ONLINE NONLINEAR MODELING FOR BIG DATA APPLICATIONS

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

By
Farhan Khan
December 2017

ONLINE NONLINEAR MODELING FOR BIG DATA APPLICATIONS
By Farhan Khan
December 2017

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

Suleyman Serdar Kozat(Advisor)

Tolga Mete Duman

Selim Aksoy

Elif Vural

Murat Özbayoğlu

Approved for the Graduate School of Engineering and Science:

Ezhan Karaşan
Director of the Graduate School

ABSTRACT

ONLINE NONLINEAR MODELING FOR BIG DATA APPLICATIONS

Farhan Khan
Ph.D. in Electrical and Electronics Engineering
Advisor: Suleyman Serdar Kozat
December 2017

We investigate online nonlinear learning for several real life, adaptive signal processing and machine learning applications involving big data, and introduce algorithms that are both efficient and effective. We present novel solutions for learning from the data that is generated at high speed and/or have big dimensions in a non-stationary environment, and needs to be processed on the fly. We specifically focus on investigating the problems arising from adverse real life conditions in a big data perspective. We propose online algorithms that are robust against the non-stationarities and corruptions in the data. We emphasize that our proposed algorithms are universally applicable to several real life applications regardless of the complexities involving high dimensionality, time varying statistics, data structures and abrupt changes.

To this end, we introduce a highly robust hierarchical trees algorithm for online nonlinear learning in a high dimensional setting where the data lies on a time varying manifold. We escape the curse of dimensionality by tracking the subspace of the underlying manifold and use the projections of the original high dimensional regressor space onto the underlying manifold as the modified regressor vectors for modeling of the nonlinear system. By using the proposed algorithm, we reduce the computational complexity to the order of the depth of the tree and the memory requirement to only linear in the intrinsic dimension of the manifold. We demonstrate the significant performance gains in terms of mean square error over the other state of the art techniques through simulated as well as real data.

We then consider real life applications of online nonlinear learning modeling, such as network intrusions detection, customers' churn analysis and channel estimation for underwater acoustic communication. We propose sequential and online learning methods that achieve significant performance in terms of detection accuracy, compared to the state-of-the-art techniques. We specifically introduce structured and deep learning methods to develop robust learning algorithms. Furthermore, we improve the performance of our proposed online nonlinear learning

models by introducing mixture-of-experts methods and the concept of boosting. The proposed algorithms achieve significant performance gain over the state-of-the-art methods with significantly reduced computational complexity and storage requirements in real life conditions.

Keywords: Online learning, big data, boosting, channel estimation, sequential data processing,intrusion detection, underwater acoustics, channel estimation, language model, tree based methods, logistic regression, deterministic analysis, non-stationarity, curse of dimensionality, stream processing, time series.

ÖZET

BÜYÜK VERİ UYGULAMALARI İÇİN ONLINE NON LİNEER OLMAYAN MODELLEME

Farhan Khan
Elekrik ve Elekronik Mühendisliği, Doktora
Tez Danışmanı: Süleyman Serdar Kozat
Aralık 2017

Birkaç gerçek yaşam, uyarlamalı sinyal için çevrimiçi doğrusal olmayan öğrenmeyi araştırırız büyük veriler içeren işleme ve makine öğrenme uygulamaları ve hem etkin hem de etkili algoritmalar. Için yeni çözümler sunuyoruz yüksek hızda üretilen ve / veya büyük boyutlu olan verilerin öğrenilmesi, durağan olmayan bir çevrede yaşıyor ve anında işlenmesi gerekiyor. Özellikle olumsuz gerçek yaşam koşullarından kaynaklanan sorunları büyük bir veri açısından incelemeye odaklanıyoruz. Verilerdeki durgunluk ve bozulmalara karşı sağlam olan çevrimiçi algoritmalar önermekteyiz. Önerilen algoritmaların, yüksek boyutluluk, zamana bağlı istatistikler, veri yapıları ve ani değişiklikler içeren karmaşıklıklara bakılmaksızın, birkaç gerçek hayat uygulaması için evrensel olarak uygulanabileceğini vurguluyoruz.

