (3.44).

equations

$$\frac{dx}{dt} = \alpha(y - x - f(x)), \quad \frac{dy}{dt} = x - y + z, \quad \frac{dz}{dt} = -\beta y, \quad (3.45)$$

where we dropped the time index for notational simplicity, and $f(x) = m_1x + 0.5(m_0 - m_1)(|x + 1| - |x - 1|)$, $\alpha = 15.6$, $\beta = 28$, $m_0 = -1.143$, $m_1 = -0.714$ with initial conditions $[x, y, z] = [0.7, 0, 0]$ for $t < 0$.

Figure 3.8 and Figure 3.9 shows the normalized accumulated squared error performances of the proposed algorithms for the Mackey-Glass and Chua's circuit sequences, respectively. We emphasize that due to the chaotic nature of the signals, we observe non-uniform curves in Figure 3.8 and Figure 3.9. In the figures, the algorithms with incomparable (i.e., unsatisfactory) performance are omitted. Figure 3.8 and Figure 3.9 presents that the IDT algorithm achieves an average of 20% relative gain in the performance with respect to the other algorithm and can accurately predict these well-known data sequences.

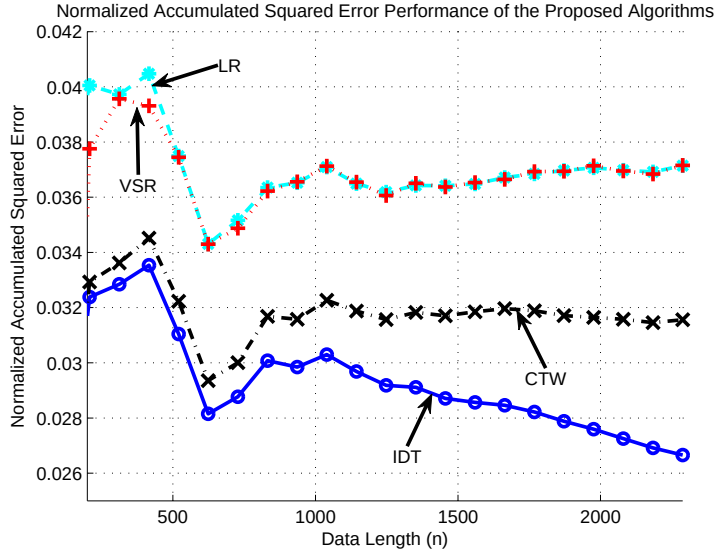


Figure 3.9: Normalized accumulated squared error performances for the Chua’s circuit sequence in (3.45).

3.4.4 Real Data

In this subsection, we evaluate the performance of the proposed algorithms for several well-known real data sets in machine learning literature.

We first compare the performances of the proposed algorithms using “kinematics” and “pumadyn” data sets that are taken from [53]. The kinematics data set involves a realistic simulation of the forward dynamics of an 8 link all-revolute robot arm and the task is to predict the distance of the end-effector from a target. Among its variants, we used the one having 9 attributes and being nonlinear as well as medium noisy. The pumadyn data set involves a realistic simulation of the dynamics of a Puma 560 robot arm. The task in these datasets is to predict the angular acceleration of one of the robot arm’s links. Among its variants, we used the one having 9 attributes and being nonlinear as well as medium noisy.

Figure 3.10 and Figure 3.11 shows the normalized accumulated squared error performances of the proposed algorithms for the kinematics and pumadyn data sets, respectively. In the experiments, all dimensions of the regressor vector and desired data are normalized between $[-1, 1]$. Although in Figure 3.10, the VSR

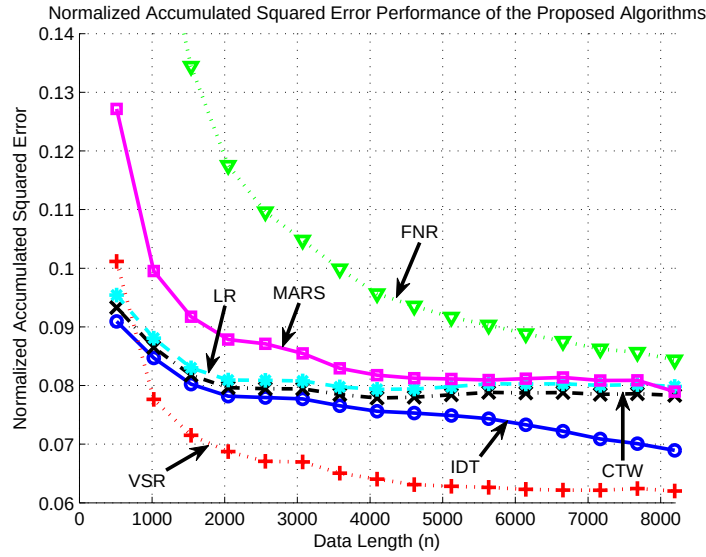


Figure 3.10: Normalized accumulated squared error performances for the “kinematics” data set.

algorithm provides the best performance and in Figure 3.11, the MARS algorithm achieves the minimum accumulated error, the performances of these algorithms in the reciprocal experiments are highly unsatisfactory. This result implies that the data in the first experiment can be well approximated by Volterra series, whereas the model that generates the data in the second experiment is more inline with B-splines. Hence, the performances of these algorithms are extremely sensitive to the underlying structure that generates the data. On the other hand, the IDT algorithm nearly achieves the performance of the best algorithm in both experiments and presents a desirable performance under different scenarios. This result implies that the introduced algorithm can be used in various frameworks without any significant performance degradations owing to its guaranteed performance upper bounds without any statistical or structural assumptions.

In Table 3.2, we present the performances of the proposed algorithms for various benchmark data sets that are widely used in machine learning literature. Below, we provide details of these data sets.

- *Dolphin* data set is taken from [63], where the data give the sound pressure of sonar signals (“clicks”) from a dolphin at various ranges to target. The

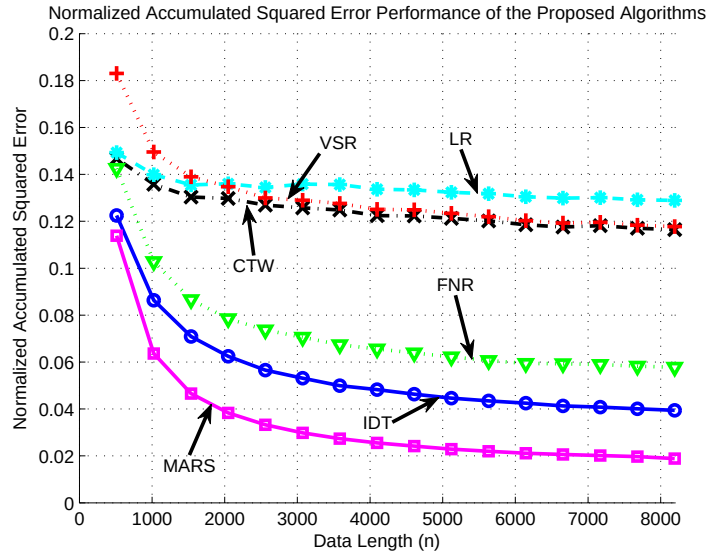


Figure 3.11: Normalized accumulated squared error performances for the “pumadyn” data set.

measurements were made off the coast of Iceland near Keflavik in 1998. The aim is to predict the sound pressure using the distance to the dolphin over 1634 data samples.

- *LMPAVW* data set is taken from [64], where the data set contains measurements of vertical shear (in units of 1/seconds) versus depths (in units of meters) in the ocean and has a length of 6875.
- *CCPP* data set is taken from [61], where the data is collected from a Combined Cycle Power Plant over 6 years (2006-2011). The aim is to predict the net hourly electrical energy output of the power plant using 4 attributes over 9568 data instances.
- *Protein Tertiary* data set is taken from [61], where the data is collected from Critical Assessment of protein Structure Prediction (CASP) experiments 5-9. The aim is to predict the size of the residue using 9 attributes over 45730 data instances.
- *Lorenz Attractor* is a mathematical model for atmospheric convection introduced by [56]. We use this model with parameters that generate its well-known chaotic solution over 5000 data samples.

	IDT	FNR	MARS	CTW	LR	VSR
Dolphin	0.0221	0.0290	0.0293	0.0235	0.0253	0.0223
LMPAVW	0.0314	0.0984	0.0934	0.1115	0.1153	0.1122
CCPP	0.0129	0.0138	0.0178	0.0131	0.0147	0.0129
Protein Tertiary	0.2195	0.2224	0.2468	0.2379	0.2458	0.2257
Lorenz Attractor	0.0336	0.0389	0.0396	0.0346	0.0404	0.0354
Hwang-SIF	0.0190	0.0952	0.0530	0.0919	0.3036	0.1617
Hwang-HF	0.1331	0.1991	0.1867	0.2388	0.3225	0.3243
Hwang-AF	0.1705	0.2578	0.2412	0.2688	0.3340	0.3007
Hwang-CIF	0.1699	0.2464	0.2325	0.2512	0.3334	0.2800

Table 3.2: Squared errors of the proposed algorithms for various benchmark data sets, where each dimension of the data sets are scaled between $[-1, 1]$.

- *Hwang* data set is taken from [53], where the data set gives the values of five test functions over a common two-dimensional domain. Test functions are chosen as Simple Interaction Function (SIF), Harmonic Function (HF), Additive Function (AF), and Complicated Interaction Function (CIF), for the corresponding data sets. This data set is first introduced by [65] and has a length of 13600.

As can be seen from Table 3.2, the proposed IDT algorithm significantly outperforms the competitor algorithms. In the first four rows of the table, the squared errors of the proposed algorithms can be found for the corresponding data sets that are collected from real life applications. In the “Dolphin”, “CCPP”, and “Protein Tertiary” data sets, the linear regressor can achieve a comparable performance with respect to the competitor nonlinear learning algorithms. This indicates that the underlying data sequences are highly noisy so that we cannot sufficiently train nonlinear models. On the other hand, for the “LMPAVW” data set, the performance of the linear regressor is significantly outperformed by the nonlinear regressors (e.g., the “FNR” and “MARS” algorithms). Thus, this data may be accurately modeled by some nonlinear function. As a consequence, the proposed IDT algorithm efficiently approximates this nonlinear function and presents a performance gain around 300 – 400% with respect to the competitor algorithms. These results indicate that the proposed IDT algorithm can be used in real life applications, especially in ones that include high levels of nonlinearity.

This conclusion can also be observed from the results of our experiments with the “Lorenz Attractor”, which is used to model atmospheric convection.

In the last four rows of Table 3.2, we present the performances of the algorithms for benchmark data sets that are synthetically generated using various test functions. We note that the competitor algorithms assume fixed basis functions and perform regression over the space spanned by these functions. Consequently, the performances of the competitor algorithms are highly dependent on the portion of the data energy that lies in this space. On the other hand, the proposed IDT algorithm can well approximate these unknown test functions using piecewise models, and hence achieves a significantly better squared error performance than the competitor algorithms.

Chapter 4

Sequential Prediction over Hierarchical Structures

In this chapter, we consider the generic prediction framework under an arbitrary Lipschitz continuous loss function [66], where we sequentially observe a real valued sequence $x_1, x_2, \dots$ and produce an output $\hat{d}_t$ based on $x_1, \dots, x_t$ at each time t . Then, the true output d_t is revealed yielding the loss $\ell(d_t, \hat{d}_t)$, e.g., $\ell(d_t, \hat{d}_t) = (d_t - \hat{d}_t)^2$. Over any data length n , the performance of the predictor is evaluated by its time accumulated loss, i.e., $\sum_{t=1}^n \ell(d_t, \hat{d}_t)$, or by its accumulated MSE, i.e., $\sum_{t=1}^n E[\ell(d_t, \hat{d}_t)]$. Nevertheless, we emphasize that our methods can incorporate different loss functions (as pointed out in Section 4.5) such as the accumulated weighted loss, i.e., $\sum_{t=1}^n \lambda^{n-t} \ell(d_t, \hat{d}_t)$, where $0 < \lambda \leq 1$ represents the forgetting factor.

4.1 Finite State Predictors

In order to produce the output $\hat{d}_t$, we use an FS predictor. In its most generic form, an FS predictor has a sequential prediction function

$$\hat{d}_t = f_t(s_t), \tag{4.1}$$

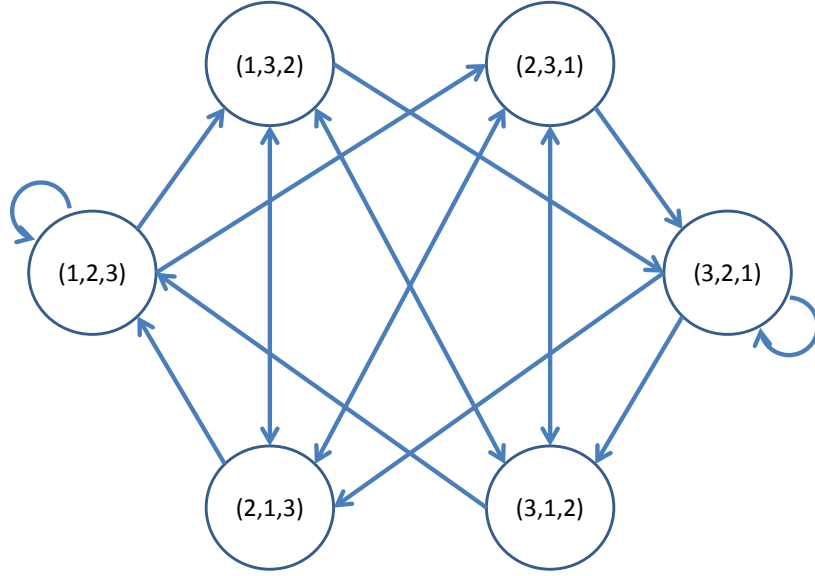


Figure 4.1: FS Diagram for $l = 3$ and $h = 2$, where all allowable transitions are drawn.

where $s_t \in \mathcal{S}$ is the current state taking values from a finite set $\mathcal{S}$, e.g., the set of relative ordering patterns. Upon the observation of the new data x_{t+1} , the states are traversed according to the next state function

$$s_{t+1} = g(s_t, x_{t+1}). \quad (4.2)$$

One can use different variations for (4.1) and (4.2), e.g., include different samples of the observed sequence in the function definitions or use time varying functions, e.g., $\hat{d}_t = f(s_t, s_{t-1})$ or $s_{t+1} = g_t(s_t, x_t, x_{t-1}, x_{t-2})$. However, either these variations can be covered by the basic setup by defining a super set of states or our results can be straightforwardly extended to these configurations [67].

In this framework, we consider a finite set of states $\mathcal{S}$ that is selected according to the underlying prediction task. As an example, for prediction problem, one can choose the past values of a sequence as the set of states. Similarly, for portfolio selection problem, the set of states can be chosen according to the ratio of the closing prices of stocks to the opening prices [68]. In our example, we consider the relative ordering patterns of the sequence history as our states. In particular, at each time t , we use the last l samples of the sequence history $x_{t-l+1}, \dots, x_t$ to define equivalence classes or states. For example, for $l = 3$, we can have $3! = 6$ different possible patterns, i.e., $\mathcal{S} = \{(1, 2, 3), (1, 3, 2), (2, 1, 3), (2, 3, 1), (3, 1, 2), (3, 2, 1)\}$,

where “3” represents the location of the largest value and “1” represents the location of the smallest value, e.g., the sequence $(x_{t-2}, x_{t-1}, x_t) = (5, 10, -1)$ corresponds to the pattern or ordering $(2, 3, 1)$. At time instant t , naturally, the current state s_t can correspond to only one of the states in $\mathcal{S}$, i.e., the state set is complete.

Thus, $\mathcal{S}$ forms the highest level of hierarchy in our framework and each elements of $\mathcal{S}$ denotes a different node in this level. Then, starting from the highest level, we tie certain states into certain equivalence classes and create lower levels in the hierarchy. As an example, for the order preserving patterns, we have the aforementioned states at the highest level (see Figure 4.2). Assume that our aim is to create a hierarchical structure of depth $h = l - 1$ according to the place of the greatest element in each equivalence class. Hence, we tie the 6 states at level-2 according to the place of their greatest element, e.g., we tie the states $(1, 2, 3)$ and $(2, 1, 3)$ into $(\cdot, \cdot, 3)$. Thus, we obtain the equivalence classes $(\cdot, \cdot, 3)$, $(\cdot, 3, \cdot)$, and $(3, \cdot, \cdot)$ at level-1. Continuing this procedure, we obtain $(\cdot, \cdot, \cdot)$ at level-0.

In the generic scenario, let $c_{i,j}$ represent the j th equivalence class at the i th hierarchy level and let $\mathcal{H}_i$ represent the set of equivalence classes at hierarchy level i , where $0 \leq i \leq h$. We start constructing our equivalence classes by setting $\mathcal{H}_h = \mathcal{S}$ and tie certain equivalence classes according to a tying function

$$\phi^{(i+1 \rightarrow i)}(c_{i+1,j}) = c_{i,k}, \quad (4.3)$$

for some $c_{i+1,j} \in \mathcal{H}_{i+1}$ and $c_{i,k} \in \mathcal{H}_i$, where $0 \leq i \leq h - 1$. With an abuse of notation, we use $\phi(c_{i,j+1})$ as our tying function in the rest of the chapter.

Having obtained a hierarchical structure using a tying function $\phi(\cdot)$, we assign an FS predictor to each of the equivalence classes (where we emphasize that each of the original states at the highest level also corresponds to an equivalence class). Continuing with our example, for ease of exposition, let us assume that our aim is to determine the relative gain/loss of the upcoming value of the sequence compared to its current value, i.e.,

$$d_t = \begin{cases} 1 & , \text{ if } x_{t+1} \geq x_t \\ -1 & , \text{ otherwise} \end{cases}. \quad (4.4)$$

Then, we can use a sequential binary prediction algorithm in each equivalence class on the hierarchical model. For this, we can assign a universal binary predictor (such as [69]) to each equivalence class $c_{i,j}$ as follows

$$\hat{d}_t^{\{c_{i,j}\}} = \frac{\sum_{z=1}^{t-1} I_z^{\{c_{i,j}\}} d_z}{\delta + \sum_{z=1}^{t-1} I_z^{\{c_{i,j}\}}} \quad (4.5)$$

to predict d_t , where $\delta > 0$ is a small constant, $\hat{d}_t^{\{c_{i,j}\}}$ is the prediction of the equivalence class $c_{i,j}$ at time t and $I_z^{\{c_{i,j}\}}$ is the indicator function representing whether the length- h sequence corresponds to equivalence class $c_{i,j}$, i.e.,

$$I_z^{\{c_{i,j}\}} \triangleq \begin{cases} 1 & , \text{ if } s_t \in c_{i,j} \\ 0 & , \text{ otherwise} \end{cases} \quad (4.6)$$

Having fixed the state definitions and the sequential predictors at each equivalence class, we then consider all different FS predictors defined on the hierarchical model. For example, one can construct an FS predictor by considering all nodes at the h th hierarchy level as follows: The output of the FS predictor is set as $\hat{d}_t = \hat{d}_t^{\{c_{h,j}\}}$, when $s_t \in c_{h,j}$, where $c_{h,j} \in \mathcal{H}_h$. Similarly, one can construct an FS predictor by considering the single equivalence class at the lowest level, i.e., $c_{0,1}$, and directly set $\hat{d}_t = \hat{d}_t^{\{c_{0,1}\}}$. In Figure 4.3, we illustrate 3 possible FS predictors for our example of order preserving patterns with $l = 3$ and $h = 2$.

We emphasize that FS predictors that are formed using equivalence classes at higher hierarchy levels includes more states. For instance, in Figure 4.3, the second FS predictor is formed using the equivalence classes at level-1 and includes 3 states, whereas the third FS predictor includes some equivalence classes from level-2, thus includes 5 states. Clearly, FS predictors containing more states require longer data sequences in order to sufficiently train each sequential equivalence class (or state) predictor $\hat{d}_t^{\{c_{i,j}\}}$ in such FS predictors. In this sense, a hierarchical structure introduces different coarse and fine FS predictors. Hence, at the beginning of the learning process, one can use coarser FS predictors that can be efficiently learned and then gradually switch to the original complete model defined on the highest level of hierarchy. However, a careless switching

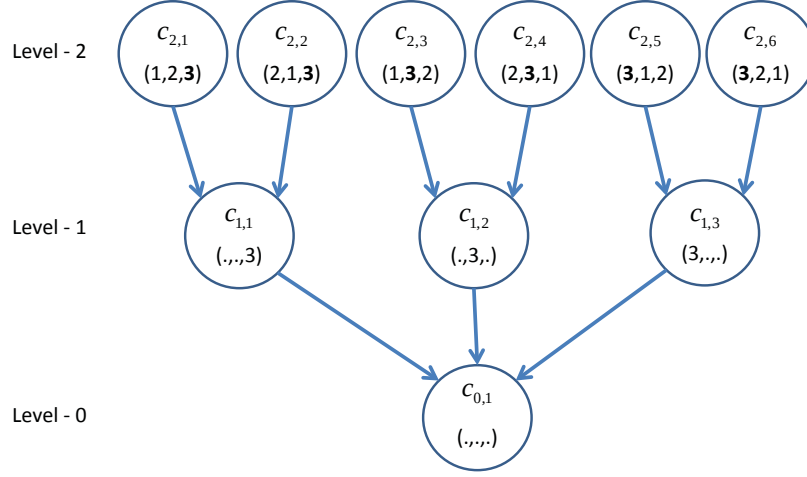


Figure 4.2: All equivalence classes for the FS diagram with $l = 3$ and $h = 2$.

between coarser and finer FS predictors can significantly deteriorate the performance of the system and the optimal selection of such a switching is highly data dependent [70]. Furthermore, the effectiveness of FS predictors may change over time, especially when the underlying data is highly nonstationary. Then, the complete model defined on the highest level of hierarchy may never have enough data to adequately train equivalence class predictors even if it observes a data of infinite length. To this end, in the following section, we introduce a sequential algorithm that elegantly and effectively performs such decisions by intrinsically implementing and combining a huge number of FS predictors.