Bu amaçla, verilerin zamanla değişen bir manifold üzerinde bulunduğu yüksek boyutlu bir ortamda çevrimiçi doğrusal olmayan öğrenme için oldukça sağlam hiyerarşik ağaçlar algoritması sunmaktayız. Altta yatan manifoldun altuzayını izleyerek boyutsallık lanetinden kaçarız ve doğrusal olmayan sistemin modellenmesi için modifiye regressor vektörleri olarak alttaki manifold üzerine orijinal yüksek boyutlu regressor uzayının projeksiyonlarını kullanırız. Önerilen algoritmayı kullanarak, hesaplama karmaşıklığını ağacın derinlik sırasına ve bellek gereksinimini yalnızca manifoldun öz boyutunda doğrusal olana indirgiyoruz. Simülasyona giren ve gerçek veriler aracılığıyla diğer teknolojik teknikler üzerindeki ortalama karesel hata açısından önemli performans artışı sergiliyoruz.

Ardından, çevrimiçi zorunlu öğrenme modellemesinin, ağ saldırıları tespiti, müşterilerin çarpma analizi ve sualtı akustik iletişimi için kanal tahmini gibi gerçek yaşam uygulamalarını ele alacağız. Son teknoloji tekniklerle karşılaştırıldığında, algılama hassasiyeti açısından önemli performans sağlayan sıralı ve çevrimiçi öğrenme yöntemleri önermekteyiz. Sağlam öğrenme algoritmalarını geliştirmek için yapılandırılmış ve derin öğrenme yöntemlerini

özellikle tanıtmaktayız. Ayrıca önerilen çevrimiçi nonlineer öğrenme modellerimizin performansını, uzmanların karışımı yöntemleri ve güçlendirme kavramı ile geliştiriyoruz. Önerilen algoritmalar, gerçek yaşam koşullarında önemli ölçüde azaltılmış hesaplama karmaşıklığı ve depolama gereksinimi ile en son teknoloji yöntemler üzerinde önemli bir performans kazancı elde etmektedir.

Anahtar sözcükler: Online öğrenme, büyük veri, artırma, kanal tahmini, sıralı veri işleme, saldırı tespiti, sualtı akustiği, kanal tahmini, dil modeli, ağaç tabanlı yöntemler, lojistik regresyon, deterministik analiz, durgunluk, boyutsallık laneti, akış işlemi, zaman serileri.

Acknowledgement

I thank my PhD advisor, Assoc. Prof. Dr. Süleyman Serdar Kozat, for his continuous guidance and support. This thesis wouldn't have been possible without his technical as well as moral advice.

I acknowledge the financial support given to me by Higher Education Commission (HEC) of Pakisan and The Scientific and Technological Research Council of Turkey (TÜBİTAK).

I thank coauthors Mr. Dariush Kari, Mr. İbrahim Delibalta and Mr. İlyas Alper Karatepe who helped in writing papers and providing data without which I would not have been able to publish my work.

I acknowledge the guidance and support I got from my teachers, specially Prof. Orhan Arıkan, Dr. Sinan Gezici and Dr. Selim Aksoy, whose patience and openness to discussions helped me understand difficult topics.

I thank my friends Adeel Shahriyar, Deepak Kumar Singh, M. Sabeeh Iqbal and Saeed Ahmed for their valuable suggestion regarding the thesis and general support throughout my time in PhD.

Finally, I would like to thank the most important people in my life, my wife Alia Khan and kids Shadan Khan, Zeydan Khan and Jannat Khan. Their exhibit of love has been keeping going through difficult times.