For a hierarchical structure with depth h , we can have at least 2^{2^h} different FS predictors provided that each equivalence class at any level is formed by tying at least two equivalence classes in the above level [57]. As an example, for the order preserving patterns, we have $K_h \approx 2^{(h/e)^h}$ different FS predictors since $K_h = (K_{h-1})^{h+1} + 1$ as can be seen in Figure 4.2. In the generic scenario, over $K > 2^{2^h}$ different FS predictors, i.e., $\hat{d}_t^{\{k\}}$, $k = 1, \dots, K$ (where we drop the subscript h for notational simplicity), one of them is optimal for the current observations. However, as we observe new samples of the data, the optimal FS predictor can change over time. For example, when there is not enough data, the coarsest FS predictor only having the equivalence class at the lower hierarchy level, i.e., $c_{0,1}$, is expected to learn much faster than the finest FS predictor

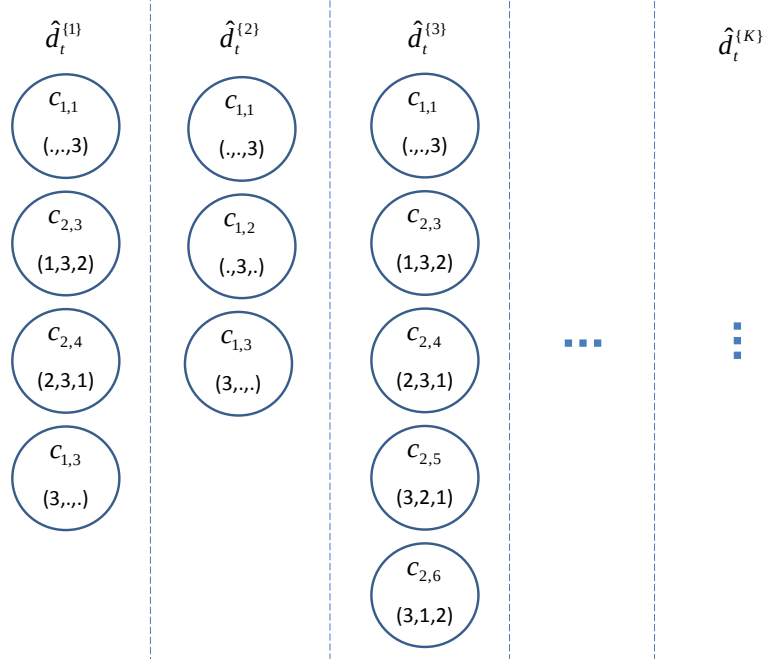


Figure 4.3: 3 possible FS predictors for the equivalence classes in Figure 4.2 with $h = 3$.

having the equivalence classes at the highest hierarchy level, i.e., $\forall c_{h,j} \in \mathcal{H}_h$. However, one expects the finest model to perform better as the observed data length increases considering its higher modeling power for stationary data. On the other hand, when the data is nonstationary, making an efficient switching from coarser to finer FS predictors may not be possible. Therefore, instead of committing to one of these FS predictors or switching between them, we use a mixture-of-experts approach to adaptively combine the outputs of all these FS predictors, $\hat{d}_t^{\{k\}}$, $k = 1, \dots, K$.

4.2 Sequential Combination of FS Predictors

Suppose we construct all possible FS predictors $\hat{d}_t^{\{k\}}$, $k = 1, \dots, K$ and run them in parallel to predict d_t . We use the well-known exponentiated gradient (EG) algorithm [71] to combine the outputs of all FS predictors to produce the final

output

$$\hat{d}_t \triangleq \sum_{k=1}^K w_t^{\{k\}} \hat{d}_t^{\{k\}}, \quad (4.7)$$

where the combination weights are recursively calculated as follows

$$w_t^{\{k\}} = \frac{w_{t-1}^{\{k\}} \exp\left(-\mu \epsilon_{t-1} \hat{d}_{t-1}^{\{k\}}\right)}{\sum_{r=1}^K w_{t-1}^{\{r\}} \exp\left(-\mu \epsilon_{t-1} \hat{d}_{t-1}^{\{r\}}\right)}, \quad (4.8)$$

for $k = 1, \dots, K$, with $\epsilon_t \triangleq \ell'(d_t, \hat{d}_t)$ representing the first derivative of $\ell(d_t, \hat{d}_t)$ with respect to $\hat{d}_t$ and $\mu > 0$ representing a positive constant controlling the learning rate. It can be shown that the weighted mixture algorithm in (4.7) sequentially achieves the performance of the best convex combination of FS predictors [71], i.e., when applied to any data sequences $x_1, x_2, \dots$ and $d_1, d_2, \dots$, this algorithm has the performance

$$\sum_{t=1}^n \ell(d_t, \hat{d}_t) \leq \min_{\|\mathbf{w}\|=1} \sum_{t=1}^n \ell(d_t, \mathbf{w}^T \hat{\mathbf{d}}_t) + O\left(\sqrt{n \log K}\right), \quad (4.9)$$

for any n without any knowledge on the optimal $\hat{d}_t^{\{k\}}$, the future values of the sequences, and the data length n , where $\hat{\mathbf{d}}_t \triangleq [\hat{d}_t^{\{1\}}, \dots, \hat{d}_t^{\{K\}}]^T$ and $\ell(\cdot, \cdot)$ is a Lipschitz continuous loss function. We emphasize that there exist various different extensions of the update in (4.8). Yet, our derivations straightforwardly generalize to these updates [71] as shown in Section 4.4.

Although one can use various cost functions $\ell(\cdot, \cdot)$ in (4.8), a widely used one is $\ell(d_t, \hat{d}_t) = (d_t - \hat{d}_t)^2$ in many signal processing applications when no statistical information about the data is available [13]. On the other hand, in various stochastic settings, steady-state MSE can be more meaningful in terms of analyzing the convergence performance of the algorithm. Therefore, we introduce the following theorem (whose proof can be found in Appendix A.3).

Theorem 4.1 *The weighted mixture algorithm in (4.7) achieves the following the steady-state MSE performance*

$$\text{MSE} = \frac{2\sigma_n^2}{2 - \mu \text{Tr}\left\{\mathbf{\Lambda} \bar{\mathbf{J}}^T\right\}}, \quad (4.10)$$

where the detailed derivation of (4.10) and the explicit descriptions of the parameters σ_n^2 , $\mathbf{\Lambda}$, and $\bar{\mathbf{J}}$ can be found in Appendix A.3. These parameters can be briefly described as follows. σ_n^2 is the variance of the weighted excess error at steady-state, $\mathbf{\Lambda}$ is a diagonal matrix containing the eigenvalues of the autocorrelation matrix of the FS predictor outputs, and $\bar{\mathbf{J}}$ is related to the eigen decomposition of this autocorrelation matrix and the Jacobian matrix of the logarithm of the unnormalized combination weights.

The MSE performance in (4.10) illustrates that we can decrease the steady-state MSE by decreasing the learning rate of the algorithm and achieve the performance of the MMSE optimal batch predictor. Thus, the excess MSE (EMSE) of our algorithm can be arbitrarily set according to the performance requirements of the application by tuning the learning rate of the algorithm.

Although there exists various alternatives of the EG algorithm in the signal processing and machine learning literatures [71, 72], the main advantage of the EG algorithm is its superior tracking performance compared to its well-known alternatives, e.g., the least mean squares (LMS) [13]. Therefore, our algorithm can track abrupt changes or nonstationary data better than its alternatives. Furthermore, our algorithm achieves the optimal linear combination of doubly exponential number of FS predictors, whereas the conventional methods can only track the best expert and achieves its performance in an asymptotical manner.

Although (4.7) is guaranteed to achieve the performance of the optimal combination over doubly exponential number of experts (i.e., FS predictors), it is not possible to practically implement such an algorithm even for a moderate hierarchy depth. As an example, for the order preserving patterns example, even for a small history length such as $l = 4$ and a small hierarchy depth such as $h = 3$, we have $K = 6562$ different FS predictors. However, in practical applications such a huge number of FS predictors cannot be implemented and run in parallel to combine their outputs. Hence, in this form the algorithm (4.7) cannot be directly implemented since we need to run K different FS predictors in parallel and monitor their performances to construct (4.7). To solve this problem, we next introduce a method that implements (4.7) with complexity only linear in

the hierarchy depth h .

4.3 Efficient Sequential Combination of FS Predictors

We first observe that although there are K different FS predictors, the states of each of these predictors are unions of a relatively small number of equivalence classes in Figure 4.2. As an example, $\hat{d}_t^{\{1\}}$ of Figure 4.3 uses only the equivalence class predictors $\hat{d}_t^{\{c_{1,1}\}}$, $\hat{d}_t^{\{c_{2,3}\}}$, $\hat{d}_t^{\{c_{2,4}\}}$, and $\hat{d}_t^{\{c_{1,3}\}}$. If this FS predictor observes the sequence $(x_{t-2}, x_{t-1}, x_t) = (1, 5, -1)$ at time t , which indicates the ordering $(2, 3, 1)$, then $\hat{d}_t^{\{1\}}$ uses the state predictor $\hat{d}_t^{\{c_{2,4}\}}$ to give its final output as $\hat{d}_t^{\{1\}} = \hat{d}_t^{\{c_{2,4}\}}$.

Therefore, although there are K different FS predictors, each of these FS predictors uses the output of only one of the $h+1$ different equivalence class predictors based on the current state. This follows from the construction of the hierarchical model, i.e., each independent state at the highest hierarchy level is connected to a single equivalence class at each of the lower hierarchy levels. To continue with our example, suppose at time t , we observed the pattern $(x_{t-2}, x_{t-1}, x_t) = (1, 5, -1)$, which is included in the equivalence classes $c_{2,4} = (2, 3, 1)$ at the highest level, i.e., level-2. This equivalence class is connected to the equivalence classes $c_{1,2} = (\cdot, 3, \cdot)$ at level-1 and $c_{0,1} = (\cdot, \cdot, \cdot)$ at level-0. Furthermore, an FS predictor cannot have more than one of these equivalence classes since these equivalence classes are super sets of each other. For example, an FS predictor has one and only one of these equivalence classes, $c_{2,4}$, $c_{1,2}$, and $c_{0,1}$ in the set of their states.

Thus, although the summation (4.7) is over K terms, there are actually $h+1$ unique equivalence class predictors (each comes from a different hierarchy level) to be combined. However, we emphasize that although we use only $h+1$ different equivalence class predictors, the sequential algorithm in (4.7) requires all K weights in (4.8). Since all FS predictors have different states, their weights $w_t^{\{k\}}$

are different. In the following, we will show that both the summation in (4.7) and weight calculations in (4.8) can be efficiently implemented yielding the algorithm in Algorithm 7, which illustrates a method to calculate (4.7) and (4.8) in $O(h)$ computation.

To illustrate this, we first introduce a technique to recursively calculate the total sum in the denominator of (4.8). Based on this recursion, we next introduce methods to calculate the numerator of (4.8) and to perform a sequential update of the combined loss. This sequential formulation is then proven to be able to produce the exact same output as in (4.7), however with a significantly reduced computational cost, i.e., only linear in the hierarchy depth h .

4.3.1 A Recursive Calculation

In this section, we introduce a recursive calculation of the denominator of (4.8). We first note that after some algebra, we can write (4.8) as follows

$$w_t^{\{k\}} = \frac{w_0^{\{k\}} \exp \left(-\mu \sum_{z=1}^{t-1} \epsilon_z \hat{d}_z^{\{k\}} \right)}{\sum_{r=1}^K w_0^{\{r\}} \exp \left(-\mu \sum_{z=1}^{t-1} \epsilon_z \hat{d}_z^{\{r\}} \right)}. \quad (4.11)$$

Then, by assigning equal prior weight to each FS predictor, i.e., $w_0^{\{k\}} = 1/K$ for all $k = 1, \dots, K$, we obtain

$$w_t^{\{k\}} = \frac{\exp \left(-\mu \sum_{z=1}^{t-1} \epsilon_z \hat{d}_z^{\{k\}} \right)}{\sum_{r=1}^K \exp \left(-\mu \sum_{z=1}^{t-1} \epsilon_z \hat{d}_z^{\{r\}} \right)}. \quad (4.12)$$

Here, we represent the sum in the denominator of (4.12) as follows

$$L_t \triangleq \sum_{k=1}^K L_t^{\{k\}}, \quad (4.13)$$

where

$$L_t^{\{k\}} \triangleq \exp \left(-\mu \sum_{z=1}^{t-1} \epsilon_z \hat{d}_z^{\{k\}} \right) \quad (4.14)$$

represents the total loss of the k th FS predictor at time t . We then define a function of loss for each equivalence class (cf. Figure 4.2) as follows

$$L_t^{\{c_{i,j}\}} \triangleq \exp \left(-\mu \sum_{z=1}^{t-1} I_z^{\{c_{i,j}\}} \epsilon_z \hat{d}_z^{\{c_{i,j}\}} \right), \quad (4.15)$$

where the indicator function $I_z^{\{c_{i,j}\}}$ is defined as in (4.6).

According to these definitions, we observe that each $L_t^{\{k\}}$ can be written as a product of the errors of its equivalence class predictors. Then, letting $\mathcal{C}^{\{k\}}$ denote the states (or equivalence classes) of the k th FS predictor, we have

$$L_t^{\{k\}} = \prod_{c_{i,j} \in \mathcal{C}^{\{k\}}} L_t^{\{c_{i,j}\}}. \quad (4.16)$$

As an example, for the first FS predictor in Figure 4.3, we have $\mathcal{C}^{\{1\}} = \{c_{1,1}, c_{2,3}, c_{2,4}, c_{1,3}\}$. Based on this observation in (4.16), we next use a recursive formulation in order to efficiently calculate the sum L_t in (4.13). To accomplish this, we start from the highest hierarchy level and go to the lower hierarchy levels by recursively defining intermediate parameters that is a function of the total accumulated loss as follows

$$T_t^{\{c_{i,j}\}} \triangleq L_t^{\{c_{i,j}\}} + \prod_{c_{i+1,k} \in \mathcal{C}^{\{c_{i,j}\}}} T_t^{\{c_{i+1,k}\}}, \quad (4.17)$$

for each equivalence class $c_{i,j}$, where $\mathcal{C}^{\{c_{i,j}\}}$ represents the set of equivalence classes at hierarchy level $i+1$ that are connected to $c_{i,j}$. As an example, for the equivalence class $c_{0,1}$, we have $\mathcal{C}^{\{c_{0,1}\}} = \{c_{1,1}, c_{1,2}, c_{1,3}\}$.

The following lemma (whose proof is given in Appendix A.4) illustrates that these intermediate parameters can be used to recursively calculate the denominator of (4.12).

Lemma 4.2 *The recursive formulations in (4.16) and (4.17) yield $T_t^{\{c_{0,1}\}} = L_t$.*

We next use Lemma 4.2 and the recursive relationship in (4.17) to efficiently calculate the final output of the algorithm in (4.7).

4.3.2 Construction of the Final Predictor

In this section, we construct the final predictor $\hat{d}_t$ in (4.7). Using Lemma 4.2 in (4.12) and putting (4.12) in (4.7), we obtain

$$\begin{aligned}
\hat{d}_t &= \sum_{k=1}^K w_t^{\{k\}} \hat{d}_t^{\{k\}} \\
&= \sum_{k=1}^K \frac{\exp\left(-\mu \sum_{z=1}^{t-1} \epsilon_z \hat{d}_z^{\{k\}}\right)}{\sum_{r=1}^K \exp\left(-\mu \sum_{z=1}^{t-1} \epsilon_z \hat{d}_z^{\{r\}}\right)} \hat{d}_t^{\{k\}} \\
&= \frac{1}{L_t} \sum_{k=1}^K L_t^{\{k\}} \hat{d}_t^{\{k\}}.
\end{aligned} \tag{4.18}$$

In order to compactly represent the term inside the sum in (4.18), we introduce another intermediate parameter as follows

$$\tilde{T}_t^{\{c_{i,j}\}} \triangleq L_t^{\{c_{i,j}\}} \hat{d}_t^{\{c_{i,j}\}} + \tilde{T}_t^{\{c_{i+1,j'}\}} \prod_{\substack{c_{i+1,k} \in \mathcal{C}^{\{c_{i,j}\}} \\ c_{i+1,k} \neq c_{i+1,j'}}} T_t^{\{c_{i+1,k}\}}, \tag{4.19}$$

for all $c_{i,j}$ that contain the current pattern (i.e., $\forall c_{i,j} : s_t \in c_{i,j}$), where $s_t \in c_{i+1,j'}$. In the following lemma (whose proof is given in Appendix A.5), we show that the intermediate parameter in (4.19) can be used to efficiently calculate the final output of the algorithm.

Lemma 4.3 *The recursive formulations in (4.16), (4.17), and (4.19) yield*

$$\tilde{T}_t^{\{c_{0,1}\}} = \sum_{k=1}^K L_t^{\{k\}} \hat{d}_t^{\{k\}}, \tag{4.20}$$

which can be calculated with a computational complexity of $O(h)$.

Using Lemma 4.3, we can construct the final output of our algorithm $\hat{d}_t$ in (4.7) as follows

$$\hat{d}_t = \frac{\tilde{T}_t^{\{c_{0,1}\}}}{T_t^{\{c_{0,1}\}}}. \tag{4.21}$$

According to Lemma 4.3, $\tilde{T}_t^{\{c_{0,1}\}}$ can be calculated with computational complexity $O(h)$. Therefore, if we can find a recursive method to calculate the denominator of (4.21), i.e., $T_t^{\{c_{0,1}\}}$, using the past values $T_{t-1}^{\{c_{i,j}\}}$, then we can sequentially obtain $\hat{d}_t$ at each time t with a computational complexity $O(h)$. In the following section, we address this recursion.

4.3.3 A Low Complexity Sequential Update

In this section, we introduce a low complexity sequential update of $T_{t+1}^{\{c_{0,1}\}}$ using (4.17). Suppose we performed our prediction $\hat{d}_t$, then d_t is revealed and we calculated $\epsilon_t = \ell'(d_t, \hat{d}_t)$. Our task is now to calculate $T_{t+1}^{\{c_{0,1}\}}$ from $T_t^{\{c_{0,1}\}}$ with a computational complexity $O(h)$. Naturally, a recursive formulation for $T_{t+1}^{\{c_{0,1}\}}$ also holds as in (4.17), where we have the terms $T_{t+1}^{\{c_{i,j}\}}$ and $L_{t+1}^{\{c_{i,j}\}}$ instead of the terms $T_t^{\{c_{i,j}\}}$ and $L_t^{\{c_{i,j}\}}$, respectively. However, from time t to $t+1$, due to the indicator function in (4.15), only the equivalence classes that contain the state s_t are effected by this update. Therefore, we have

$$L_{t+1}^{\{c_{i,j}\}} = \begin{cases} L_t^{\{c_{i,j}\}} \exp\left(-\mu \epsilon_t \hat{d}_t^{\{c_{i,j}\}}\right) & , \text{ if } s_t \in c_{i,j} \\ L_t^{\{c_{i,j}\}} & , \text{ otherwise} \end{cases}, \quad (4.22)$$

and similarly we also have $T_{t+1}^{\{c_{i,j}\}} = T_t^{\{c_{i,j}\}}, \forall c_{i,j} : s_t \notin c_{i,j}$. Hence, according to (4.17), we have

$$T_{t+1}^{\{c_{i,j}\}} = L_t^{\{c_{i,j}\}} \exp\left(-\mu \epsilon_t \hat{d}_t^{\{c_{i,j}\}}\right) + T_{t+1}^{\{c_{i+1,j'}\}} \prod_{\substack{c_{i+1,k} \in \mathcal{C}^{\{c_{i,j}\}} \\ c_{i+1,k} \neq c_{i+1,j'}}} T_{t+1}^{\{c_{i+1,k}\}}, \quad (4.23)$$

$\forall c_{i,j} : s_t \in c_{i,j}$, where $c_{i+1,j'} \in \mathcal{C}^{\{c_{i,j}\}} : s_t \in c_{i+1,j'}$.

We next introduce the following lemma (whose proof is given in Appendix A.6), which illustrates the computational complexity of the proposed algorithm.

Lemma 4.4 *The recursive formulations in (4.22) and (4.23) can be calculated in $O(h)$ computations.*

This lemma concludes that (4.21) can be efficiently calculated without any approximations with a computational complexity of $O(h)$. Hence, the introduced algorithm achieves the same deterministic and stochastic performance guarantees as the original algorithm, whose computational complexity is doubly exponential in h .

4.3.4 Summary of the Algorithm

The summary of these steps for the generic case can be outlined as follows (and the complete description of the algorithm is given in Algorithm 7). At time t , we have the variables $T_t^{\{c_{i,j}\}}$, $L_t^{\{c_{i,j}\}}$ and the predictions $\hat{d}_t^{\{c_{i,j}\}}$ for each equivalence class in the entire state diagram.

- After we find the current state s_t , we then recursively calculate $\tilde{T}_t^{\{c_{i,j}\}}$, $\forall c_{i,j} : s_t \in c_{i,j}$ starting from the hierarchy level $i = h - 1$ to get the numerator of (4.18). After this recursion, the output of our algorithm is found as $\hat{d}_t = \tilde{T}_t^{\{c_{0,1}\}} / T_t^{\{c_{0,1}\}}$, which, in total, can be found in $O(h)$ calculations (as $T_t^{\{c_{0,1}\}}$ is calculated at the previous step).
- After the true output d_t is revealed, the variables $L_{t+1}^{\{c_{i,j}\}}$ and $T_{t+1}^{\{c_{i,j}\}}$ should be updated. For the equivalence classes that contain the current pattern (in total $h + 1$ of them), the updates of $L_{t+1}^{\{c_{i,j}\}}$ and $T_{t+1}^{\{c_{i,j}\}}$ are done using the recursions in (4.22) and (4.23), where we emphasize that for the equivalence classes that do not include the current state, no update is necessary.
- Lastly, for the equivalence classes that contain the current state, the equivalence class predictions $\hat{d}_{t+1}^{\{c_{i,j}\}}$ can be updated using any desired method, such as the one in (4.5).

In the following two subsections, we extend our discussions for different linear combination weights and different cost functions, respectively.