This thesis is dedicated to my parents, Mir Ajab Khan and Farhad Begum, and my brother Imran Khan for their endless love, support and encouragement.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Sequential and nonlinear learning problems are widely investigated in the machine learning [1–3], adaptive signal processing [4–8], neural networks [9–16], knowledge engineering [17–20] and cyber security [21–23, 48] literatures. However, the classic learning algorithms and techniques are inadequate for applications in adverse environments involving big data, missing information, time varying and nonlinear settings with abrupt changes, and varying and unknown statistical distribution of the input space [27, 28]. These adverse conditions cause significant performance loss and demand computationally complex models with large memory requirements [17, 18]. In modern applications including artificial intelligence, natural language processing, underwater acoustic communication and big data analysis, it is important to consider and directly work against these adverse conditions while developing learning and estimation algorithms. Usually, nonlinear models are considered for these applications where linear models are inadequate. However, nonlinear models usually suffer from over-fitting, stability and convergence issues [8–13].

As an example, for applications involving big data [17–19], for instance, when the input vectors are of high dimensions, the nonlinear modeling offers substantial challenges. These challenges include computational complexity increase, which is usually beyond manageable, and time varying statistical distributions [14]. In this work, we address these adversities one by one and investigate possible solutions using highly efficient models with substantial real life performance. We are interested in developing algorithms that are universal, adaptive, independent of the problem settings and assumptions, and perform as well as or better than the state-of-the-art methods for real world applications.

We note that, for high dimensional data, learning and regression on the manifolds are widely investigated in the current literatures [24, 25]. For instance, in network traffic [26], large amounts of high dimensional, time varying data from various nodes are used to identify certain trends. Another example is video surveillance [29] (which involves high dimensional, time varying images from various cameras). The high dimensional data from security cameras is analyzed for any mischievous activity in a sensitive area [30]. However, online regression on high dimensional data suffers from performance degradation and computational complexity, known as Bellman's *curse of dimensionality*, and is statistically challenging [14, 31–33]. The problem of manifold learning and regression is rather easy when all the data is available in advance (batch), and lies around the same single static submanifold [28]. In online manifold learning, however, it is difficult to track the variation in data because of the high dimensionality and time varying statistical distributions [27, 28]. Therefore, we introduce a comprehensive solution that includes online learning and adaptive regression over high dimensional feature vectors in a dynamic setting.

To this end, we first tackle the non-stationarity and high dimensionality using piecewise nonlinear models in the second chapter. We use an adaptive tree structure where a linear model is trained on each node of the tree and the combination of these node-wise linear models are used to model the nonlinearity. Furthermore, we use the projections on the low dimensional subspace of the tree nodes to escape the curse of dimensionality such that the new projection vectors are used as the input to the nonlinear learning model. In this sense, the adaptive hierarchical tree learns the underlying structure of the input space and hence reduce the dimensionality, and learn the nonlinearity. The algorithm is online and dynamically update all the model specifications with the new data samples. These specifications include the parameters of each linear regression model, the combination weights that depend upon the performance of each linear model, and the low dimensional subspace basis that varies according to the curvature of the input space. Hence, we achieve a highly robust learning model that is capable of tracking the nonlinearity and non-stationarity, and successfully escape the curse of dimensionality.

As a real life application of nonlinear modeling, we investigate recurrent neural networks (RNN) for network intrusion detection where the network data is highly erratic and arrives with astonishing rates [50] in the third chapter. Here, we try to detect whether a network payload is normal or anomalous by modeling the relationship in-between using a high dimensional, state dependent nonlinear structure. We use the analogy of a language model to generate a deep learning model using combination of Long Short Term Memory (LSTM) units and convolution neural networks (CNNs).

We then follow these discussions with another real life application of nonlinear modeling in high dimensions, where we investigate online churn detection in cellular networks in the fourth chapter. We analyze the sequential data, recorded in real life telecommunications network, of each cellular customer to introduce a nonlinear anomaly detector by using the methods discussed in the previous chapters. We then continue our discussion, in the fifth chapter, where we use boosting type ensemble methods to improve our nonlinear modeling algorithms. We specifically introduce online boosting methods that improve the robustness of nonlinear models suitable for highly non-stationary settings. We finally, in the sixth chapter, investigate online nonlinear modeling approach for the highly non-stationary underwater acoustic (UWA) channel estimation. Here, we seek algorithms that are robust against the variations in an UWA channel by introducing a logarithmic cost function.

1.1 Summary and Priot Art

For applications involving high dimensional data, various approaches are studied to escape the curse of dimensionality and perform online learning [19, 20, 32–37]. In [27], the authors performed online tracking of high dimensional data by approximating the underlying submanifolds as union of subsets of the high dimensional space. The low dimensional approximation is used as a pre-processing step for the change point detection [28] and logistic regression [38] for high dimensional time series. In our approaches, however, we use context trees to perform nonlinear regression, which adapts automatically to the intrinsic low dimensionality of the data by maintaining the "geodesic distance" [39, 40] while operating on the original regressor space. Note that context trees perform a hierarchical, nested partitioning of the regressor space for the piecewise linear models [41, 42]. However, unlike [27, 28, 38] where a single partitioning is used with varying depth, context trees construct a weighted average of all possible partitions defined on a tree [41, 42], and the partitioning in [27, 28, 38] is a subclass of our tree structure. We propose an algorithm where the weights assigned to each node as well as the partitions vary according to variations in the data. Furthermore, we use a piecewise linear model, i.e., different linear regression parameters in each region defined by the tree, whereas in [38], the same logistic regression model is used in each subset or region. In this manner, our algorithm inherently uses weighted combination of all possible partitions defined by trees of various depths, and compete well against a doubly exponential class yet with a complexity linear in the depth of the tree [42]. For instance, an adaptive hierarchical tree of depth K also inherently incorporates trees with depths less than K, i.e., $0, 1, ..., K-1$. This makes the algorithm suitable for various ranges of complexity in the data

structure. Thus, our proposed algorithm in this thesis not only inherently solves the curse of dimensionality in an adaptive and rigorous manner but also achieves such feats working in an online setting independent of the structural variation of the input data and the underlying system.

We consider network intrusion detection as a real life application of nonlinear modeling and erratic data analysis. We develop a deep learning model that sequentially analyze the network payloads and detect the anomalies. Apart from the complexities and challenges associated with the underlying relationship between input and target data in the learning process, there also exist the irregularities within the input data. For instance, in natural language processing, the main challenge is that the raw data is not in a readily machine readable format [43–46]. The text data comprises of complex structures, hierarchies, syntax and sequential information. It contains documents, sentences, words and characters for which the meaningful information is interlinked [45, 46]. Furthermore, the input data and the information it contains are of various types such as numbers, images, text, speech and distinct categories. Therefore, extensive preprocessing is required to generate "feature maps" in a machine learning perspective. For structural and sequential learning, deep learning approaches are widely investigated, that use multi-layer and multi-level "encoding" of the input data. We specifically seek to develop deep learning and sequential algorithm for the online learning of network traffic [23, 47–49]. We consider network intrusion detection systems as a special case of natural language processing by analyzing the raw network packets for malicious traffic. The information sent over a computer network usually consists of data packets corresponding to request codes and responses that contains the requested information. These packets are encoded according to a certain scheme, analogous to a language structure, with sequential and syntactic meaning [48,50]. Furthermore, from a rigorous cyber-security point of view, we are interested to analyze the data sent over a network in "real time" such as to prevent any damage from the malicious activity. Therefore, we seek to develop models and algorithms that are online, fast, efficient and can distinguish between normal and malicious traffic by sequential and deep analysis. We specifically use convolution neural networks, recurrent neural networks and sequential anomaly detection on quality benchmark datasets to develop a robust intrusion detection system [45, 51–53].

For this purpose, we apply our nonlinear modeling approaches to network intrusion detection systems (IDS) that are a core part of the cyber-security and information management systems. Such systems typically analyze the data traffic transmitted over a computer network for possible anomalies that may exploit the vulnerabilities and short comings of an apparently secure network [54]. The anomalous traffic are usually unauthorized activities in order to gather private information, utilization of network resources, or corrupting the data or resources. These anomalous activities are termed as attacks or intrusions. The intruders may

use such attacks for various illegal purposes such as credit card theft, destroying or taking down the network to cause financial harms to private or public entities. Network attacks are classified various categories, e.g., unauthorized access, denial of service, overuse of the resources, flooding etc. One of the most dangerous intrusion technique is distributed denial of service (DDoS) where the intruders use a large number of routes to attack the network, for instance, by sending floods of data. These intrusions are extremely dangerous, however, hard to detect since they can use automated processes and legal hosts to launch attacks. Other examples of network attacks use Trojans, malware, executable programs (viruses) and ransomware that are activated when received.