Algorithm 7 Online Hierarchical Predictor

```
1: % Initialization:  $L_0^{\{c_{i,j}\}} = 1$ , calculate  $T_0^{\{c_{i,j}\}}$ .
2: for  $t = 1$  to  $T$  do
3:   % Find the current state  $s_t$  by  $s_t = g(s_{t-1}, x_t)$ .
4:   % Find the set of equivalence classes  $\mathcal{E}_t$  that contain the current state  $s_t$ ,
   i.e.,  $\mathcal{E}_t = \{c_{i,j} : s_t \in c_{i,j}\}$ .
5:   % Prediction
6:   for all  $c_{i,j} \in \mathcal{E}_t$  (from  $i = h - 1$  to 0) do
7:     if  $i = h - 1$  then
8:        $\tilde{T}_t^{\{c_{i,j}\}} = L_t^{\{c_{i,j}\}} \hat{d}_t^{\{c_{i,j}\}}$ 
9:     else
10:       $P_t^{\{c_{i,j}\}} = (T_t^{\{c_{i,j}\}} - L_t^{\{c_{i,j}\}}) / T_t^{\{c_{i+1,j'}\}}$ 
11:       $\tilde{T}_t^{\{c_{i,j}\}} = L_t^{\{c_{i,j}\}} \hat{d}_t^{\{c_{i,j}\}} + \tilde{T}_t^{\{c_{i+1,j'}\}} P_t^{\{c_{i,j}\}}$ 
12:    end if
13:  end for
14:   $\hat{d}_t = \tilde{T}_t^{\{c_{0,1}\}} / T_t^{\{c_{0,1}\}}$ 
15:   $\epsilon_t = \ell'(d_t, \hat{d}_t)$ 
16:  % Update
17:  for all  $c_{i,j} \in \mathcal{E}_t$  (from  $i = h - 1$  to 0) do
18:    % Update  $\hat{d}_t^{\{c_{i,j}\}}$  as desired such as in (4.5).
19:     $L_{t+1}^{\{c_{i,j}\}} = L_t^{\{c_{i,j}\}} \exp(-\mu \epsilon_t \hat{d}_t^{\{c_{i,j}\}})$ 
20:    if  $i = h - 1$  then
21:       $T_{t+1}^{\{c_{i,j}\}} = L_{t+1}^{\{c_{i,j}\}}$ 
22:    else
23:       $T_{t+1}^{\{c_{i,j}\}} = L_{t+1}^{\{c_{i,j}\}} + T_{t+1}^{\{c_{i+1,j'}\}} P_t^{\{c_{i,j}\}}$ 
24:    end if
25:  end for
26: end for
```

4.4 Positive and Negative Weights

In many mixture-of-experts frameworks, the weighting parameters are usually restricted to be positive and sum up to 1 as in the case for the EG algorithm (cf. (4.12)). In order to overcome this limitation, we can use the EG algorithm with positive and negative weights [71]. To this end, we consider the output of each FS predictor, say $\hat{d}_t^{\{k\}}$, and instead of directly scaling this value with a weighting parameter, we consider $\beta \hat{d}_t^{\{k\}}$ and $-\beta \hat{d}_t^{\{k\}}$ as the outputs of two different experts. We then scale this values by the weighting parameters $w_{t,+}^{\{k\}}$ and $w_{t,-}^{\{k\}}$, respectively.

Hence, we obtain the output of the final predictor as follows

$$\hat{d}_t = \sum_{k=1}^K \left(w_{t,+}^{\{k\}} - w_{t,-}^{\{k\}} \right) \hat{d}_t^{\{k\}}, \quad (4.24)$$

where the combination weights are recursively calculated as follows

$$w_{t+1,\pm}^{\{k\}} = \beta \frac{w_{t,\pm}^{\{k\}} \exp \left(\mp \mu \beta \epsilon_t \hat{d}_t^{\{k\}} \right)}{\sum_{r=1}^K w_{t,+}^{\{r\}} \exp \left(-\mu \beta \epsilon_t \hat{d}_t^{\{r\}} \right) + w_{t,-}^{\{r\}} \exp \left(\mu \beta \epsilon_t \hat{d}_t^{\{r\}} \right)}. \quad (4.25)$$

In this manner, while we have $w_{t,+}^{\{k\}}, w_{t,-}^{\{k\}} > 0$, $\sum_{k=1}^K w_{t,+}^{\{k\}} = \beta$, and $\sum_{k=1}^K w_{t,-}^{\{k\}} = \beta$, the resulting combination weights in (4.24), i.e., $w_{t,+}^{\{k\}} - w_{t,-}^{\{k\}}$ can take any value satisfying $\sum_{k=1}^K \left| w_{t,+}^{\{k\}} - w_{t,-}^{\{k\}} \right| \leq \beta$.

Following similar lines to Section 4.3, we define

$$L_t \triangleq L_{t,+} + L_{t,-}, \quad (4.26)$$

where

$$L_{t,\pm} \triangleq \exp \left(\mp \mu \beta \sum_{z=1}^t \epsilon_z \hat{d}_z^{\{k\}} \right). \quad (4.27)$$

After defining the equivalence class losses and recursion parameters as in Section 4.3, we obtain independent recursions over parameters $L_{t,\pm}^{\{c_{i,j}\}}$ and $T_{t,\pm}^{\{c_{i,j}\}}$, where $L_{t,\pm}^{\{c_{i,j}\}}$ represents the loss of the equivalence class $c_{i,j}$ for the positive and negative weights, respectively, which is defined similar to (4.15) and the parameters $T_{t,\pm}^{\{c_{i,j}\}}$ can be calculated with independent recursions as follows

$$T_{t,\pm}^{\{c_{i,j}\}} = L_{t,\pm}^{\{c_{i,j}\}} + \prod_{c_{i+1,k} \in \mathcal{C}^{\{c_{i,j}\}}} T_{t,\pm}^{\{c_{i+1,k}\}}. \quad (4.28)$$

Using these definitions, one can obtain the final algorithm after following similar lines to Section 4.3.

4.5 Implementation of the Algorithm with Forgetting Factor

In this section, we consider the weighted loss as our cost function, i.e.,

$$\sum_{t=1}^n \lambda^{n-t} \ell(d_t, \hat{d}_t), \quad (4.29)$$

where $0 < \lambda < 1$ represents the forgetting factor. We emphasize that the objective function in (4.29) is used in various different applications, especially when the aim is to track a drifting parameter [13]. However, directly using (4.29) as our objective function may significantly deteriorate the performance of the algorithm since $\hat{d}_t$ is generated according to the current state s_t and even for a moderate h , the time difference between two consecutive appearances of the same state may take significantly long time, e.g., for the order preserving patterns scenario this recurrence time is $\sim (h/e)^h$. Therefore, instead of using (4.29) in its current form, we use different forgetting factors for each equivalence class as follows

$$\sum_{t_z \in \mathcal{T}_n^{\{c_{i,j}\}}} \lambda^{|\mathcal{T}_n^{\{c_{i,j}\}}| - z} \ell(d_t, \hat{d}_t), \quad (4.30)$$

where $\mathcal{T}_t^{\{c_{i,j}\}}$ represents the time instances (up to t) at which the current state is included in the equivalence class $c_{i,j}$, i.e., $\mathcal{T}_t^{\{c_{i,j}\}} \triangleq \{1 \leq t_z \leq t : s_{t_z} \in c_{i,j}\}$, and $t_z \in \mathcal{T}_t^{\{c_{i,j}\}}$ represents the z th smallest value in the set $\mathcal{T}_t^{\{c_{i,j}\}}$. By this formulation we can overcome the aforementioned recurrence time issue and also assign different forgetting factors for each equivalence class $c_{i,j}$, i.e., using $\lambda_{c_{i,j}}$ instead of λ .

Owing to the comprehensive structure of the introduced algorithm such recursive cost functions can be directly incorporated in our framework. To this end, we can define our new total loss function as follows

$$L_t = \sum_{k=1}^K L_t^{\{k\}}, \quad (4.31)$$

where

$$L_t^{\{k\}} = \exp \left(-\mu \sum_{z=1}^{t-1} \lambda^{t-z-1} \epsilon_z \hat{d}_z^{\{k\}} \right) \quad (4.32)$$

represents the total weighted loss of the k th FS predictor at time t . Following similar lines to Section 4.3, we define $T_t^{\{c_{i,j}\}}$ similar to (4.17). Then, the recursion of the parameter $L_t^{\{c_{i,j}\}}$ is updated as follows

$$L_{t+1}^{\{c_{i,j}\}} = \exp\left(\lambda \ln L_t^{\{c_{i,j}\}}\right) \exp\left(-\mu \epsilon_t \hat{d}_t^{\{c_{i,j}\}}\right), \quad (4.33)$$

$\forall c_{i,j} : s_t \in c_{i,j}$, whereas the parameter $T_t^{\{c_{i,j}\}}$ is updated as in (4.23). Note that for the equivalence classes that do not contain the current state, no update is necessary. Following similar steps to Section 4.3, one can construct the desired algorithm after some algebra.

4.6 Simulations

In this section, we illustrate the performance of our algorithm in various scenarios. Throughout this section, we set $\mu = 1$ for a fair performance comparison between our algorithm and its competitors.

4.6.1 Real Life Energy Profile Forecasting

In this experiment, we consider prediction of the energy consumption in Turkish energy markets using real data [73]. Particularly, we forecast the energy consumption of consumers using their past consumption patterns, where the aim is to predict the consumption trend such that $d_t = 1$ if $x_{t+1} \geq x_t$ and $d_t = -1$, otherwise, i.e., we try to forecast an increasing or decreasing trend in the energy consumption patterns using real data collected in Turkish markets. Note that this scenario perfectly matches with the example framework we have illustrated throughout the chapter. In this experiment, we illustrate the performance guarantee in Theorem 1 and the mitigation of undertraining as well as convergence issues by our algorithm. We set $h = 4$ for this real life experiment. In Figure 4.4, the accumulated square error performances (normalized with time) of the proposed algorithms are compared, where “OHP” represents the online hierarchical

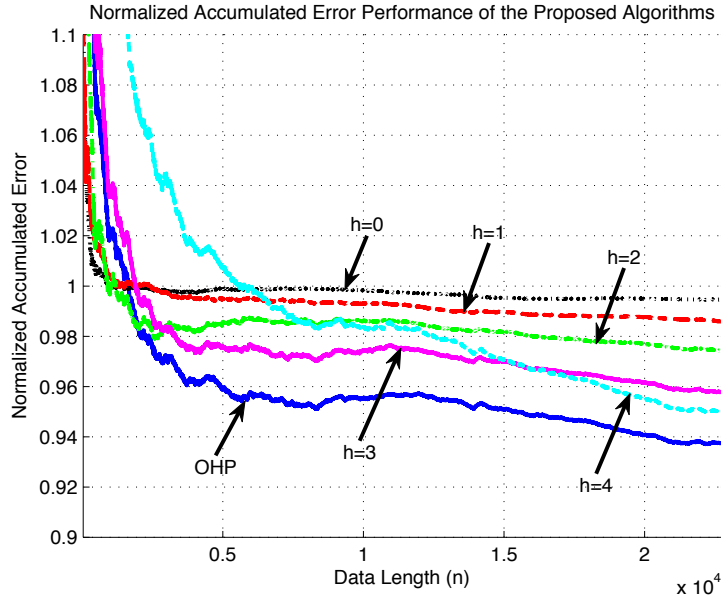


Figure 4.4: Normalized cumulative square errors of the proposed algorithms for the real life electricity consumption data.

predictor introduced in this chapter and “ $h = i$ ” represents the predictor using all equivalence classes at the i th hierarchy level as its states (e.g., see example hierarchy levels in Figure 4.2 for $h = 2$).

Figure 4.4 illustrates that the performance of the OHP algorithm is comparable with the performances of the coarser predictors (e.g., $h = 0$ and $h = 1$) when there is not sufficient amount of data to train finer energy consumption patterns (equivalence classes). However, as the data length increases, the performances of the coarser predictors deteriorate with respect to the predictors having finer equivalence classes (e.g., $h = 3$ and $h = 4$). Nevertheless, the performance of the OHP algorithm is still better than the finest predictor even after a significantly large amount of observations.

We emphasize that as the pattern order h increases or when the underlying data is highly nonstationary, the convergence performance of the OHP algorithm will significantly outperform the performance of the finer predictors since the finer predictors may not be able to observe enough training sequences to achieve a satisfactory performance. This result is also apparent in Figure 4.4, where over

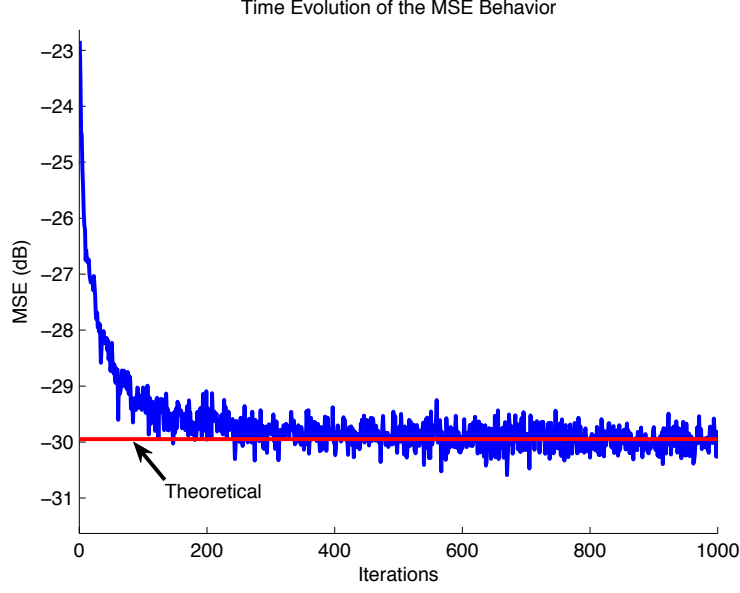


Figure 4.5: The experimental MSE of the proposed algorithm converges to the theoretical steady-state MSE performance. The results are averaged over 500 independent trials.

short data sequences the performance of the finer predictors is worse compared to the coarser predictors and the OHP algorithm. Hence, the OHP algorithm outperforms the constituent FS predictors by exploiting the time-dependent nature of the best choice among constituent FS predictors that are defined on the hierarchical structure.

4.6.2 Synthetic MSE Analysis

In this section, we illustrate the steady-state MSE performance of our algorithm. To this end, we consider the following prediction model

$$x_t = 0.5x_{t-1} + 0.5x_{t-2} + n_t, \quad (4.34)$$

where n_t represents the realization of the i.i.d. additive Gaussian noise with zero mean and variance $\sigma_n^2 = 10^{-3}$ at time t . The equivalence classes, in this experiment, are defined according to the partitioning of the space of the past data. To be more precise, the space of the past data is chosen to be $x_{t-1} \times x_{t-2}$. Then, this space is partitioned in a hierarchical manner and a different linear

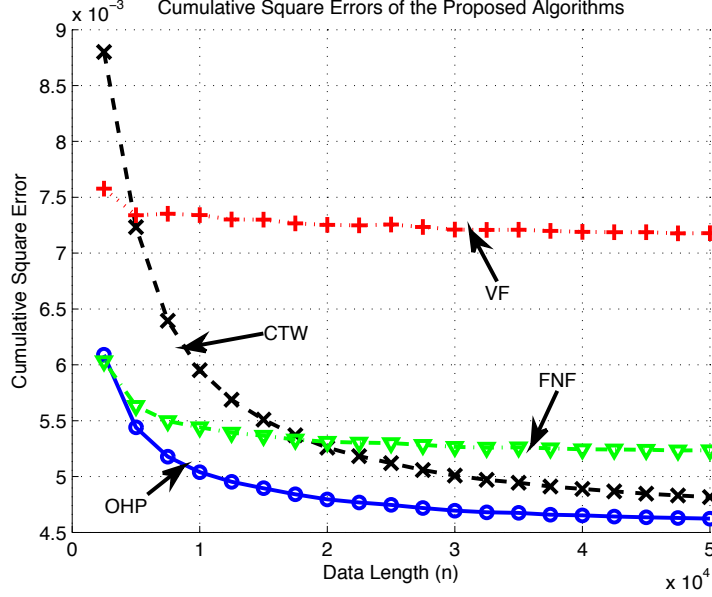


Figure 4.6: Normalized cumulative square errors of the proposed algorithms for the SETAR model in (4.35).

predictor is assigned to each node. We emphasize that such piecewise linear prediction scenarios are extensively studied in the literature, cf. [2, 4, 74] (and references therein). We set the depth of the hierarchical structure to 2.

In Figure 4.5, we illustrate the theoretical and experimental MSE results averaged over 500 independent trials. This figure shows that the actual MSE behavior of our algorithm can be quite accurately represented by the theoretical steady-state MSE result in (4.10).

4.6.3 SETAR Time Series Prediction

In this set of experiments, we consider the prediction of the signals generated from self-exciting threshold autoregressive (SETAR) models. As the first experiment, we consider the following SETAR model

$$x_t = \begin{cases} 1.71x_{t-1} - 0.81x_{t-2} + 0.356 + \varepsilon_t & , \text{ if } x_{t-1} > 0 \\ -0.562x_{t-2} - 3.91 + \varepsilon_t & , \text{ otherwise} \end{cases}, \quad (4.35)$$

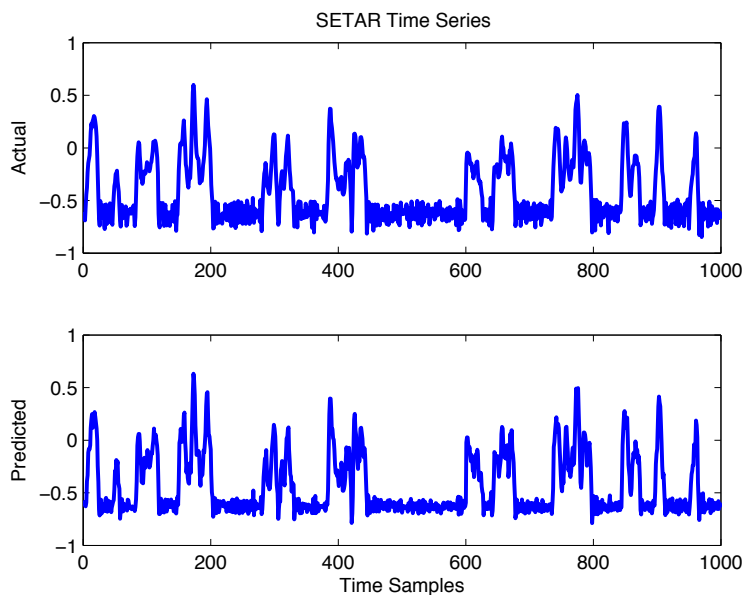


Figure 4.7: The actual and the predicted time series for the SETAR model in (4.35) over a data length of 1000.

where ε_t represents the realization of the i.i.d. additive Gaussian noise with zero mean and unit variance at time t . Note that this SETAR model is used in various other papers, e.g., [2]. The equivalence classes, in this set of experiments, are defined according to the partitioning of the space of the past data similar to the previous experiment. Here, we normalize the both dimensions of this space between $[-1, 1]$ to provide a fair comparison between algorithms.

Throughout this section, “OHP” represents the online hierarchical predictor proposed in this chapter, “CTW” represents the context tree weighting algorithm of [4], “VF” represents the Volterra filter [75], and “FNF” represents the Fourier nonlinear filter of [7]. We set the depths of the OHP and CTW algorithms to 4 and the order of the VF and FNF algorithms to 3 for a fair performance comparison (since the computational complexities of the OHP and CTW algorithms scale linearly with their depth, whereas the computational complexities of the VF and FNF algorithms scale exponentially with their order). To train the linear predictors at each node of the OHP and CTW algorithms and to update the VF and FNF algorithms, we use the recursive least squares (RLS) method [13].

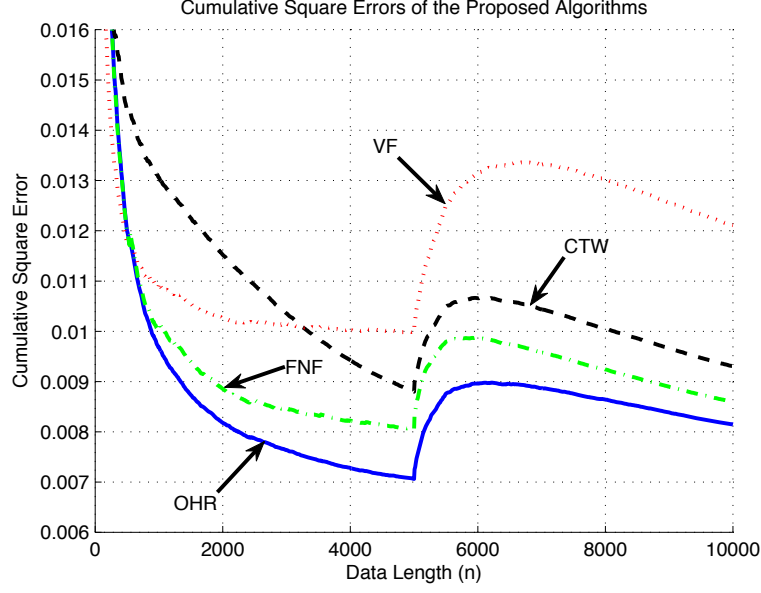


Figure 4.8: Normalized cumulative square errors of the proposed algorithms for the SETAR models in (4.35) and (4.36). Here, the first 5000 samples of the data are generated using (4.35), whereas the last 5000 are generated using (4.36).

In Figure 4.6, we present the cumulative square errors of the OHP, CTW, VF, and FNF algorithms. This figure indicates that our algorithm has a significantly faster convergence rate especially with respect to other tree based learning methods such as the CTW algorithm. That is because, our algorithm achieves the optimal combination of the different FS predictors, whereas the CYW algorithm uses ad-hoc weights to achieve universal performance guarantees. Furthermore, the OHP algorithm achieves a significantly lower square error performance with respect to its competitors. To illustrate the accuracy of our algorithm, in Figure 4.7, we present a time segment of the actual and the predicted time series.

To illustrate the convergence rate of our algorithm, we generate a time series of length 5000 using the SETAR model in (4.35) and then switch to the following SETAR model

$$x_t = \begin{cases} -1.71x_{t-1} + 0.81x_{t-2} - 0.356 + \varepsilon_t & , \text{ if } x_{t-1} \leq 0 \\ 0.562x_{t-2} + 3.91 + \varepsilon_t & , \text{ otherwise} \end{cases}, \quad (4.36)$$

and generate an additional time series of length 5000. In Figure 4.8, we present the cumulative square errors of the proposed algorithms. This figure illustrates

that the convergence rate of our algorithm is exceptionally better than ones of the competitor algorithms. Particularly, our algorithm presents a faster convergence compared to the similar tree based methods such as the CTW algorithm. That is because, our algorithm is based on the EG method, which dynamically updates the combination weights (owing to the exponentiated steps), whereas the CTW algorithm cannot perform such quick adaptations due to its universality over the entire data history. In this sense, our algorithm is highly efficient for applications involving nonstationary data.

Chapter 5

Sequential Nonlinear Learning for Distributed Multi-Agent Systems via Feedforward Networks

In this chapter, we study distributed sequential training of SLFN structures, which can be used in various applications including nonlinear optimization, regression and classification. As an example, in wireless sensor networks, a number of agents observe different data sequences related to (or generated by) a phenomenon of interest. In centralized approach to this framework (cf. [33] and references therein), the entire data sequences are required to be processed by a single centralized agent. To this end, each agent transmits its observations into a centralized processor, where a sequential algorithm, e.g., online ELM [33] is applied. However, this centralized structure cannot process the data in a truly sequential manner since the data is delayed due to transmission and then processed in chunks at the centralized processor. Furthermore, each agent generates huge volumes of data due to the recent developments in hardware technologies [39]. Therefore, centralized processing of such huge amounts of data may not be feasible or even possible due to computational complexity and storage issues [40].

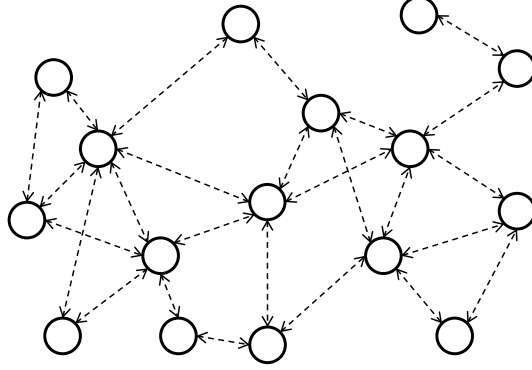


Figure 5.1: An example multi-agent network. Each agent is connected to and communicate with a set of other agents, which form its neighborhood.

To resolve these problems, we introduce a distributed framework, where each agent sequentially processes its own data and shares the extracted information with its neighbors as shown in Figure 5.1. Thus, the data is processed in a truly sequential and completely decentralized manner. Nevertheless, as rigorously shown in Section 5.4, the proposed distributed sequential method achieves asymptotically the same performance with its optimal centralized batch variant.

5.1 Extreme Learning Machines

More formally, at each time t , each agent $k \in \{1, \dots, K\}$ observes a pair of input vector and target data, i.e., $(\mathbf{x}_{t,k}, y_{t,k})$, where $\mathbf{x}_{t,k} \in \mathbb{R}^m$ and $y_{t,k} \in \mathbb{R}$. The aim of each agent is to learn the relationship between these input vector and target data sequences. This learning process is performed in a sequential manner such that after $y_{t,k}$ is predicted and its true value is revealed, each agent updates its prediction model in order to more accurately predict the next sample $y_{t+1,k}$. In many real life applications, this relationship between the input vectors and the target data is highly nonlinear [20]. Hence, in order to effectively capture such nonlinear relationships, at each agent k , we use a SLFN with N hidden nodes and estimate the target data at each agent $k \in \{1, \dots, K\}$ as

$$\hat{y}_{t,k} = \sum_{n=1}^N w_{t,n,k} z(\mathbf{a}_{n,k}, b_{n,k}; \mathbf{x}_{t,k}), \quad (5.1)$$

where for the k th agent, $w_{t,n,k}$ is the weight connecting the n th hidden node to the output node, $z(\mathbf{a}_{n,k}, b_{n,k}; \mathbf{x}_{t,k})$ is the output of the n th hidden node for the input $\mathbf{x}_{t,k}$, $\mathbf{a}_{n,k} \in \mathbb{R}^m$ is the weight vector connecting the input layer to the n th hidden node, and $b_{n,k} \in \mathbb{R}$ is the bias parameter of the n th hidden node. We can randomly pick the weight vector $\mathbf{a}_{n,k}$ and the bias parameter $b_{n,k}$ without any performance degradation for infinitely differentiable activation functions $z(\cdot, \cdot; \cdot)$ as shown in [33]. Thus, we randomly generate a weight vector $\mathbf{a}_n$ and a bias parameter b_n , then use these parameters at each agent, i.e., we set $\mathbf{a}_{n,k} = \mathbf{a}_n$ and $b_{n,k} = b_n, \forall k \in \{1, \dots, K\}$.¹ Then, we can rewrite (5.1) in the vector form as

$$\hat{y}_{t,k} = \langle \mathbf{w}_{t,k}, \mathbf{z}_{t,k} \rangle, \quad (5.2)$$

where

$$\begin{aligned} \mathbf{z}_{t,k} &\triangleq [z(\mathbf{a}_{1,k}, b_{1,k}; \mathbf{x}_{t,k}), \dots, z(\mathbf{a}_{N,k}, b_{N,k}; \mathbf{x}_{t,k})]^T, \\ \mathbf{w}_{t,k} &\triangleq [w_{t,1,k}, \dots, w_{t,N,k}]^T. \end{aligned} \quad (5.3)$$

In the distributed setting, the aim of each agent is to minimize a cost function $f_{t,k}(\cdot) + r_k(\cdot)$ at time t over a convex set $\mathcal{W}$, where $f_{t,k}(\cdot)$ and $r_k(\cdot)$ are generic convex functions. Here, for the k th agent, the function $f_{t,k}(\cdot)$ is an empirical loss and the function $r_k(\cdot)$ is a regularization term. As an example, by setting $f_{t,k}(\mathbf{w}) = (y_t - \langle \mathbf{w}, \mathbf{z}_{t,k} \rangle)^2$ and $r_k(\mathbf{w}) = \lambda \|\mathbf{w}\|_p$, we obtain the regularized least squares cost function. We emphasize that we consider this generic cost function since we can train the ELMs using different cost functions instead of the squared error loss depending on the application. Our approach covers these different cases and models a broad range of practical cost functions that are widely used in optimization [76], machine learning [77], and signal processing [26] literatures. In a more formal manner, the aim of each agent is to sequentially solve the following optimization problem

$$\text{minimize } \sum_{t=1}^T [f_{t,k}(\mathbf{w}_{t,k}) + r_k(\mathbf{w}_{t,k})], \quad (5.4)$$

$$\text{subject to } \mathbf{w}_{t,k} \in \mathcal{W}, \quad (5.5)$$

¹This procedure can be performed before the processing starts by exchanging either the corresponding parameters or the seed of the random number generator.

for any T , where T is the length of the data, which is arbitrary and unknown, i.e., the agents perform optimization in a truly sequential manner without the knowledge of the time horizon.

Although the objective of each agent is to minimize the cost function in (5.4), the aim in the distributed framework is to propose an algorithm that achieves asymptotically the same performance as the optimal batch centralized processor. To this end, we define the following global loss functions

$$f_t(\mathbf{w}_{t,k}) \triangleq \sum_{j=1}^K f_{t,j}(\mathbf{w}_{t,k}), \quad (5.6)$$

$$r(\mathbf{w}_{t,k}) \triangleq \sum_{j=1}^K r_j(\mathbf{w}_{t,k}), \quad (5.7)$$

which represent the loss of the SLFN weight vector of the k th agent on the data observed by all agents on the distributed network at time t . In this way, we aim to construct a distributed learning algorithm that is consistent among all the agents, i.e., each agent should be able to acquire the information from every input and output data pairs observed by every other agent. In particular, our aim is to perform the following optimization

$$\min_{\mathbf{w}_{t,k} \in \mathcal{W}} \sum_{t=1}^T [f_t(\mathbf{w}_{t,k}) + r(\mathbf{w}_{t,k})], \quad (5.8)$$

over all $k \in \{1, \dots, K\}$. Here, we emphasize that we avoid any statistical assumptions on the input vector and the target data. Instead, our aim is to introduce a sequential algorithm that provides guaranteed performance in a strong mathematical sense without any statistical assumptions [20]. For this, we next introduce an algorithm, which is used by each agent to minimize the cost function of that agent, cf. (5.6), (5.7). Then, SLFN weight vectors trained by this algorithm are diffused among the agents to solve the global SLFN-based optimization problem in (5.8).

5.2 The Forward-Backward Splitting Method

For applications involving big data and/or requiring sequential processing, the dimensionality of the input data can be relatively large [20]. Hence, we aim to design highly efficient algorithms with low complexity to solve the optimization problem in (5.8). To this end, we use subgradient methods [41] due to their computational efficiency and generalization capabilities [42]. The subgradient of a function i at $\mathbf{w}$ is defined as follows

$$\partial i(\mathbf{w}) = \{\mathbf{g} \in \mathbb{R}^m : i(\mathbf{v}) \geq i(\mathbf{w}) + \langle \mathbf{g}, \mathbf{v} - \mathbf{w} \rangle, \forall \mathbf{v} \in \mathbb{R}^m\}. \quad (5.9)$$

By using the subgradient method to minimize (5.6) at each individual node k , we obtain iterates of the form $\mathbf{w}_{t+1,k} = \mathbf{w}_{t,k} - \mu_t \mathbf{g}_{t,k} - \mu_t \mathbf{h}_{t,k}$, where μ_t is the learning rate of the subgradient method, $\mathbf{g}_{t,k} \in \partial f_{t,k}(\mathbf{w}_t)$, and $\mathbf{h}_{t,k} \in \partial r_k(\mathbf{w}_t)$. However, if the cost functions $f_{t,k}(\cdot)$ or $r_k(\cdot)$ are non-differentiable at certain points, then the iterates $\mathbf{w}_{t,k}$ cannot usually attain these values [76], which are usually the desired minima of the cost functions. As an example, for $r_k(\mathbf{w}) = \lambda \|\mathbf{w}\|_1$, the desired minima is rarely achievable by the aforementioned update rule. We emphasize that such non-differentiable cost functions are usually studied in the context of optimization and learning theory via SLFNs and ELM methods [33].

To address this problem, we use the forward-backward splitting method [78]. In particular, without loss of generality we assume that the cost function $f_{t,k}(\cdot)$ is differentiable and our aim is to ensure that the iterates $\mathbf{w}_{t,k}$ attain the non-differentiable points of the function $r_k(\cdot)$ (Note that if $f_{t,k}(\cdot)$ is non-differentiable at some points whereas $r_k(\cdot)$ is differentiable, one can easily switch the labels of the cost functions and apply the below procedure). To solve the optimization problem in (5.6), we perform the following updates

$$\mathbf{w}'_{t+1,k} = \mathbf{w}_{t,k} - \mu_t \mathbf{g}_{t,k}, \quad (5.10)$$

$$\mathbf{w}_{t+1,k} = \arg \min_{\mathbf{w}} \frac{1}{2} \|\mathbf{w} - \mathbf{w}'_{t+1,k}\|^2 + \mu_t r_k(\mathbf{w}), \quad (5.11)$$

where (5.10) updates the parameter vector using a subgradient of the first function $f_{t,k}(\cdot)$. Then, in (5.11), we seek to minimize the regularization cost $r_k(\cdot)$ while

not getting too far from the intermediate parameter vector calculated in (5.10). Note that by using the updates in (5.10) and (5.11), we make sure that the iterates attain the aforementioned non-differentiable points due to the following observation: The zero vector should be an element of the set of subgradients of (5.11). Therefore, $\mathbf{0} \in \mathbf{w}_{t+1,k} - \mathbf{w}'_{t+1,k} + \mu_t \partial r_k(\mathbf{w}_{t+1,k})$. This indicates that there always exists a subgradient vector $\mathbf{h}_{t+1,k} \in \partial r_k(\mathbf{w}_{t+1,k})$ such that

$$\mathbf{w}_{t+1,k} = \mathbf{w}_{t,k} - \mu_t \mathbf{g}_{t,k} - \mu_t \mathbf{h}_{t+1,k}, \quad (5.12)$$

i.e., we can reach any non-differentiable points.

In the following, we introduce a distributed algorithm that use the subgradient method described in this section to train SFLN models in a truly sequential manner over distributed architectures and derive the corresponding guaranteed performance bounds.

5.3 Distributed Sequential Splitting Extreme Learning Machine (DSS-ELM)

In this section, we introduce a sequential distributed SLFN-based algorithm, which uses the forward-backward splitting method for optimization (the complete description of the algorithm is given in Algorithm 8). At each time t , each agent k calculates its estimate of the target data as in (5.2) and then diffuses the weights of the SFLN, i.e., $\mathbf{w}_{t,k}$, to its neighbors as shown in Figure 5.1. Each agent k , after receiving the weight vectors from the neighboring nodes, perform a weighted averaging

$$\phi_{t+1,k} = \sum_{j=1}^K \mathbf{C}_{jk} \mathbf{w}_{t,j}, \quad (5.13)$$

to construct an intermediate weight vector, where $\mathbf{C}$ is the communication matrix of the graph such that $\mathbf{C}_{jk}$'s are the combination weights (e.g., these weights can be chosen according to the Metropolis rule [39]). After diffusion of the weights of the SLFN, each agent then performs its update using the forward-backward

Algorithm 8 Distributed Sequential Splitting ELM (DSS-ELM)

```

1: for  $t = 1$  to  $T$  do
2:   for  $k = 1$  to  $K$  do
3:      $\hat{y}_{t,k} = \langle \mathbf{w}_{t,k}, \mathbf{z}_{t,k} \rangle$ 
4:      $\phi_{t+1,k} = \sum_{j=1}^K \mathbf{H}_{jk} \mathbf{w}_{t,j}$ 
5:      $\mathbf{w}'_{t+1,k} = \phi_{t+1,k} - \mu_t \mathbf{g}_{t,k}$ 
6:      $\mathbf{w}_{t+1,k} = \arg \min_{\mathbf{w}} \frac{1}{2} \|\mathbf{w} - \mathbf{w}'_{t+1,k}\|^2 + \mu_t r_k(\mathbf{w})$ 
7:   end for
8: end for

```

splitting method based on the combined weight vector as described in Section 5.2.

For this sequential distributed kernel-adaptive learning algorithm, we have the following theorem, which states that the introduced algorithm minimizes a sum of convex objective functions with a regret of $\mathcal{O}(K^{1.25}\sqrt{T})$, uniformly at each agent. We note that the convergence result in the following theorem uniformly holds for all agents, without any statistical assumptions on the data and without any knowledge of the time horizon, i.e., our algorithm does not need T in hindsight. Hence, the proposed algorithm is truly sequential and consequently highly suitable for applications involving high-dimensional data. We also emphasize that the proposed algorithm is the first complete implementation of a SLFN-based learning algorithm over distributed networks with guaranteed performance bounds.

Theorem 5.1 *Under Assumptions 1-4 (listed after Corollary 5.2), Algorithm 8, when applied to any input and output data sequences, achieves the following convergence guarantee*

$$\sum_{t=1}^T [f_t(\mathbf{w}_{t,k}) + r(\mathbf{w}_{t,k})] - \min_{\mathbf{w} \in \mathcal{W}} \sum_{t=1}^T [f_t(\mathbf{w}) + r(\mathbf{w})] \leq \mathcal{O} \left(\frac{AD}{\sqrt{\alpha}} K^{1.25} \sqrt{T} \right), \quad (5.14)$$

for all T and $\forall k \in \{1, \dots, K\}$, with a suitable learning rate. Here, α is the spectral gap of the communication matrix $\mathbf{C}$, D is the diameter of the convex set of the optimization problem, and A is an upper bound on the norms of the subgradients.

This theorem states that the weights of the SLFNs at every agent uniformly achieve the performance of the optimal weights that can be chosen only in hindsight, i.e., after observing all the data in a batch mode and centralized manner. We emphasize that although each agent observes a data of length T , all agents can still achieve the performance of the centralized batch algorithm (which processes the entire data of length KT) through the communication between agents. Furthermore, if the cost function $f_{t,k}(\cdot)$ is F -strongly convex and $r_k(\cdot)$ is R -strongly convex, where $S \triangleq F + R > 0$, then Algorithm 8 achieves a convergence guarantee of $\mathcal{O}(K^{1.5} \log T)$ as presented in the following corollary.

Corollary 5.2 *Under Assumptions 1-4 (listed below) and when the sum of the cost functions are S -strongly convex for some $S > 0$, Algorithm 8, when applied to any input and output data sequences, achieves the following convergence guarantee*

$$\sum_{t=1}^T [f_t(\mathbf{w}_{t,k}) + r(\mathbf{w}_{t,k})] - \min_{\mathbf{w} \in \mathcal{W}} \sum_{t=1}^T [f_t(\mathbf{w}) + r(\mathbf{w})] \leq \mathcal{O} \left(\frac{A^2}{\alpha} K^{1.5} \log T \right), \quad (5.15)$$

for all T and $\forall k \in \{1, \dots, K\}$, with a suitable learning rate. Here, α is the spectral gap of the communication matrix $\mathbf{C}$ and A is an upper bound on the norms of the subgradients.

For the proofs of Theorem 5.1 and Corollary 5.2, we make the following technical assumptions that are widely used in the machine learning literature [41–43, 77–84]:

1. The convex set $\mathcal{W}$ has a diameter of D , i.e., $\|\mathbf{w} - \mathbf{v}\| \leq D$, $\forall \mathbf{w}, \mathbf{v} \in \mathcal{W}$.
2. The norms of the subgradients are bounded by A , i.e., $\|\mathbf{g}_{t,k}\|, \|\mathbf{h}_{t,k}\| \leq A$, $\forall \mathbf{g}_{t,k} \in \partial f_{t,k}(\mathbf{w}_{t,k})$ and $\forall \mathbf{h}_{t,k} \in \partial r_k(\mathbf{w}_{t,k})$
3. The communication graph $\mathbf{C}$ forms a doubly stochastic matrix such that $\mathbf{C}$ is irreducible and aperiodic, i.e., $\mathbf{C}\mathbf{1} = \mathbf{C}^T\mathbf{1} = \mathbf{1}$.
4. Initial weights are equal at each node, i.e., $\mathbf{w}_{1,k} = \mathbf{w}_{1,j}$, $\forall k, j \in \{1, \dots, K\}$.

We point out that similar assumptions are often used to derive convergence guarantees for sequential algorithms, cf. [41–43, 77–79, 81–83]. Here, the first assumption enforces a bounded parameter vector, which is clearly realistic since our cost function has regularization penalties. The second assumption holds when we have a bounded input vector and the outputs of the hidden nodes are bounded-input bounded-output (BIBO) stable (since these outputs are infinitely differentiable). Third assumption enforces that the contributions of each agent to the overall multi-agent framework are equal. The last assumption is an unbiasedness condition, which is used only for presentation purposes and our bounds hold (with $\mathcal{O}(1)$ excess factors) even when this assumption is not satisfied.

5.4 Proofs of Theorem 5.1 & Corollary 5.2

In order to relate the performance of the weights of an individual agent to the optimal weights, we define an average weight vector. We first compare the performance of this weight vector with respect to the optimal weights and then relate this average weight to the weights at each agent. To this end, let

$$\bar{\mathbf{w}}_t = \frac{1}{K} \sum_{k=1}^K \mathbf{w}_{t,k} \quad (5.16)$$

denote the average weight vector. According to this average weight, we also define the following subgradients

$$\bar{\mathbf{g}}_{t,k} \in \partial f_{t,k}(\bar{\mathbf{w}}_t), \quad (5.17)$$

$$\bar{\mathbf{h}}_{t,k} \in \partial r_k(\bar{\mathbf{w}}_t). \quad (5.18)$$

We then consider the average estimation parameter $\bar{\mathbf{w}}_t$ in (5.16) and obtain

the following recursion

$$\begin{aligned}
\bar{\mathbf{w}}_{t+1} &= \frac{1}{K} \sum_{k=1}^K \mathbf{w}_{t+1,k} \\
&= \frac{1}{K} \sum_{k=1}^K \left[\sum_{j=1}^K \mathbf{C}_{jk} \mathbf{w}_{t,j} - \mu_t \mathbf{g}_{t,k} - \mu_t \mathbf{h}_{t+1,k} \right] \\
&= \bar{\mathbf{w}}_t - \frac{\mu_t}{K} \sum_{k=1}^K (\mathbf{g}_{t,k} + \mathbf{h}_{t+1,k}), \tag{5.19}
\end{aligned}$$

where the last line follows from Assumption 3, i.e., $\sum_{k=1}^K \mathbf{C}_{jk} = 1$. By subtracting an arbitrary $\mathbf{w} \in \mathcal{W}$ from both sides of (5.19) and taking the squared norm of both sides, we obtain

$$\begin{aligned}
\|\bar{\mathbf{w}}_{t+1} - \mathbf{w}\|^2 &= \|\bar{\mathbf{w}}_t - \mathbf{w}\|^2 + \frac{\mu_t^2}{K^2} \left\| \sum_{k=1}^K \mathbf{g}_{t,k} + \mathbf{h}_{t+1,k} \right\|^2 \\
&\quad - \frac{2\mu_t}{K} \sum_{k=1}^K \langle \mathbf{g}_{t,k} + \mathbf{h}_{t+1,k}, \bar{\mathbf{w}}_t - \mathbf{w} \rangle. \tag{5.20}
\end{aligned}$$

The second term of (5.20) can be bounded as follows

$$\begin{aligned}
\left\| \sum_{k=1}^K \mathbf{g}_{t,k} + \mathbf{h}_{t+1,k} \right\|^2 &\leq \left(\sum_{k=1}^K \|\mathbf{g}_{t,k}\| + \|\mathbf{h}_{t+1,k}\| \right)^2 \\
&\leq 4A^2 K^2. \tag{5.21}
\end{aligned}$$

We then consider the third term of (5.20) and write the first inner product $-\langle \mathbf{g}_{t,k}, \bar{\mathbf{w}}_t - \mathbf{w} \rangle$ as follows

$$-\langle \mathbf{g}_{t,k}, \bar{\mathbf{w}}_t - \mathbf{w} \rangle = -\langle \mathbf{g}_{t,k}, \bar{\mathbf{w}}_t - \mathbf{w}_{t,k} \rangle - \langle \mathbf{g}_{t,k}, \mathbf{w}_{t,k} - \mathbf{w} \rangle. \tag{5.22}$$

The last term in (5.22) can be upper bounded using the convexity of $f_{t,k}(\cdot)$ as follows

$$\begin{aligned}
\langle \mathbf{g}_{t,k}, \mathbf{w} - \mathbf{w}_{t,k} \rangle &\leq f_{t,k}(\mathbf{w}) - f_{t,k}(\mathbf{w}_{t,k}) - \frac{F}{2} \|\mathbf{w} - \mathbf{w}_{t,k}\|^2 \\
&\leq f_{t,k}(\mathbf{w}) - f_{t,k}(\bar{\mathbf{w}}_t) + \langle \bar{\mathbf{g}}_{t,k}, \bar{\mathbf{w}}_t - \mathbf{w}_{t,k} \rangle \\
&\quad - \frac{F}{2} (\|\mathbf{w} - \mathbf{w}_{t,k}\|^2 + \|\mathbf{w}_{t,k} - \bar{\mathbf{w}}_t\|^2). \tag{5.23}
\end{aligned}$$

Putting (5.23) back in (5.22) and noting that the norm of the gradients are upper bounded by A , we obtain

$$\begin{aligned} -\langle \mathbf{g}_{t,k}, \bar{\mathbf{w}}_t - \mathbf{w} \rangle &\leq f_{t,k}(\mathbf{w}) - f_{t,k}(\bar{\mathbf{w}}_t) + 2A \|\bar{\mathbf{w}}_t - \mathbf{w}_{t,k}\| \\ &\quad - \frac{F}{2} (\|\mathbf{w} - \mathbf{w}_{t,k}\|^2 + \|\mathbf{w}_{t,k} - \bar{\mathbf{w}}_t\|^2), \end{aligned} \quad (5.24)$$

We next consider the second inner product in the third term of (5.20) and denote it as follows

$$\begin{aligned} -\langle \mathbf{h}_{t+1,k}, \bar{\mathbf{w}}_t - \mathbf{w} \rangle &= -\langle \mathbf{h}_{t+1,k}, \sum_{j=1}^K \mathbf{C}_{jk} \mathbf{w}_{t,j} - \mathbf{w}_{t+1,k} \rangle \\ &\quad - \langle \mathbf{h}_{t+1,k}, \bar{\mathbf{w}}_t - \sum_{j=1}^K \mathbf{C}_{jk} \mathbf{w}_{t,j} \rangle - \langle \mathbf{h}_{t+1,k}, \mathbf{w}_{t+1,k} - \mathbf{w} \rangle. \end{aligned} \quad (5.25)$$

Here, we observe that the first term of (5.25) can be upper bounded as

$$\begin{aligned} \langle \mathbf{h}_{t+1,k}, \mathbf{w}_{t+1,k} - \sum_{j=1}^K \mathbf{C}_{jk} \mathbf{w}_{t,j} \rangle &= \langle \mathbf{h}_{t+1,k}, -\mu_t \mathbf{g}_{t,k} - \mu_t \mathbf{h}_{t+1,k} \rangle \\ &\leq 2A^2 \mu_t, \end{aligned} \quad (5.26)$$

where the second line follows from our update rule. We then consider the second term of (5.25) and upper bound it as

$$\begin{aligned} -\langle \mathbf{h}_{t+1,k}, \bar{\mathbf{w}}_t - \sum_{j=1}^K \mathbf{C}_{jk} \mathbf{w}_{t,j} \rangle &= -\langle \mathbf{h}_{t+1,k}, \sum_{j=1}^K \mathbf{C}_{jk} (\bar{\mathbf{w}}_t - \mathbf{w}_{t,j}) \rangle \\ &\leq \sum_{j=1}^K \mathbf{C}_{jk} A \|\bar{\mathbf{w}}_t - \mathbf{w}_{t,j}\|, \end{aligned} \quad (5.27)$$

where the second line follows due to Assumption 3 since $\sum_{j=1}^K \mathbf{C}_{jk} = 1$. Lastly, we consider the third term of (5.25) and upper bound it using the convexity of $r_k(\cdot)$ (with using a similar trick that we have done in (5.23)) and obtain

$$\begin{aligned} \langle \mathbf{h}_{t+1,k}, \mathbf{w} - \mathbf{w}_{t+1,k} \rangle &\leq r_k(\mathbf{w}) - r_k(\bar{\mathbf{w}}_{t+1}) + \langle \bar{\mathbf{h}}_{t+1,k}, \bar{\mathbf{w}}_{t+1} - \mathbf{w}_{t+1,k} \rangle \\ &\quad - \frac{R}{2} (\|\mathbf{w} - \mathbf{w}_{t+1,k}\|^2 + \|\mathbf{w}_{t+1,k} - \bar{\mathbf{w}}_{t+1}\|^2). \end{aligned} \quad (5.28)$$

Putting (5.26), (5.27), and (5.28) back in (5.25), we obtain

$$\begin{aligned}
-\langle \mathbf{h}_{t+1,k}, \bar{\mathbf{w}}_t - \mathbf{w} \rangle &\leq r_k(\mathbf{w}) - r_k(\bar{\mathbf{w}}_{t+1}) + 2A^2\mu_t + A\|\bar{\mathbf{w}}_{t+1} - \mathbf{w}_{t+1,k}\| \\
&\quad + \sum_{j=1}^K C_{jk}A\|\bar{\mathbf{w}}_t - \mathbf{w}_{t,j}\| \\
&\quad - \frac{R}{2}(\|\mathbf{w} - \mathbf{w}_{t+1,k}\|^2 + \|\mathbf{w}_{t+1,k} - \bar{\mathbf{w}}_{t+1}\|^2). \quad (5.29)
\end{aligned}$$

Adding (5.24) and (5.29), and summing from $k = 1$ to K , we obtain the third term of (5.20) as follows

$$\begin{aligned}
&-\sum_{k=1}^K \langle \mathbf{g}_{t,k} + \mathbf{h}_{t+1,k}, \bar{\mathbf{w}}_t - \mathbf{w} \rangle \\
&\leq \sum_{k=1}^K \left\{ f_{t,k}(\mathbf{w}) - f_{t,k}(\bar{\mathbf{w}}_t) + r_k(\mathbf{w}) - r_k(\bar{\mathbf{w}}_{t+1}) + \sum_{j=1}^K C_{jk}A\|\bar{\mathbf{w}}_t - \mathbf{w}_{t,j}\| \right. \\
&\quad + 2A\|\bar{\mathbf{w}}_t - \mathbf{w}_{t,k}\| + A\|\bar{\mathbf{w}}_{t+1} - \mathbf{w}_{t+1,k}\| + 2A^2\mu_t \\
&\quad - \frac{F}{2}(\|\mathbf{w} - \mathbf{w}_{t,k}\|^2 + \|\mathbf{w}_{t,k} - \bar{\mathbf{w}}_t\|^2) \\
&\quad \left. - \frac{R}{2}(\|\mathbf{w} - \mathbf{w}_{t+1,k}\|^2 + \|\mathbf{w}_{t+1,k} - \bar{\mathbf{w}}_{t+1}\|^2) \right\}. \quad (5.30)
\end{aligned}$$

From the definition of the subgradient, we have

$$f_{t,k}(\bar{\mathbf{w}}_t) \geq f_{t,k}(\mathbf{w}_{t,k}) + \langle \mathbf{g}_{t,k}, \mathbf{w}_{t,k} - \bar{\mathbf{w}}_t \rangle,$$

and

$$r_k(\bar{\mathbf{w}}_{t+1}) \geq r_k(\mathbf{w}_{t+1,k}) + \langle \mathbf{h}_{t+1,k}, \mathbf{w}_{t+1,k} - \bar{\mathbf{w}}_{t+1} \rangle.$$

Using these inequalities, we can write (5.30) as follows

$$\begin{aligned}
&-\sum_{k=1}^K \langle \mathbf{g}_{t,k} + \mathbf{h}_{t+1,k}, \bar{\mathbf{w}}_t - \mathbf{w} \rangle \\
&\leq \sum_{k=1}^K \left\{ f_{t,k}(\mathbf{w}) - f_{t,k}(\mathbf{w}_{t,k}) + r_k(\mathbf{w}) - r_k(\mathbf{w}_{t+1,k}) + \sum_{j=1}^K C_{jk}A\|\bar{\mathbf{w}}_t - \mathbf{w}_{t,j}\| \right. \\
&\quad + 3A\|\bar{\mathbf{w}}_t - \mathbf{w}_{t,k}\| + 2A\|\bar{\mathbf{w}}_{t+1} - \mathbf{w}_{t+1,k}\| + 2A^2\mu_t \\
&\quad - \frac{F}{2}(\|\mathbf{w} - \mathbf{w}_{t,k}\|^2 + \|\mathbf{w}_{t,k} - \bar{\mathbf{w}}_t\|^2) \\
&\quad \left. - \frac{R}{2}(\|\mathbf{w} - \mathbf{w}_{t+1,k}\|^2 + \|\mathbf{w}_{t+1,k} - \bar{\mathbf{w}}_{t+1}\|^2) \right\}. \quad (5.31)
\end{aligned}$$

To simplify (5.31), we need the following lemma, whose proof is given in Appendix A.7

Lemma 5.3 *Under Assumptions 2-4, the deviation of the parameter vector of each agent from the average parameter vector is upper bounded as follows*

$$\|\bar{\mathbf{w}}_t - \mathbf{w}_{t,k}\| \leq 2A\sqrt{K} \sum_{s=1}^{t-1} \mu_{t-s} \sigma^{s-1}, \quad (5.32)$$

where σ is the second largest singular value of the communication matrix $\mathbf{C}$.

Considering the definitions of the cost functions, noting that

$$\frac{1}{K} \sum_{k=1}^K (\|\mathbf{w} - \mathbf{w}_{t,k}\|^2 + \|\mathbf{w}_{t,k} - \bar{\mathbf{w}}_t\|^2) \leq \|\mathbf{w} - \bar{\mathbf{w}}_t\|^2, \quad (5.33)$$

from the Jensen's inequality, and using Lemma 1, we can rewrite (5.31) as follows

$$\begin{aligned} & - \sum_{k=1}^K \langle \mathbf{g}_{t,k} + \mathbf{h}_{t+1,k}, \bar{\mathbf{w}}_t - \mathbf{w} \rangle \\ & \leq f_t(\mathbf{w}) - f_t(\mathbf{w}_{t,k}) + r(\mathbf{w}) - r(\mathbf{w}_{t+1,k}) + 8A^2K\sqrt{K} \sum_{s=1}^{t-1} \mu_{t-s} \sigma^{s-1} \\ & \quad + 2A^2K\mu_t + 4A^2K\sqrt{K} \sum_{s=1}^t \mu_{t-s+1} \sigma^{s-1} \\ & \quad - \frac{KF}{2} (\|\mathbf{w} - \bar{\mathbf{w}}_t\|^2) - \frac{KR}{2} (\|\mathbf{w} - \bar{\mathbf{w}}_{t+1}\|^2). \end{aligned} \quad (5.34)$$

In order to obtain the desired upper bound, we put (5.21) and (5.34) back in (5.20) and summing from $t = 1$ to T , we obtain

$$\begin{aligned} & \sum_{t=1}^T f_t(\mathbf{w}_{t,k}) - f_t(\mathbf{w}) + r(\mathbf{w}_{t+1,k}) - r(\mathbf{w}) \\ & \leq \sum_{t=1}^T \left\{ \frac{K}{2} \left(\frac{1}{\mu_t} - F \right) \|\mathbf{w} - \bar{\mathbf{w}}_t\|^2 - \frac{K}{2} \left(\frac{1}{\mu_t} + R \right) \|\mathbf{w} - \bar{\mathbf{w}}_{t+1}\|^2 \right. \\ & \quad \left. + 8A^2K\sqrt{K} \sum_{s=1}^{t-1} \mu_{t-s} \sigma^{s-1} + 4A^2K\mu_t + 4A^2K\sqrt{K} \sum_{s=1}^t \mu_{t-s+1} \sigma^{s-1} \right\}. \end{aligned} \quad (5.35)$$

Here, we make the make the following observation

$$\begin{aligned} \sum_{t=1}^T \sum_{s=1}^{t-1} \mu_{t-s} \sigma^{s-1} &= \sum_{s=1}^T \sigma^{s-1} \sum_{t=s+1}^T \mu_{t-s} \\ &\leq \frac{1}{1-\sigma} \sum_{t=1}^T \mu_t, \end{aligned} \quad (5.36)$$

which yields (by adding $r(\bar{\mathbf{w}}_1)$ to the both sides of the inequality) the following regret bound

$$\begin{aligned} \sum_{t=1}^T f_t(\mathbf{w}_{t,k}) - f_t(\mathbf{w}) + r(\mathbf{w}_{t,k}) - r(\mathbf{w}) \\ \leq \frac{K}{2} \left(\frac{1}{\mu_1} - F \right) \|\mathbf{w} - \bar{\mathbf{w}}_1\|^2 + \frac{K}{2} \sum_{t=2}^T \left(\frac{1}{\mu_t} - \frac{1}{\mu_{t-1}} - F - R \right) \|\mathbf{w} - \bar{\mathbf{w}}_t\|^2 \\ + AD + 4A^2K \left(1 + \frac{3\sqrt{K}}{1-\sigma} \right) \sum_{t=1}^T \mu_t. \end{aligned} \quad (5.37)$$

If we have $F = R = 0$, i.e., if neither of the cost functions are strongly convex, we obtain the following bound

$$\begin{aligned} \sum_{t=1}^T f_t(\mathbf{w}_{t,k}) - f_t(\mathbf{w}) + r(\mathbf{w}_{t,k}) - r(\mathbf{w}) \\ \leq \frac{D^2K}{2} \left[\frac{1}{\mu_1} + \sum_{t=2}^T \left(\frac{1}{\mu_t} - \frac{1}{\mu_{t-1}} \right) \right] + AD + 4A^2K \left(1 + \frac{3\sqrt{K}}{1-\sigma} \right) \sum_{t=1}^T \mu_t \\ \leq AD + \frac{D^2K}{2\mu_T} + 4A^2K \left(1 + \frac{3\sqrt{K}}{1-\sigma} \right) \sum_{t=1}^T \mu_t. \end{aligned} \quad (5.38)$$

Defining the function $\beta(K, \sigma) \triangleq 1 + \frac{3\sqrt{K}}{1-\sigma}$ and letting $\mu_t = \frac{D}{4A\sqrt{\beta(K, \sigma)t}}$, we can rewrite the upper bound in (5.38) as follows

$$\sum_{t=1}^T f_t(\mathbf{w}_{t,k}) - f_t(\mathbf{w}) + r(\mathbf{w}_{t,k}) - r(\mathbf{w}) \leq AD \left(1 + 4K\sqrt{\beta(K, \sigma)} \right), \quad (5.39)$$

since

$$\sum_{t=1}^T \frac{D}{4A\sqrt{\beta(K, \sigma)t}} \leq \frac{D\sqrt{T}}{2A\sqrt{\beta(K, \sigma)}}. \quad (5.40)$$

This yields the desired upper bound in (5.14).

If we have $S = F + R > 0$, where we do not require both F and R to be strictly positive, instead either $F > 0$ or $R > 0$ is sufficient, then we obtain the following bound

$$\begin{aligned} & \sum_{t=1}^T f_t(\mathbf{w}_{t,k}) - f_t(\mathbf{w}) + r(\mathbf{w}_{t,k}) - r(\mathbf{w}) \\ & \leq \frac{D^2 K}{2} \left[\frac{1}{\mu_1} + \sum_{t=2}^T \left(\frac{1}{\mu_t} - \frac{1}{\mu_{t-1}} - S \right) \right] + AD + 4A^2 K \left(1 + \frac{3\sqrt{K}}{1-\sigma} \right) \sum_{t=1}^T \mu_t. \end{aligned} \quad (5.41)$$

Picking $\mu_t = \frac{1}{St}$, we obtain $\frac{1}{\mu_t} - \frac{1}{\mu_{t-1}} - S = 0$, which yields

$$\begin{aligned} & \sum_{t=1}^T f_t(\mathbf{w}_{t,k}) - f_t(\mathbf{w}) + r(\mathbf{w}_{t,k}) - r(\mathbf{w}) \\ & \leq AD + \frac{SD^2 K}{2} + 4A^2 K \left(1 + \frac{3\sqrt{K}}{1-\sigma} \right) \frac{1 + \log T}{S}. \end{aligned} \quad (5.42)$$

since

$$\sum_{t=1}^T \frac{1}{St} \leq \frac{1 + \log T}{S}. \quad (5.43)$$

This yields the desired upper bound in (5.15) and concludes the proof of Theorem 5.1 and Corollary 5.2. $\square$

5.5 Simulations

In this section, we evaluate the performance of DSS-ELM algorithm (presented in Algorithm 8) for various multi-agent networks with different regularization factors. We illustrate the superior performance of our algorithm for different regression applications involving both real and synthetic data.

5.5.1 Stationary Scenario

In this section, we consider a multi-agent network of size $K = 10$, where each agent observes a desired signal through a linear model. In particular, the k th

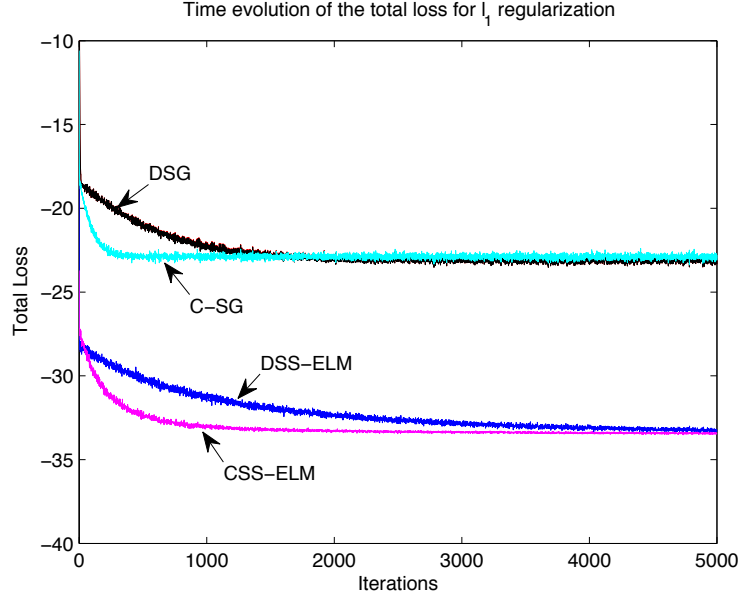


Figure 5.2: Comparison of the algorithms for the linear regression model in (5.44) with mean square error loss and ℓ_1 norm regularization.

agent observes

$$y_{t,k} = \mathbf{w}^T \mathbf{x}_{t,k} + n_{t,k}, \quad (5.44)$$

where $\mathbf{w} \in \mathbb{R}^{10}$ is chosen from a normal distribution while 50% of the entries are randomly set to zero, $\mathbf{x}_{t,k} \in \mathbb{R}^{10}$ is a realization of a spatially and temporally independent normal random vector process, and $n_{t,k} \in \mathbb{R}$ is a zero-mean white Gaussian noise with variance $\sigma_{n_k}^2 = 10^{-3}$. We normalize the input and output attributes to the interval $[0, 1]$ as recommended in [33].

The empirical loss function is chosen as the square error loss, i.e., $f_{t,k} = (y_{t,k} - \hat{y}_{t,k})^2$, whereas the regularization loss function is chosen as ℓ_1 -norm regularization in Figure 5.2 and ℓ_2^2 -norm regularization in Figure 5.2, where we set $\lambda = 10^{-4}$. We point out that for these regularization factors, the update rule (i.e., the 6th line in Algorithm 8) is given by

$$\mathbf{w}_{t+1,k} = \begin{cases} \text{sign}(\mathbf{w}'_{t+1,k}) \odot [|\mathbf{w}'_{t+1,k}| - \mu_t \lambda]_+ & , \text{ for } \ell_1 \\ \frac{1}{1+\mu_t \lambda} \mathbf{w}'_{t+1,k} & , \text{ for } \ell_2^2 \end{cases} \quad (5.45)$$

In Figs. 5.2 and 5.3, we compare the performance of the DSS-ELM algorithm

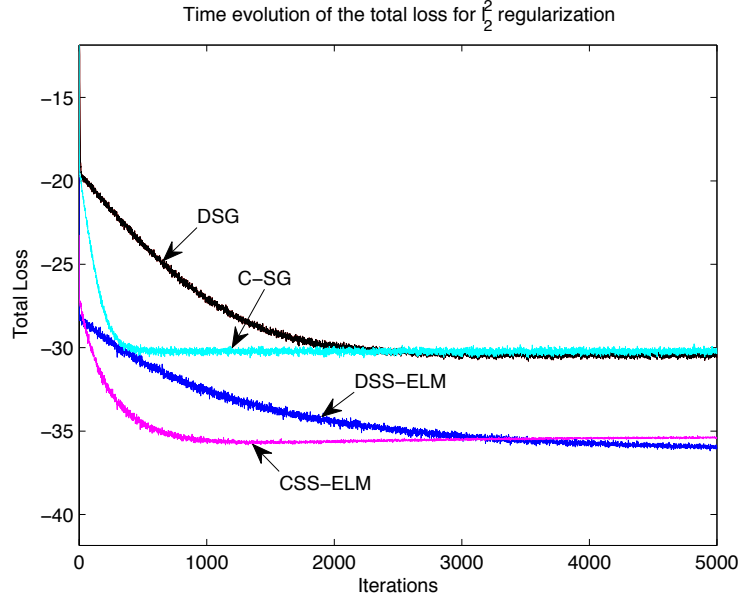


Figure 5.3: Comparison of the algorithms for the linear regression model in (5.44) with mean square error loss and ℓ_2^2 norm regularization.

with the distributed stochastic gradient (D-SG) algorithm [39], and their centralized variants, i.e., the centralized stochastic gradient (C-SG) and the centralized sequential splitting ELM (CSS-ELM) algorithms. In the distributed multi-agent framework, each agent exchanges information with its neighboring agents over a communication network. We have generated this network randomly. Agents employ the uniform combination rule [85], where $H_{j,k} = 1/\pi_k$ with π_k representing the number of neighboring agents of the k th agent. In the centralized framework, all the data observed by every agent is collected at a central processing unit and processed through conventional processing techniques. In the experiments, the step sizes of the D-SG and C-SG algorithms are set to 0.05, whereas it is 0.1 for the DSS-ELM and CSS-ELM algorithms (These learning rates are chosen for a fair performance comparison since the number of hidden nodes is approximately twice of the length of the linear parameter vector). We set the number of hidden neurons to 24 in order to achieve a reasonable steady-state loss and use sigmoidal additive activation function. We have performed 250 independent trials and presented the averaged results in Figs. 5.2 and 5.3. The offset and bias parameters of the hidden neurons are chosen from a uniform distribution $\mathcal{U}[-1, 1]$.

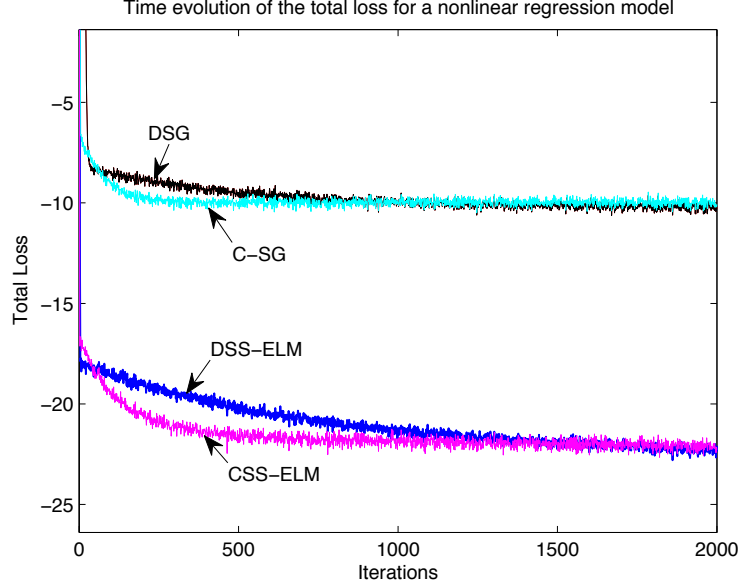


Figure 5.4: Comparison of the algorithms for the nonlinear regression model in (5.46) with mean square error loss and ℓ_2^2 norm regularization.

In Figs. 5.2 and 5.3, we observe that the distributed processing algorithms achieve a comparable performance with their corresponding centralized versions. We also point out that the DSS-ELM algorithm significantly outperforms the D-SG algorithm for both ℓ_1 and ℓ_2^2 regularization costs. For the ℓ_1 -regularization case, the performance of the D-SG significantly deteriorates with respect to the ℓ_2^2 -regularization case. On the other hand, the performance of the proposed DSS-ELM algorithm is not significantly affected from these regularization factors since we use the forward-backward splitting method. Furthermore, since the underlying parameter vector $\mathbf{w}$ is sparse, the linear models fail to achieve low error rates, whereas the proposed SLFN-based algorithm yields a considerably smaller error.

We next consider the following nonlinear regression model

$$y_{t,k} = \text{sign}(\mathbf{w}^T \mathbf{x}_{t,k}), \quad (5.46)$$

where $\mathbf{w}$ and $\mathbf{x}_{t,k}$ is chosen as in the previous setup and the labels of 10% of the desired signals, i.e., $y_{t,k}$'s, are flipped in order to introduce label noise to the model. In Figure 5.4, we present the performances of the DSS-ELM, CSS-ELM, D-SG, and C-SG algorithms for the ℓ_2^2 -regularization loss. We observe

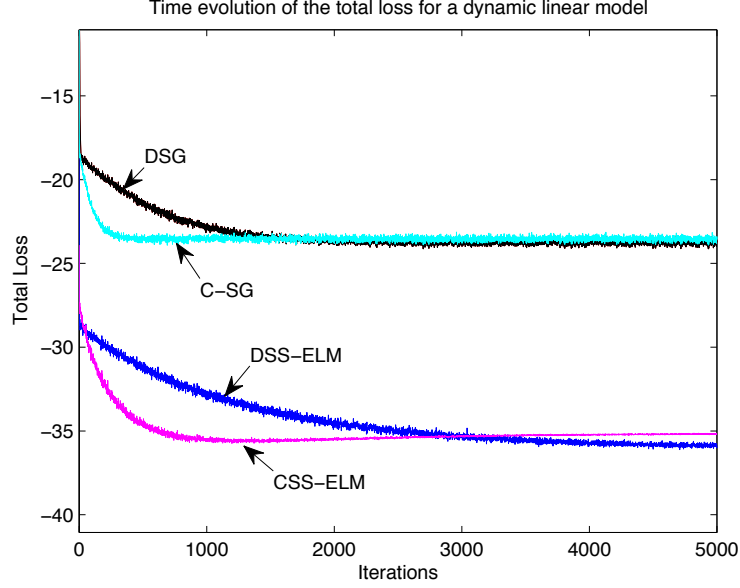


Figure 5.5: Comparison of the algorithms for the dynamic linear regression model in (5.44) and (5.47) with mean square error loss and ℓ_2^2 norm regularization.

that the proposed DSS-ELM algorithm significantly outperforms the linear gradient descent based algorithms. This follows since the SLFN-based structures can elegantly capture the nonlinear relationships between the desired data and regressor vectors, whereas linear models cannot learn this relationships satisfactorily. Furthermore, the proposed DSS-ELM algorithm achieves the performance of the centralized CSS-ELM algorithm, which verifies the asymptotical convergence results presented in Theorem 5.1 and Corollary 5.2.

5.5.2 Nonstationary Scenario

In this example, we evaluate the performance of the DSS-ELM, CSS-ELM, D-SG, and C-SG algorithms in nonstationary environments. We consider the regression setup in (5.44), where the unknown parameter vector $\mathbf{w}$ evolves in time through a random walk model (we emphasize that this setup is extensively used in the machine learning and signal processing literatures [13]) as follows

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \mathbf{q}_t, \quad (5.47)$$

where $\mathbf{q}_t \in \mathbb{R}^{10}$ is a realization of i.i.d. zero-mean Gaussian random vector process with auto-covariance matrix $\mathbf{C}_q = 10^{-6}\mathbf{I}_{10}$.

In Figure 5.5, we present the time evolution of the ℓ_2^2 -regularized square error loss function of the proposed algorithms. Comparing this figure with the stationary setup in Figure 5.3, we observe that our algorithm suffers approximately 10dB performance loss, whereas the linear learning models suffer approximately 20dB performance loss. Therefore, the proposed DSS-ELM algorithm is considerably more robust against the nonstationarity compared to the linear learning algorithms. This follows from the efficient SLFN-based nonlinear formulation of our regression model.

5.5.3 Real Data Sets

In this section, we illustrate the performance of our algorithm for the benchmark data sets, i.e., California housing, protein tertiary structure, and household electrical power consumption [61, 86]. In these data sets, we compare the proposed DSS-ELM algorithm with the OS-ELM algorithm of [33]. Throughout this section, we use the sigmoidal additive activation function to generate SLFNs and we use squared error loss function (with different regularization terms for the DSS-ELM algorithm). We normalize the input and output attributes of the data set to the interval $[0, 1]$ as recommended in [33]. We use 100 data samples (which is more than the recommended value in [33]) at the initialization phase of the OS-ELM algorithm. For the DSS-ELM algorithm, we randomly generate a distributed multi-agent network with the specified size, and we set the step size $\mu_t = 10^{-2}$ and the regularization constant $\lambda = 10^{-4}$.

- **California housing (CH)** data set involves the prediction of the median housing value based on the information collected on several parameters through all the block groups in California from the 1990 census. The number of hidden nodes are set to 90 for the DSS-ELM algorithm and 50 for the OS-ELM algorithm. For the DSS-ELM algorithm, the number of agents are

Data Set	# attributes	# training data	# test data
CH	8	12000	8620
PTS	9	36000	9730
HPC	7	450000	46067

Table 5.1: Specifications of Benchmark Data Sets.

set to 4 for this experiment.

- **Protein tertiary structure (PTS)** data set considers the learning the size of the residue based on several other parameters, e.g., the molecular mass weighted exposed area. The number of hidden nodes are set to 50 for the DSS-ELM algorithm and 70 for the OS-ELM algorithm. For the DSS-ELM algorithm, the number of agents are set to 8 for this experiment.
- **Household electrical power consumption (HPC)** data set contains 2075259 measurements of individual households gathered between December 2006 and November 2010, where the aim is to predict the global active power. In our experiments, we used the initial $\sim 25\%$ of the data corresponding to the data between December 2006 and November 2007. The number of hidden nodes are set to 70 for the DSS-ELM algorithm and 50 for the OS-ELM algorithm. For the DSS-ELM algorithm, the number of agents are set to 10 for this experiment.

We emphasize that the number of hidden nodes for the DSS-ELM and OS-ELM algorithms are carefully chosen to minimize the performance of the corresponding algorithms for a fair comparison between these methods. The number of agents are chosen according to the data length to prevent undertraining issues. As an example, for short data sequences, it may not be possible to train SLFNs for huge number of agents since each agent observes only a portion of the underlying data.

For each data set, the number of attributes as well as the length of the data that is used for training and testing can be found in Table 5.1. We randomly choose the training and testing data for each trial and evaluate the average performance of the proposed algorithms over 50 independent trials. Note that the OS-ELM algorithm operates on a single centralized agent, whereas the DSS-ELM

Data Set	Algorithms	Time (s)	RMSE (training)	RMSE (testing)
CH	DSS-ELM (ℓ_1)	0.0821	0.1781	0.1785
	DSS-ELM (ℓ_2^2 -)	0.0759	0.1765	0.1766
	OS-ELM	1.4978	0.1303	0.1310
PTS	DSS-ELM (ℓ_1)	0.1190	0.0636	0.0637
	DSS-ELM (ℓ_2^2)	0.1330	0.0637	0.0637
	OS-ELM	5.6141	0.2306	0.2315
HPC	DSS-ELM (ℓ_1)	1.3740	0.0170	0.0170
	DSS-ELM (ℓ_2^2)	1.3291	0.0099	0.0100
	OS-ELM	124.7953	0.0326	0.0332

Table 5.2: Comparison of the RMSE performance of the algorithms.

algorithm works for multi-agent systems. Therefore, for the OS-ELM algorithm, the centralized agent is trained using all the training data, whereas for the DSS-ELM algorithm, each agent is trained using only a randomly chosen portion of the training data. As an example, for California housing data set, we have $K = 4$ agents and a training data of length 12000. In this scenario, the OS-ELM algorithm is trained using the entire training data, whereas for the DSS-ELM algorithm this entire training data is randomly separated into 4 different training data of length 3000 and assigned to different agents.

Table 5.2 illustrates the training times and the RMSE performances for training as well as testing for the DSS-ELM and OS-ELM algorithms. This table shows that the DSS-ELM algorithm can train the SLFNs at each agent significantly faster than the OS-ELM algorithm. As an example, the training time for the DSS-ELM algorithm (that employs ℓ_2^2 -regularization) is approximately 1.33 seconds at each agent, where we have 10 agents in the distributed multi-agent network. On the other hand, for the OS-ELM algorithm the training time is approximately 124.80 seconds. Therefore, our algorithm presents an order of magnitude improvement in terms of the training time. Furthermore, even though the computational complexity of the proposed DSS-ELM algorithm is much lower than the OS-ELM algorithm, it yields a significantly better RMSE performance than the OS-ELM algorithm for protein tertiary and household power consumption data sets. This follows since the DSS-ELM algorithm can adapt nonstationary environments in a faster manner compared to the OS-ELM algorithm. The

# agents	# samples/agent	Time (s)	RMSE (training)	RMSE (testing)
4	9000	0.2623	0.0620	0.0621
5	7200	0.2109	0.0624	0.0629
6	6000	0.1764	0.0630	0.0633
8	4500	0.1330	0.0637	0.0637
9	4000	0.1202	0.0639	0.0641
10	3600	0.1034	0.0644	0.0644
12	3000	0.0847	0.0657	0.0658

Table 5.3: RMSE performance of the DSS-ELM algorithm for different network sizes for the protein tertiary data with ℓ_2^2 -regularization.

proposed DSS-ELM algorithm uses a gradient descent based learning algorithm that can effectively track nonstationarity (as also illustrated in Section 5.5.2), whereas the OS-ELM algorithm uses a recursive least squares based approach, which is less resilient to nonstationarity.

Finally, in Table 5.3, we analyze the effects of the network size (i.e., the number of agents) to the performance of the overall system. As can be seen from the table, as the number of agent increases, the training time of the SLFNs at each agent decreases (as the number of training data per agent decreases). On the other hand, the RMSE at each agent increases as the number of agents increases, which is expected from the convergence upper bounds presented in Theorem 5.1 and Corollary 5.2.

Chapter 6

Conclusion

We study nonlinear learning in a deterministic framework. In Chapter 2, 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. In Chapter 3, we introduce a tree based algorithm that sequentially increases its nonlinear modeling power and achieves the performance of the optimal twice differentiable function as well as the performance of the best piecewise linear model defined on the incremental decision tree. Furthermore, this performance is achieved only with a computational complexity logarithmic in the data length n , i.e., $O(\log(n))$ (under regularity conditions). We demonstrate the superior performance of the introduced algorithm over a series of well-known benchmark applications in the regression literature. 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.

In Chapter 4, we introduce a sequential FS prediction algorithm for real valued sequences, where we construct hierarchical structures to define states. Instead of directly using the equivalence classes at the highest level of hierarchy, which can result a prohibitively large number of states even for moderate hierarchy depths, we define hierarchical equivalence classes by recursively tying certain states to avoid undertraining problems. With this hierarchical equivalence class definitions, we construct a doubly exponential number (in the hierarchy depth) of FS predictors. By using the EG algorithm, we show that we can sequentially achieve the performance of the optimal combination of all FS predictors that can be defined on this hierarchical structure with a computational complexity only linear in the length of the hierarchy depth.

In Chapter 5, we study sequential training of SLFNs over distributed multi-agent networks. The aim of each agent is to minimize arbitrary cost functions by training an individual SLFN-based regressor (or classifier) using the data that is revealed only to this particular agent. On the other hand, the goal of the multi-agent network is to train these individual SLFNs at each agent as well as the optimal centralized batch SLFN that has access to all the data revealed to all agents. As the first time in the literature, we solve this problem by introducing a distributed sequential ELM-based algorithm. In particular, we show that our algorithm guarantees that the performances of these individual SLFNs at each agent are asymptotically the same as the performance of the optimal centralized batch SLFN that is trained by an ELM-based method. The proposed algorithm works in a truly sequential manner such that it can be used without any training or initialization phase. Furthermore, the computational complexity of the algorithm is only linear in the number of hidden nodes, hence it is extremely appealing for applications involving big data.

Bibliography

- [1] A. C. Singer, G. W. Wornell, and A. V. Oppenheim, “Nonlinear autoregressive modeling and estimation in the presence of noise,” *Digital Signal Processing*, vol. 4, no. 4, pp. 207–221, 1994.
- [2] O. J. J. Michel, A. O. Hero, and A.-E. Badel, “Tree-structured nonlinear signal modeling and prediction,” *IEEE Transactions on Signal Processing*, vol. 47, no. 11, pp. 3027–3041, 1999.
- [3] R. J. Drost and A. C. Singer, “Constrained complexity generalized context-tree algorithms,” in *IEEE/SP 14th Workshop on Statistical Signal Processing*, pp. 131–135, 2007.
- [4] S. S. Kozat, A. C. Singer, and G. C. Zeitler, “Universal piecewise linear prediction via context trees,” *IEEE Transactions on Signal Processing*, vol. 55, no. 7, pp. 3730–3745, 2007.
- [5] M. Schetzen, *The Volterra and Wiener Theories of Nonlinear Systems*. NJ: John Wiley & Sons, 1980.
- [6] M. Scarpiniti, D. Comminiello, R. Parisi, and A. Uncini, “Nonlinear spline adaptive filtering,” *Signal Processing*, vol. 93, no. 4, pp. 772 – 783, 2013.
- [7] A. Carini and G. L. Sicuranza, “Fourier nonlinear filters,” *Signal Processing*, vol. 94, no. 0, pp. 183 – 194, 2014.
- [8] D. P. Helmbold and R. E. Schapire, “Predicting nearly as well as the best pruning of a decision tree,” *Machine Learning*, vol. 27, no. 1, pp. 51–68, 1997.

- [9] R. J. Sela and J. S. Simonoff, “RE-EM trees: a data mining approach for longitudinal and clustered data,” *Machine Learning*, vol. 86, no. 2, pp. 169–207, 2012.
- [10] Z. Zheng, “Constructing X-of-N attributes for decision tree learning,” *Machine Learning*, vol. 40, no. 1, pp. 35–75, 2000.
- [11] A. C. Singer and M. Feder, “Universal linear prediction by model order weighting,” *IEEE Transactions on Signal Processing*, vol. 47, no. 10, pp. 2685–2699, 1999.
- [12] T. Moon and T. Weissman, “Universal FIR MMSE filtering,” *IEEE Transactions on Signal Processing*, vol. 57, no. 3, pp. 1068–1083, 2009.
- [13] A. H. Sayed, *Fundamentals of Adaptive Filtering*. NJ: John Wiley & Sons, 2003.
- [14] V. H. Nascimento and A. H. Sayed, “On the learning mechanism of adaptive filters,” *IEEE Transactions on Signal Processing*, vol. 48, no. 6, pp. 1609–1625, 2000.
- [15] A. H. Sayed, S.-Y. Tu, J. Chen, X. Zhao, and Z. J. Towfic, “Diffusion strategies for adaptation and learning over networks: an examination of distributed strategies and network behavior,” *IEEE Signal Processing Magazine*, vol. 30, pp. 155–171, May 2013.
- [16] J. Arenas-Garcia, A. R. Figueiras-Vidal, and A. H. Sayed, “Mean-square performance of a convex combination of two adaptive filters,” *IEEE Transactions on Signal Processing*, vol. 54, no. 3, pp. 1078–1090, 2006.
- [17] S. Dasgupta and Y. Freund, “Random projection trees for vector quantization,” *IEEE Transactions on Information Theory*, vol. 55, no. 7, pp. 3229–3242, 2009.
- [18] L. Devroye, T. Linder, and G. Lugosi, “Nonparametric estimation and classification using radial basis function nets and empirical risk minimization,” *IEEE Transactions on Neural Networks*, vol. 7, no. 2, pp. 475–487, 1996.

- [19] A. Krzyzak and T. Linder, “Radial basis function networks and complexity regularization in function learning,” *IEEE Transactions on Neural Networks*, vol. 9, no. 2, pp. 247–256, 1998.
- [20] N. D. Vanli and S. S. Kozat, “A comprehensive approach to universal piecewise nonlinear regression based on trees,” *IEEE Transactions on Signal Processing*, vol. 62, pp. 5471–5486, Oct 2014.
- [21] N. D. Vanli and S. S. Kozat, “A unified approach to universal prediction: Generalized upper and lower bounds,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 26, pp. 646–651, March 2015.
- [22] Y. Yilmaz and S. S. Kozat, “Competitive randomized nonlinear prediction under additive noise,” *IEEE Signal Processing Letters*, vol. 17, pp. 335–339, April 2010.
- [23] G. David and A. Averbuch, “Hierarchical data organization, clustering and denoising via localized diffusion folders,” *Applied and Computational Harmonic Analysis*, vol. 33, no. 1, pp. 1 – 23, 2012.
- [24] A. B. Lee, B. Nadler, and L. Wasserman, “Treelets – an adaptive multiscale basis for sparse unordered data,” *The Annals of Applied Statistics*, vol. 2, no. 2, pp. 435–471, 2008.
- [25] J. L. Bentley, “Multidimensional binary search trees in database applications,” *IEEE Transactions on Software Engineering*, vol. SE-5, no. 4, pp. 333–340, 1979.
- [26] A. C. Singer, S. S. Kozat, and M. Feder, “Universal linear least squares prediction: upper and lower bounds,” *IEEE Transactions on Information Theory*, vol. 48, no. 8, pp. 2354–2362, 2002.
- [27] E. Takimoto, A. Maruoka, and V. Vovk, “Predicting nearly as well as the best pruning of a decision tree through dyanamic programming scheme,” *Theoretical Computer Science*, vol. 261, pp. 179–209, 2001.

- [28] E. Takimoto and M. K. Warmuth, “Predicting nearly as well as the best pruning of a planar decision graph,” *Theoretical Computer Science*, vol. 288, pp. 217–235, 2002.
- [29] J. Arenas-Garcia, V. Gomez-Verdejo, and A. R. Figueiras-Vidal, “New algorithms for improved adaptive convex combination of LMS transversal filters,” *IEEE Transactions on Instrumentation and Measurement*, vol. 54, no. 6, pp. 2239–2249, 2005.
- [30] S. S. Kozat, A. T. Erdogan, A. C. Singer, and A. H. Sayed, “Steady state MSE performance analysis of mixture approaches to adaptive filtering,” *IEEE Transactions on Signal Processing*, vol. 58, pp. 4050–4063, August 2010.
- [31] A. H. Sayed, *Fundamentals of Adaptive Filtering*. John Wiley and Sons, 2003.
- [32] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, “Extreme learning machine: Theory and applications,” *Neurocomputing*, vol. 70, no. 13, pp. 489 – 501, 2006.
- [33] N.-Y. Liang, G.-B. Huang, P. Saratchandran, and N. Sundararajan, “A fast and accurate online sequential learning algorithm for feedforward networks,” *IEEE Transactions on Neural Networks*, vol. 17, no. 6, pp. 1411–1423, 2006.
- [34] N.-Y. Liang, G.-B. Huang, P. Saratchandran, and N. Sundararajan, “A fast and accurate online sequential learning algorithm for feedforward networks,” *IEEE Transactions on Neural Networks*, vol. 17, no. 6, pp. 1411–1423, 2006.
- [35] G.-B. Huang and L. Chen, “Convex incremental extreme learning machine,” *Neurocomputing*, vol. 70, no. 1618, pp. 3056 – 3062, 2007.
- [36] S. Balasundaram, D. Gupta, and Kapil, “1-norm extreme learning machine for regression and multiclass classification using newton method,” *Neurocomputing*, vol. 128, no. 0, pp. 4 – 14, 2014.
- [37] X. Wu, X. Zhu, G.-Q. Wu, and W. Ding, “Data mining with big data,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 1, pp. 97–107, 2014.

- [38] R. W. Parks, D. L. Long, D. S. Levine, D. J. Crockett, E. G. McGeer, P. L. McGeer, I. E. Dalton, R. F. Zec, R. E. Becker, K. L. Coburn, G. Siler, M. E. Nelson, and J. M. Bower, “Parallel distributed processing and neural networks: Origins, methodology and cognitive functions,” *International Journal of Neuroscience*, vol. 60, no. 3-4, pp. 195–214, 1991.
- [39] A. H. Sayed, “Adaptive networks,” *Proceedings of the IEEE*, vol. 102, pp. 460–497, April 2014.
- [40] J.-J. Xiao, A. Ribeiro, Z.-Q. Luo, and G. B. Giannakis, “Distributed compression-estimation using wireless sensor networks,” *IEEE Signal Processing Magazine*, vol. 23, pp. 27–41, July 2006.
- [41] S. S. Ram, A. Nedic, and V. V. Veeravalli, “Distributed stochastic subgradient projection algorithms for convex optimization,” *Journal of Optimization Theory and Applications*, vol. 147, no. 3, pp. 516–545, 2010.
- [42] A. Nedic and A. Ozdaglar, “Distributed subgradient methods for multi-agent optimization,” *IEEE Transactions on Automatic Control*, vol. 54, pp. 48–61, Jan 2009.
- [43] A. Nedic, A. Ozdaglar, and P. A. Parrilo, “Constrained consensus and optimization in multi-agent networks,” *IEEE Transactions on Automatic Control*, vol. 55, pp. 922–938, April 2010.
- [44] N. D. Vanli, M. O. Sayin, S. Ergut, and S. S. Kozat, “Piecewise nonlinear regression via decision adaptive trees,” in *2014 Proceedings of the 22nd European Signal Processing Conference (EUSIPCO)*, pp. 1188–1192, Sept 2014.
- [45] N. D. Vanli and S. S. Kozat, “Sequential nonlinear regression via context trees,” in *2014 22nd Signal Processing and Communications Applications Conference (SIU)*, pp. 1865–1868, April 2014.
- [46] N. D. Vanli, M. O. Sayin, S. Ergut, and S. S. Kozat, “Comprehensive lower bounds on sequential prediction,” in *2014 Proceedings of the 22nd European Signal Processing Conference (EUSIPCO)*, pp. 1193–1196, Sept 2014.

- [47] H. Chen, J. Peng, Y. Zhou, L. Li, and Z. Pan, “Extreme learning machine for ranking: Generalization analysis and applications,” *Neural Networks*, vol. 53, no. 0, pp. 119 – 126, 2014.
- [48] F. M. J. Willems, Y. M. Shtarkov, and T. J. Tjalkens, “The context-tree weighting method: basic properties,” *IEEE Transactions on Information Theory*, vol. 41, no. 3, pp. 653–664, 1995.
- [49] D. W. Hosmer, S. Lemeshow, and R. X. Sturdivant, *Applied Logistic Regression*. NJ: John Wiley & Sons, 2013.
- [50] E. Hazan, A. Agarwal, and S. Kale, “Logarithmic regret algorithms for online convex optimization,” *Machine Learning*, vol. 69, no. 2-3, pp. 169–192, 2007.
- [51] E. Eweda, “Comparison of RLS, LMS, and sign algorithms for tracking randomly time-varying channels,” *IEEE Transactions on Signal Processing*, vol. 42, no. 11, pp. 2937–2944, 1994.
- [52] J. Arenas-Garcia, M. Martinez-Ramon, V. Gomez-Verdejo, and A. R. Figueiras-Vidal, “Multiple plant identifier via adaptive LMS convex combination,” in *2003 IEEE International Symposium on Intelligent Signal Processing*, pp. 137–142, 2003.
- [53] C. E. Rasmussen, R. M. Neal, G. Hinton, D. Camp, M. Revow, Z. Ghahramani, R. Kustra, and R. Tibshirani, “Delve data sets.” <http://www.cs.toronto.edu/~delve/data/datasets.html>, 2015.
- [54] J. Alcalá-Fdez, A. Fernández, J. Luengo, J. Derrac, S. García, L. Sánchez, and F. Herrera, “KEEL data-mining software tool: Data set repository, integration of algorithms and experimental analysis framework,” *Journal of Multiple-Valued Logic and Soft Computing*, vol. 17, no. 2-3, pp. 255–287, 2015.
- [55] L. Torgo, “Regression data sets.” <http://www.dcc.fc.up.pt/~ltorgo/Regression/DataSets.html>, 2015.
- [56] E. N. Lorenz, “Deterministic nonperiodic flow,” *Journal of the Atmospheric Sciences*, vol. 20, no. 2, pp. 130–141, 1963.

- [57] A. V. Aho and N. J. A. Sloane, “Some doubly exponential sequences,” *Fibonacci Quarterly*, vol. 11, pp. 429–437, 1970.
- [58] F. M. J. Willems, “Coding for a binary independent piecewise-identically-distributed source,” *IEEE Transactions on Information Theory*, vol. 42, pp. 2210–2217, Nov 1996.
- [59] J. H. Friedman, “Multivariate adaptive regression splines,” *The Annals of Statistics*, vol. 19, no. 1, pp. 1–67, 1991.
- [60] J. H. Friedman, “Fast MARS.” <http://www.milbo.users.sonic.net/earth/Friedman-FastMars.pdf>, 1993.
- [61] C. Blake and C. Merz, “UCI Machine Learning Repository.” <http://archive.ics.uci.edu/ml/index.html>, 2015.
- [62] S. Wiggins, *Introduction to Applied Nonlinear Dynamical Systems and Chaos*, vol. 2. Springer New York, 2003.
- [63] G. K. Smyth, “Australasian data and story library (OzDASL).” <http://www.statsci.org/data>, 2015.
- [64] P. Vlachos, “StatLib.” <http://lib.stat.cmu.edu/datasets>, 2015.
- [65] J.-N. Hwang, S.-R. Lay, M. Maechler, R. D. Martin, and J. Schimert, “Regression modeling in back-propagation and projection pursuit learning,” *IEEE Transactions on Neural Networks*, vol. 5, pp. 342–353, May 1994.
- [66] D. Haussler, J. Kivinen, and M. K. Warmuth, “Sequential prediction of individual sequences under general loss functions,” *IEEE Transactions on Information Theory*, vol. 44, no. 2, pp. 1906–1925, 1998.
- [67] L. Rabiner and R. Schafer, *Digital Processing of Speech Signals*. Englewood Cliffs, NJ: Prentice-Hall, 1978.
- [68] S. Tunc, M. A. Donmez, and S. S. Kozat, “Optimal investment under transaction costs: A threshold rebalanced portfolio approach,” *IEEE Transactions on Signal Processing*, vol. 61, pp. 3129–3142, June 2013.

- [69] T. Weissman and N. Merhav, “Universal prediction of individual binary sequences in the presence of noise,” *IEEE Transactions on Information Theory*, vol. 47, no. 6, pp. 2151–2173, 2001.
- [70] S. S. Kozat and A. C. Singer, “Universal switching linear least squares prediction,” *IEEE Transactions on Signal Processing*, vol. 56, pp. 189–204, Jan. 2008.
- [71] J. Kivinen and M. K. Warmuth, “Exponentiated gradient versus gradient descent for linear predictors,” *Journal of Information and Computation*, vol. 132, no. 1, pp. 1–62, 1997.
- [72] N. Cesa-Bianchi and G. Lugosi, *Prediction, Learning, and Games*. Cambridge University Press, 2006.
- [73] N. D. Vanli, M. O. Sayin, H. Yildiz, T. Goze, and S. S. Kozat, “Energy consumption forecasting via order preserving pattern matching,” in *2014 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, pp. 238–242, Dec 2014.
- [74] D. P. Helmbold and R. E. Schapire, “Predicting nearly as well as the best pruning of a decision tree,” *Machine Learning*, vol. 27, no. 1, pp. 51–68, 1997.
- [75] V. Mathews, “Adaptive polynomial filters,” *IEEE Signal Processing Magazine*, vol. 8, no. 3, pp. 10–26, 1991.
- [76] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, UK: Cambridge University Press, 2004.
- [77] E. Hazan, A. Agarwal, and S. Kale, “Logarithmic regret algorithms for online convex optimization,” *Machine Learning*, vol. 69, no. 2-3, pp. 169–192, 2007.
- [78] J. C. Duchi and Y. Singer, “Efficient online and batch learning using forward-backward splitting,” *Journal of Machine Learning Research*, vol. 10, pp. 2873–2898, Dec 2009.

- [79] E. Hazan and S. Kale, “Beyond the regret minimization barrier: optimal algorithms for stochastic strongly-convex optimization,” *Journal of Machine Learning Research*, vol. 15, p. 24892512, 2014.
- [80] N. D. Vanli, M. O. Sayin, and S. S. Kozat, “Stochastic subgradient algorithms for strongly convex optimization over distributed networks.” <http://arxiv.org/pdf/1409.8277v1.pdf>, 2015.
- [81] J. C. Duchi, A. Agarwal, and M. J. Wainwright, “Dual averaging for distributed optimization: Convergence analysis and network scaling,” *IEEE Transactions on Automatic Control*, vol. 57, pp. 592–606, March 2012.
- [82] J. C. Duchi, E. Hazan, and Y. Singer, “Adaptive subgradient methods for online learning and stochastic optimization,” *Journal of Machine Learning Research*, vol. 12, pp. 2121–2159, July 2011.
- [83] F. Yan, S. Sundaram, S. V. N. Vishwanathan, and Y. Qi, “Distributed autonomous online learning: Regrets and intrinsic privacy-preserving properties,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, pp. 2483–2493, Nov 2013.
- [84] M. Zinkevich, “Online convex programming and generalized infinitesimal gradient ascent,” in *Proceedings of the 20th International Conference on Machine Learning*, 2003.
- [85] V. D. Blondel, J. M. Hendrickx, A. Olshevsky, and J. N. Tsitsiklis, “Convergence in multiagent coordination, consensus, and flocking,” in *Proceedings of 44th IEEE Conference on Decision and Control (CDC-ECC)*, pp. 2996–3000, 2005.
- [86] R. K. Pace and R. Barry, “Sparse spatial autoregressions,” *Statistics and Probability Letters*, vol. 33, pp. 291–297, 1997.
- [87] S. I. Hill and R. C. Williamson, “Convergence of exponentiated gradient algorithms,” *IEEE Transactions on Signal Processing*, vol. 49, pp. 1208–1215, Jun 2001.

Appendix A

Proofs

A.1 Proof of Lemma 3.3

We prove this lemma by induction. For a depth-0 tree, we clearly have

$$P_\lambda(n) = \exp \left\{ -\frac{1}{2a} \sum_{\substack{t \leq n \\ \mathbf{x}[t] \in \mathcal{R}_\lambda}} (d[t] - \delta_\lambda[t])^2 \right\}$$

where $\hat{d}_\lambda[t]$ is the prediction of the only pruning in the depth-0 tree. Since a single model (or a certain event) requires zero bits to be represented on a binary tree, the equality follows. Now let us assume that for a complete depth- k tree, the equality

$$P_\lambda(n) = \sum_{m_k \in \mathcal{C}_k} 2^{-B_{m_k}} P(n|m_k),$$

holds, where $\mathcal{C}_k$ represents the set of all prunings on the depth- k complete tree, $m_k \in \mathcal{C}_k$ represents the m th pruning on the depth- k complete tree, and

$$P(n|m_k) \triangleq \exp \left\{ -\frac{1}{2a} \sum_{t=1}^n (d[t] - \delta^{(m_k)}[t])^2 \right\}.$$

Then, the complete decision tree of depth $k+1$ can be constructed by building two different depth- k trees and creating a root node whose children are the root

nodes of these two depth- k trees. That is, we add bit 0 as the prefix to the first depth- k tree and add bit 1 as the prefix to the second depth- k tree. Then, letting $m_k^{(1)}$ represent the m th pruning on the first depth- k complete tree and $m_k^{(2)}$ represent the m th pruning on the second depth- k complete tree, we obtain

$$\begin{aligned}
P_\lambda(n) &= \frac{1}{2}P_0(n)P_1(n) + \frac{1}{2} \exp \left\{ -\frac{1}{2a} \sum_{\substack{t \leq n \\ \mathbf{x}[t] \in \mathcal{R}_\lambda}} (d[t] - \delta_\lambda[t])^2 \right\} \\
&= \frac{1}{2} \left(\sum_{m_k^{(1)} \in \mathcal{C}_k^{(1)}} 2^{-B_{m_k^{(1)}}} P(n|m_k^{(1)}) \right) \left(\sum_{m_k^{(2)} \in \mathcal{C}_k^{(2)}} 2^{-B_{m_k^{(2)}}} P(n|m_k^{(2)}) \right) \\
&\quad + \frac{1}{2} \exp \left\{ -\frac{1}{2a} \sum_{\substack{t \leq n \\ \mathbf{x}[t] \in \mathcal{R}_\lambda}} (d[t] - \delta_\lambda[t])^2 \right\} \\
&= \sum_{m_k^{(1)} \in \mathcal{C}_k^{(1)}} \sum_{m_k^{(2)} \in \mathcal{C}_k^{(2)}} 2^{-1-B_{m_k^{(1)}}-B_{m_k^{(2)}}} P(n|m_k^{(1)})P(n|m_k^{(2)}) \\
&\quad + 2^{-1} \exp \left\{ -\frac{1}{2a} \sum_{t \leq n} (d[t] - \delta_\lambda[t])^2 \right\},
\end{aligned}$$

where the second step follows from the induction hypotheses. Here, we make the following observation. Let $\mathcal{L}(m)$ represent the set of leaf nodes in pruning m . Then, for any $m_{k+1} \in \mathcal{C}_{k+1}$, we either have $\mathcal{L}(m_{k+1}) = \{\lambda\}$ or $\mathcal{L}(m_{k+1}) = \mathcal{L}(m_k^{(1)}) \cup \mathcal{L}(m_k^{(2)})$. Thus, for any $m_{k+1} \in \mathcal{C}_{k+1} - \lambda$, we have $P(n|m_{k+1}) = P(n|m_k^{(1)})P(n|m_k^{(2)})$. We also note that since $m_k^{(1)}$ and $m_k^{(2)}$ can be represented by $B_{m_k^{(1)}}$ and $B_{m_k^{(2)}}$ bits, respectively, any $m_{k+1} \in \mathcal{C}_{k+1} - \lambda$ can be represented by $1 + B_{m_k^{(1)}} + B_{m_k^{(2)}}$ bits, where the first bit indicates that the pruning is not λ . Therefore, we obtain

$$P_\lambda(n) = \sum_{m_{k+1} \in \mathcal{C}_{k+1}} 2^{-B_{m_{k+1}}} P(n|m_{k+1}), \quad (\text{A.1})$$

which is the desired result. $\square$

A.2 Proof of Lemma 3.5

We prove this lemma by induction. For $l = 1$, we have

$$\begin{aligned}
P_\lambda(t) &= \frac{1}{2}P_0(t)P_1(t) + \frac{1}{2}\exp\left\{-\frac{1}{2a}\sum_{\substack{t' \leq t \\ \mathbf{x}[t'] \in \mathcal{R}_\lambda}}\left(d[t'] - \hat{d}_\lambda[t']\right)^2\right\} \\
&= \frac{1}{2}\exp\left\{-\frac{1}{2a}\sum_{\substack{t' \leq t \\ \mathbf{x}[t'] \in \mathcal{R}_0}}\left(d[t'] - \hat{d}_0[t']\right)^2\right\}\exp\left\{-\frac{1}{2a}\sum_{\substack{t' \leq t \\ \mathbf{x}[t'] \in \mathcal{R}_1}}\left(d[t'] - \hat{d}_1[t']\right)^2\right\} \\
&\quad + \frac{1}{2}\exp\left\{-\frac{1}{2a}\sum_{\substack{t' \leq t \\ \mathbf{x}[t'] \in \mathcal{R}_\lambda}}\left(d[t'] - \hat{d}_\lambda[t']\right)^2\right\}.
\end{aligned}$$

Then, our claim holds with $\pi_\lambda[t] = \frac{1}{2}$, and

$$\begin{aligned}
\pi_0[t] &= \frac{1}{2}\exp\left\{-\frac{1}{2a}\sum_{\substack{t' \leq t \\ \mathbf{x}[t'] \in \mathcal{R}_1}}\left(d[t'] - \hat{d}_1[t']\right)^2\right\} = \pi_\lambda[t]P_1(t), \\
\pi_1[t] &= \frac{1}{2}\exp\left\{-\frac{1}{2a}\sum_{\substack{t' \leq t \\ \mathbf{x}[t'] \in \mathcal{R}_0}}\left(d[t'] - \hat{d}_0[t']\right)^2\right\} = \pi_\lambda[t]P_0(t).
\end{aligned}$$

We then assume that the claim holds for a depth- k complete tree, where for any node κ of length $l = k$, we have

$$P_\lambda^{(k)}(t) = \sum_{\kappa_i \in \mathcal{P}(\kappa)} \pi_{\kappa_i}^{(k)}[t] \exp\left\{-\frac{1}{2a}\sum_{\substack{t' \leq t \\ \mathbf{x}[t'] \in \mathcal{R}_{\kappa_i}}}\left(d[t'] - \hat{d}_{\kappa_i}[t']\right)^2\right\},$$

where $\pi_{\kappa_i}^{(k)}[t]$ are the corresponding weights as in (3.10). Then, we consider a depth- $(k+1)$ complete tree, where for a node $\sigma\kappa$ of length $l = k+1$ (where σ is

a binary letter), we have

$$\begin{aligned}
P_\lambda^{(k+1)}(t) &= \frac{1}{2}P_{\sigma^c}^{(k+1)}(t)P_\sigma^{(k+1)}(t) + \frac{1}{2}\exp\left\{-\frac{1}{2a}\sum_{\substack{t' \leq t \\ \mathbf{x}[t'] \in \mathcal{R}_\lambda}} \left(d[t'] - \hat{d}_\lambda[t']\right)^2\right\} \\
&= \frac{1}{2}P_{\sigma^c}^{(k+1)}(t)\left(\sum_{\kappa_i \in \mathcal{P}(\kappa)} \pi_{\sigma\kappa_i}^{(k)}[t] \exp\left\{-\frac{1}{2a}\sum_{\substack{t' \leq t \\ \mathbf{x}[t'] \in \mathcal{R}_{\sigma\kappa_i}}} \left(d[t'] - \hat{d}_{\sigma\kappa_i}[t']\right)^2\right\}\right) \\
&\quad + \frac{1}{2}\exp\left\{-\frac{1}{2a}\sum_{\substack{t' \leq t \\ \mathbf{x}[t'] \in \mathcal{R}_\lambda}} \left(d[t'] - \hat{d}_\lambda[t']\right)^2\right\} \\
&= \sum_{\kappa_i \in \mathcal{P}(\kappa)} \frac{1}{2}P_{\sigma^c}^{(k+1)}(t)\pi_{\sigma\kappa_i}^{(k)}[t] \exp\left\{-\frac{1}{2a}\sum_{\substack{t' \leq t \\ \mathbf{x}[t'] \in \mathcal{R}_{\sigma\kappa_i}}} \left(d[t'] - \hat{d}_{\sigma\kappa_i}[t']\right)^2\right\} \\
&\quad + \frac{1}{2}\exp\left\{-\frac{1}{2a}\sum_{\substack{t' \leq t \\ \mathbf{x}[t'] \in \mathcal{R}_\lambda}} \left(d[t'] - \hat{d}_\lambda[t']\right)^2\right\}, \tag{A.2}
\end{aligned}$$

where the second line follows from the induction hypotheses. Here, we note that from the induction hypotheses, we also have

$$\pi_{\sigma\kappa_i}^{(k)}[t] \triangleq \begin{cases} \frac{1}{2} & , \text{ if } i = 0 \\ \frac{1}{2}P_{\sigma\kappa_{i-1}\nu_i^c}^{(k)}(t)\pi_{\sigma\kappa_{i-1}}^{(k)}[t] & , \text{ if } 1 \leq i \leq l-1 \\ P_{\sigma\kappa_{i-1}\nu_i^c}^{(k)}(t)\pi_{\sigma\kappa_{i-1}}^{(k)}[t] & , \text{ if } i = l \end{cases}$$

$\forall \kappa_i \in \mathcal{P}(\kappa)$. We also note that we have $P_{\sigma\kappa_{i-1}\nu_i^c}^{(k)}(t) = P_{\sigma\kappa_{i-1}\nu_i^c}^{(k+1)}(t)$ from the definition of the function $P_\kappa(\cdot)$ since the depth- k tree is a subtree of the depth- $(k+1)$ tree. Then, by defining an intermediate parameter

$$\beta_{\sigma\kappa_i}^{(k+1)}[t] \triangleq \begin{cases} \frac{1}{4}P_{\sigma^c}^{(k+1)}(t) & , \text{ if } i = 0 \\ \frac{1}{2}P_{\sigma\kappa_{i-1}\nu_i^c}^{(k+1)}(t)\beta_{\sigma\kappa_{i-1}}^{(k+1)}[t] & , \text{ if } 1 \leq i \leq l-1 \\ P_{\sigma\kappa_{i-1}\nu_i^c}^{(k+1)}(t)\beta_{\sigma\kappa_{i-1}}^{(k+1)}[t] & , \text{ if } i = l \end{cases} \tag{A.3}$$

$\forall \kappa_i \in \mathcal{P}(\kappa)$, we can rewrite (A.2) as follows

$$P_\lambda^{(k+1)}(t) = \sum_{\kappa_i \in \mathcal{P}(\kappa)} \beta_{\sigma\kappa_i}^{(k+1)} \exp \left\{ -\frac{1}{2a} \sum_{\substack{t' \leq t \\ \mathbf{x}[t'] \in \mathcal{R}_{\sigma\kappa_i}}} \left(d[t'] - \hat{d}_{\sigma\kappa_i}[t'] \right)^2 \right\} \\ + \frac{1}{2} \exp \left\{ -\frac{1}{2a} \sum_{\substack{t' \leq t \\ \mathbf{x}[t'] \in \mathcal{R}_\lambda}} \left(d[t'] - \hat{d}_\lambda[t'] \right)^2 \right\}.$$

Here, we note that the sum on the right hand side is over $\mathcal{P}(\kappa)$. However, we can combine all the terms in the right hand side into a single parentheses of $\mathcal{P}(\sigma\kappa)$ as follows

$$P_\lambda^{(k+1)}(t) = \sum_{(\sigma\kappa)_i \in \mathcal{P}(\sigma\kappa)} \pi_{(\sigma\kappa)_i}^{(k+1)}[t] \exp \left\{ -\frac{1}{2a} \sum_{\substack{t' \leq t \\ \mathbf{x}[t'] \in \mathcal{R}_{(\sigma\kappa)_i}}} \left(d[t'] - \hat{d}_{(\sigma\kappa)_i}[t'] \right)^2 \right\},$$

where $(\sigma\kappa)_i \in \mathcal{P}(\sigma\kappa)$ is the length- i prefix to $\sigma\kappa$ (where we note that $(\sigma\kappa)_i = \sigma\kappa_{i-1}$ for $1 \leq i \leq l+1$) and

$$\pi_{(\sigma\kappa)_i}^{(k+1)}[t] \triangleq \begin{cases} \frac{1}{2} & , \text{ if } i = 0 \\ \beta_{\sigma\kappa_{i-1}}^{(k+1)} & , \text{ if } 1 \leq i \leq l+1 \end{cases} \\ = \begin{cases} \frac{1}{2} & , \text{ if } i = 0 \\ \frac{1}{2} P_{\sigma\kappa_{i-1}\nu_i^c}^{(k+1)}(t) \pi_{\sigma\kappa_{i-1}}^{(k+1)}[t] & , \text{ if } 1 \leq i \leq l \\ P_{\sigma\kappa_{i-1}\nu_i^c}^{(k+1)}(t) \pi_{\sigma\kappa_{i-1}}^{(k+1)}[t] & , \text{ if } i = l+1 \end{cases}$$

where the last line follows from the definition in (A.3). This concludes the proof.

□

A.3 Proof of Theorem 4.1

We define an intermediate variable $z_t^{\{k\}}$ such that

$$w_t^{\{k\}} = \frac{e^{-z_t^{\{k\}}}}{\sum_{r=1}^K e^{-z_t^{\{r\}}}}$$

and correspondingly a function $\mathbf{w}_t = f(\mathbf{z}_t)$, where $\mathbf{w}_t \triangleq [w_t^{\{1\}}, \dots, w_t^{\{K\}}]^T$ and $\mathbf{z}_t \triangleq [z_t^{\{1\}}, \dots, z_t^{\{K\}}]^T$. Then, (4.11) yields

$$\mathbf{z}_t = \mathbf{z}_{t-1} - \mu \hat{\mathbf{d}}_{t-1} \epsilon_{t-1}. \quad (\text{A.4})$$

We then apply the Euler discretization technique [87] to (A.4) and obtain

$$\mathbf{w}_t = \mathbf{w}_{t-1} + \mu J(f(\mathbf{z}_{t-1})) \hat{\mathbf{d}}_{t-1} \epsilon_{t-1}, \quad (\text{A.5})$$

where $J(f(\mathbf{z}_{t-1}))$ is the Jacobian matrix of $f(\mathbf{z}_{t-1})$ with respect to $\mathbf{z}_{t-1}$. In particular, $J(f(\mathbf{z}_t))$ is given by

$$J(f(\mathbf{z}_t)) = \frac{1}{\left(\sum_{r=1}^K e^{-z_t^{\{r\}}}\right)^2} \left\{ \left(\sum_{r=1}^K e^{-z_t^{\{r\}}} \right) \mathbf{D}_t - \mathbf{Z}_t \right\}, \quad (\text{A.6})$$

where

$$\mathbf{D}_t \triangleq \begin{bmatrix} e^{-z_t^{\{1\}}} & & & \\ & \ddots & & \\ & & e^{-z_t^{\{K\}}} & \end{bmatrix} \quad (\text{A.7})$$

and

$$\mathbf{Z}_t \triangleq \begin{bmatrix} \left(e^{-z_t^{\{1\}}}\right)^2 & e^{-z_t^{\{1\}}} e^{-z_t^{\{2\}}} & \dots & e^{-z_t^{\{1\}}} e^{-z_t^{\{K\}}} \\ e^{-z_t^{\{2\}}} e^{-z_t^{\{1\}}} & \left(e^{-z_t^{\{2\}}}\right)^2 & \dots & e^{-z_t^{\{2\}}} e^{-z_t^{\{K\}}} \\ \vdots & \vdots & \ddots & \vdots \\ e^{-z_t^{\{K\}}} e^{-z_t^{\{1\}}} & e^{-z_t^{\{K\}}} e^{-z_t^{\{2\}}} & \dots & \left(e^{-z_t^{\{K\}}}\right)^2 \end{bmatrix}.$$

Letting

$$\mathbf{w}_o = \arg \min_{\mathbf{w} \in \Delta^K} \sum_{t=1}^{\infty} \ell(d_t, \hat{\mathbf{d}}_t^T \mathbf{w}), \quad (\text{A.8})$$

where

$$\Delta^K \triangleq \{\mathbf{w} \in \mathbb{R}^K \mid w^{(i)} \geq 0 \forall i \in \{1, \dots, K\}, \|\mathbf{w}\|_1 = 1\}$$

denotes K -dimensional unit simplex, we define a deviation parameter $\tilde{\mathbf{w}}_t = \mathbf{w}_o - \mathbf{w}_t$. Then, (A.5) yields

$$\tilde{\mathbf{w}}_t = \tilde{\mathbf{w}}_{t-1} - \mu J(f(\mathbf{z}_{t-1})) \hat{\mathbf{d}}_{t-1} \epsilon_{t-1}. \quad (\text{A.9})$$

Here, note that the outputs of the predictors are correlated. Hence, let the autocorrelation matrix of the outputs of the FS predictors at the steady-state be defined as

$$\mathbf{R} \triangleq \lim_{t \rightarrow \infty} E \left[\hat{\mathbf{d}}_t \hat{\mathbf{d}}_t^T \right], \quad (\text{A.10})$$

where $\hat{\mathbf{d}}_t \triangleq [\hat{d}_t^{\{1\}}, \dots, \hat{d}_t^{\{K\}}]^T$. We can decompose $\mathbf{R}$ through the eigen decomposition as follows $\mathbf{R} = \mathbf{T}\mathbf{\Lambda}\mathbf{T}^T$. Multiplying both sides of (A.9) by $\mathbf{T}^T$, we obtain

$$\bar{\mathbf{w}}_t = \bar{\mathbf{w}}_{t-1} - \mu \mathbf{T}^T J(f(\mathbf{z}_{t-1})) \mathbf{T} \bar{\mathbf{d}}_{t-1} \epsilon_{t-1}, \quad (\text{A.11})$$

where $\bar{\mathbf{w}}_t \triangleq \mathbf{T}^T \mathbf{w}_t$ and $\bar{\mathbf{d}}_t \triangleq \mathbf{T}^T \hat{\mathbf{d}}_t$. For notational simplicity, we denote $\bar{\mathbf{J}}_{t-1} \triangleq \mathbf{T}^T J(f(\mathbf{z}_{t-1})) \mathbf{T}$. Then, the weighted energy recursion of (A.5) is given by

$$\begin{aligned} E\|\bar{\mathbf{w}}_t\|_{\Sigma}^2 &= E\|\bar{\mathbf{w}}_{t-1}\|_{\Sigma}^2 - 2\mu E \left[\bar{\mathbf{w}}_{t-1}^T \Sigma \bar{\mathbf{J}}_{t-1} \bar{\mathbf{d}}_{t-1} \bar{\mathbf{d}}_{t-1}^T \bar{\mathbf{w}}_{t-1} \right] \\ &\quad + \mu^2 E \left[\epsilon_{t-1}^2 \bar{\mathbf{d}}_{t-1}^T \bar{\mathbf{J}}_{t-1}^T \Sigma \bar{\mathbf{J}}_{t-1} \bar{\mathbf{d}}_{t-1} \right], \end{aligned} \quad (\text{A.12})$$

where Σ is a positive semi-definite weight matrix.

We assume that the random variables $\|\bar{\mathbf{d}}_t\|_{\bar{\mathbf{J}}_t^T \Sigma \bar{\mathbf{J}}_t}^2$ and ϵ_t^2 are asymptotically uncorrelated. Hence, at steady-state, we have

$$\lim_{t \rightarrow \infty} E\|\bar{\mathbf{w}}_t\|_{\Sigma \bar{\mathbf{J}} \mathbf{\Lambda}}^2 = \lim_{t \rightarrow \infty} \frac{\mu}{2} E[\epsilon_t^2] \text{Tr} \left\{ \mathbf{\Lambda} \bar{\mathbf{J}}^T \Sigma \bar{\mathbf{J}} \right\}, \quad (\text{A.13})$$

where we approximate the Jacobian matrix as follows

$$\bar{\mathbf{J}} \triangleq \mathbf{T}^T J(f(\mathbf{z}_o)) \mathbf{T}, \quad (\text{A.14})$$

and define

$$\mathbf{z}_o = \lim_{\mathbf{w}_t \rightarrow \mathbf{w}_o} f(\mathbf{w}_t)^{-1}. \quad (\text{A.15})$$

We note that $\epsilon_t = \bar{\mathbf{w}}_t^T \bar{\mathbf{d}}_t + n_t$, where

$$n_t = \sum_{k=1}^K w_o^{\{k\}} e_t^{\{k\}}, \quad (\text{A.16})$$

and $e_t^{\{k\}} = d_t - \hat{d}_t^{\{k\}}$. This yields that

$$\lim_{t \rightarrow \infty} E[\epsilon_t^2] = \lim_{t \rightarrow \infty} E\|\bar{\mathbf{w}}_t\|_{\mathbf{\Lambda}}^2 + E[n_t^2]. \quad (\text{A.17})$$

Letting $\Sigma = \bar{\mathbf{J}}^{-1}$ and

$$\sigma_n^2 = \lim_{t \rightarrow \infty} E[n_t^2], \quad (\text{A.18})$$

we obtain the steady-state excess MSE $\zeta \triangleq \lim_{t \rightarrow \infty} E\|\bar{\mathbf{w}}_t\|_{\mathbf{\Lambda}}^2$ as

$$\zeta = \frac{\mu \sigma_n^2 \text{Tr} \left\{ \mathbf{\Lambda} \bar{\mathbf{J}}^T \right\}}{2 - \mu \text{Tr} \left\{ \mathbf{\Lambda} \bar{\mathbf{J}}^T \right\}}, \quad (\text{A.19})$$

and the steady-state MSE is given by

$$\text{MSE} = \frac{2\sigma_n^2}{2 - \mu \text{Tr} \left\{ \mathbf{\Lambda} \bar{\mathbf{J}}^T \right\}}. \quad (\text{A.20})$$

This concludes the proof of Theorem 4.1. $\square$

A.4 Proof of Lemma 4.2

In order to prove $T_t^{\{c_{0,1}\}} = L_t$, we use mathematical induction. First, for $h = 1$, we have a single state and the equation $T_t^{\{c_{0,1}\}} = L_t$ is trivially satisfied for this case. Assuming that $T_t^{\{c_{0,1}\}} = L_t$ holds for some $h \geq 1$, let us consider the term $T_t^{\{c_{0,1}\}}$ for $h + 1$.

For ease of exposition let us drop the time index t from the subscript, and use $T_h^{\{c_{i,j}\}}$ to refer to the induction hypothesis and $T_{h+1}^{\{c_{i,j}\}}$ to refer to the objective function. Similarly, we let $\mathcal{C}_{h+1}^{\{c_{i,j}\}}$ represent the set of equivalence classes at hierarchy level $i + 1$ that are connected to $c_{i,j}$ for the induction case. According to the definition in (4.17), we have

$$T_{h+1}^{\{c_{0,1}\}} = L_{h+1}^{\{c_{0,1}\}} + \prod_{c_{1,j} \in \mathcal{C}_{h+1}^{\{c_{0,1}\}}} T_{h+1}^{\{c_{1,j}\}} \quad (\text{A.21})$$

$$= L_{h+1}^{\{c_{0,1}\}} + \prod_{j=1}^{J_{h+1}} L_{h,j}, \quad (\text{A.22})$$

where the last line follows from the induction hypothesis with $J_{h+1} \triangleq |\mathcal{C}_{h+1}^{\{c_{0,1}\}}|$ and $L_{h,j}$ representing the loss of the j th equivalence class in $\mathcal{C}_{h+1}^{\{c_{0,1}\}}$. Note that for hierarchical structure of depth $h + 1$, we have $K_{h+1} = (K_h)^{J_{h+1}} + 1$, which can be immediately observed from (A.22). Inserting the loss definition in (4.13) to (A.22), we obtain

$$\begin{aligned} T_{h+1}^{\{c_{0,1}\}} &= L_{h+1}^{\{c_{0,1}\}} + \prod_{j=1}^{J_{h+1}} \left(\sum_{k=1}^{K_h} L_{h,j}^{\{k\}} \right) \\ &= L_{h+1}^{\{c_{0,1}\}} + \sum_{k=1}^{(K_h)^{J_{h+1}}} L_{h+1}^{\{k\}}, \end{aligned}$$

which shows that $T_{h+1}^{\{c_{0,1}\}} = L_{h+1}$ holds, where the last line follows from (4.16) and from the following observation: Combining the states of an arbitrary FS predictor with hierarchy level h from each J_{h+1} paths, we obtain an FS predictor with hierarchy level $h + 1$. Hence, the proof is concluded. $\square$

A.5 Proof of Lemma 4.3

We first consider the right hand side of the equality in (3.5) and observe that it is similar to

$$L_{t+1}^{\{c_{0,1}\}} = \sum_{k=1}^K L_t^{\{k\}} \exp\left(-\mu \epsilon_t \hat{d}_t^{\{k\}}\right), \quad (\text{A.23})$$

where only the last exponential term in (A.23) is replaced by $\hat{d}_t^{\{k\}}$. Hence, following similar lines to the proof of Lemma 4.2 (i.e., using an induction method), it can be shown that the equality (3.5) can be achieved using the recursion in (4.19).

We then consider the product term in (4.19) and using (4.17), we obtain this product term (i.e., the last term in (4.19)) as follows

$$P_t^{\{c_{i,j}\}} \triangleq \prod_{\substack{c_{i+1,k} \in \mathcal{C}^{\{c_{i,j}\}} \\ c_{i+1,k} \neq c_{i+1,j'}}} T_t^{\{c_{k,l}\}} = \frac{T_t^{\{c_{i,j}\}} - L_t^{\{c_{i,j}\}}}{T_t^{\{c_{i+1,j'}\}}}. \quad (\text{A.24})$$

Putting (A.24) back in (4.19), we obtain

$$\tilde{T}_t^{\{c_{i,j}\}} = L_t^{\{c_{i,j}\}} \hat{d}_t^{\{c_{i,j}\}} + \tilde{T}_t^{\{c_{i',j'}\}} P_t^{\{c_{i,j}\}}. \quad (\text{A.25})$$

Hence, $\tilde{T}_t^{\{c_{0,1}\}}$ can be calculated with computational complexity $O(h)$. $\square$

A.6 Proof of Lemma 4.4

Since s_t is an element of one and only one equivalence class from each hierarchy level $0 \leq i \leq h$, computing $L_{t+1}^{\{c_{i,j}\}}$ from $L_t^{\{c_{i,j}\}}$ using (4.22) requires only $h + 1$

updates. Hence, (4.22) can be calculated in $O(h)$ computations. We next analyze the number of computations required to perform (4.23).

Since the product term in (4.23) contains the equivalence classes in $\mathcal{C}^{\{c_{i,j}\}}$ that do not contain s_t , they are not updated from time t to $t+1$, i.e., $T_{t+1}^{\{c_{i+1,k}\}} = T_t^{\{c_{i+1,k}\}}$, $\forall c_{i+1,k} \in \mathcal{C}^{\{c_{i,j}\}} : s_t \notin c_{i+1,k}$. Then, we have $P_{t+1}^{\{c_{i,j}\}} = P_t^{\{c_{i,j}\}}$ according to (A.24). Putting (A.24) back in (4.23), we obtain

$$T_{t+1}^{\{c_{i,j}\}} = L_t^{\{c_{i,j}\}} \exp\left(-\mu \hat{d}_t^{\{c_{i,j}\}} \epsilon_t\right) + T_{t+1}^{\{c_{i+1,j'}\}} P_{t+1}^{\{c_{i,j}\}}, \quad (\text{A.26})$$

which, in its current form, can be calculated with a computational complexity $O(h)$. $\square$

A.7 Proof of Lemma 5.3

Let $\mathbf{W}_t \triangleq [\mathbf{w}_{t,1}, \dots, \mathbf{w}_{t,K}]$, $\mathbf{G}_t \triangleq [\mathbf{g}_{t,1}, \dots, \mathbf{g}_{t,K}]$, and $\mathbf{H}_t \triangleq [\mathbf{h}_{t,1}, \dots, \mathbf{h}_{t,K}]$. Then, by letting $\mathbf{1}_k$ denote the vector of all zeros but only a single one at its k th entry, we can write the desired norm as

$$\|\bar{\mathbf{w}}_t - \mathbf{w}_{t,k}\| = \left\| \mathbf{W}_t \left(\frac{1}{K} \mathbf{1} - \mathbf{1}_k \right) \right\|. \quad (\text{A.27})$$

According to our update rule, we have the following recursion

$$\mathbf{W}_t = \mathbf{W}_1 \mathbf{C}^{t-1} - \sum_{s=1}^{t-1} \mu_{t-s} (\mathbf{G}_{t-s} + \mathbf{H}_{t-s+1}) \mathbf{C}^{s-1}. \quad (\text{A.28})$$

Putting (A.28) back in (A.27) and noting that $\mathbf{C}\mathbf{1} = \mathbf{1}$ due to Assumption 3 and $\bar{\mathbf{w}}_1 = \mathbf{w}_{1,k}$, $\forall k \in \{1, \dots, K\}$ due to Assumption 4, we obtain

$$\begin{aligned} \|\bar{\mathbf{w}}_t - \mathbf{w}_{t,k}\| &= \sum_{s=1}^{t-1} \mu_{t-s} \left\| (\mathbf{G}_{t-s} + \mathbf{H}_{t-s+1}) \left(\frac{1}{K} \mathbf{1} - \mathbf{C}^{s-1} \mathbf{1}_k \right) \right\| \\ &\leq \sum_{s=1}^{t-1} \mu_{t-s} \|\mathbf{G}_{t-s} + \mathbf{H}_{t-s+1}\|_F \left\| \frac{1}{K} \mathbf{1} - \mathbf{C}^{s-1} \mathbf{1}_k \right\|. \end{aligned} \quad (\text{A.29})$$

Here, the first term of (A.29) can be upper bounded as follows

$$\|\mathbf{G}_{t-s} + \mathbf{H}_{t-s+1}\|_F \leq 2A\sqrt{K}, \quad (\text{A.30})$$

since the norms of the subgradients are upper bounded by A due to Assumption 2.

We then consider the second term of (A.29). In order to derive an upper bound, we first let $\sigma_1(\mathbf{A}) \geq \sigma_2(\mathbf{A}) \geq \dots \geq \sigma_K(\mathbf{A})$ denote the singular values of a matrix $\mathbf{A}$ and $\lambda_1(\mathbf{A}) \geq \lambda_2(\mathbf{A}) \geq \dots \geq \lambda_K(\mathbf{A})$ denote the eigenvalues of a symmetric matrix $\mathbf{A}$. Since the communication graph $\mathbf{C}$ is a doubly stochastic matrix (by Assumption 3), we have $\sigma_1(\mathbf{C}) = 1$ and $\lambda_1(\mathbf{C}^T \mathbf{C}) = 1$. Then, we let $\mathbf{B} \triangleq \frac{1}{K} \mathbf{1} \mathbf{1}^T$ and consider the following norm

$$\begin{aligned} \left\| \frac{1}{K} \mathbf{1} - \mathbf{C}^{s-1} \mathbf{1}_k \right\| &= \left\| \mathbf{B} \mathbf{1}_k - \mathbf{C}^{s-1} \mathbf{1}_k \right\| \\ &= \left\| (\mathbf{B} - \mathbf{C})^{s-1} \mathbf{1}_k \right\|, \end{aligned} \quad (\text{A.31})$$

where

$$\left\| (\mathbf{B} - \mathbf{C})^t \mathbf{1}_k \right\| \leq \sigma_1(\mathbf{B} - \mathbf{C}) \left\| (\mathbf{B} - \mathbf{C})^{t-1} \mathbf{1}_k \right\|, \quad (\text{A.32})$$

$\forall t \geq 1$. Thus, applying (A.32) $s - 1$ times to (A.31), we obtain

$$\left\| \frac{1}{K} \mathbf{1} - \mathbf{C}^{s-1} \mathbf{1}_k \right\| \leq \sigma_1^{s-1}(\mathbf{B} - \mathbf{C}) \left\| \mathbf{1}_k \right\|. \quad (\text{A.33})$$

Here, we observe that $\mathbf{C}$ is a doubly stochastic matrix by Assumption 3, hence $\lambda_1(\mathbf{C}) = 1$. Therefore, the eigenspectrums of $\mathbf{B} - \mathbf{C}$ and $(\mathbf{B} - \mathbf{C})^T(\mathbf{B} - \mathbf{C})$ are equal to the eigenspectrums of $\mathbf{C}$ and $\mathbf{C}^T \mathbf{C}$, respectively, except the largest eigenvalues, i.e., $\lambda_1(\mathbf{C}) = \lambda_1(\mathbf{C}^T \mathbf{C}) = 1$, are removed. Thus, from (A.33), we obtain

$$\left\| \frac{1}{K} \mathbf{1} - \mathbf{C}^{s-1} \mathbf{1}_k \right\| \leq \sigma_2^{s-1}(\mathbf{C}). \quad (\text{A.34})$$

Putting (A.30) and (A.34) back in (A.29), we obtain

$$\left\| \bar{\mathbf{w}}_t - \mathbf{w}_{t,k} \right\| \leq 2A\sqrt{K} \sum_{s=1}^{t-1} \mu_{t-s} \sigma_2^{s-1}(\mathbf{C}), \quad (\text{A.35})$$

which is the desired result. This concludes the proof of Lemma 5.3. $\square$