We employ online nonlinear anomaly detection methods suitable for non-stationary environments. There are primarily two approaches that are employed to design an intrusion detection system, i.e., signature detection and anomaly detection [56, 57]. In the former, the IDS are designed to "identify" of anomalous traffic according to certain properties of the already known attacks. These IDS uses definition of attacks, for instance an executable file can be identified as a malware or multiple login attempts can be considered as unauthorized access. However, signature detection fails to identify novel and previously unseen attacks which results in lower detection probability. Anomaly detection makes a good IDS system since it is trained to "know" the normal traffic and would raise an alert for intrusion if anything happens beyond the "normal" boundaries. Anomaly detection techniques may identify a novel but authorized traffic as attack, therefore generating a false alarm. A good IDS must be capable of detecting novel attacks, however, raising minimal false alarms. Therefore, the threshold or boundaries between the normal and anomalous instances must be carefully determined and may change over time. We specifically design and develop algorithms for a robust IDS in a machine learning perspective that achieves significant performance in terms of high true positive and low false alarm rates. Furthermore, we emphasize that our algorithms must be fast and work in real time so as to reduce the damage from attack as much as possible.

We next consider customers' churn detection as real life application of the online nonlinear learning models. We apply the online and sequential learning algorithms for logistic regression to cellular network customers data in order to predict their probability of leaving or retaining the network [58–60]. It is important for service providers to know about the customers' satisfaction from their services. Therefore, the companies track the data of users for possible anomalies that may result in leaving their network for the competitors. The cellular network companies may then use this information to provide better services and overcome any issues to retain their valuable customers. We investigate the churn analysis under the framework where the users data is periodically collected and analyzed in real time for anomalies. For instance, a customer record of calls, duration,

frequency, internet usage statistics, billing, disconnection issues might be analyzed to decide whether there is a high probability of attrition in the near future. In a data driven framework, knowledge of the previously churned customers is analyzed to identify the patterns.

Subsequently, we use the ensemble approaches to improve the performance of our nonlinear modeling algorithms. Specifically, we use online boosting to further improve the performance and robustness of adaptive and piecewise nonlinear model based on hierarchical tree structure. Boosting is considered as one of the most important ensemble learning methods in the machine learning literature and it is extensively used in several different real life applications from classification to regression [61–68]. As an ensemble learning method [69–75], boosting combines several parallel running "weakly" performing algorithms to build a final "strongly" performing algorithm [76–78]. Conventionally, boosting has been used as a batch learning algorithm, i.e., it has been applied over a fixed length of training that had been available in advance [79]. However, recently the notion of boosting is extended to online setting [80,81]. We neither need nor have access to a fixed set of training data, since the data samples arrive one by one as a stream, in the online settings [74, 82]. The arriving data samples are readily processed and it is not required to store large chunks of data. Therefore, the online learning is memory efficient and most suitable for real life applications involving big data where storage space is an issue as well as the problem at hand require instant processing [83]. For all that, we focus on the online boosting framework and introduce several algorithms for online learning tasks. Furthermore, since our algorithms are online, they can be directly used in adaptive filtering applications to improve the performance of conventional mixture-of-experts methods in a non-stationary environment [84]. For adaptive filtering purposes, the online setting is especially important, where the sequentially arriving data is used to adjust the internal parameters of the filter, either to dynamically learn the underlying model or to track the non-stationary data statistics [4, 84].

As a real life application of online learning in a non-stationary environment, we consider the channel estimation of underwater acoustic communication. There has been a recently growing interest in Underwater acoustic (UWA) communication due to the advent of new and exciting applications such as marine environmental monitoring and sea bottom resource exploitation [85–87]. However, UWA channel is acutely affected by several adversities because of the constant movement of waves. These adverse effect are multi-path fading, Doppler spreads and frequency dependent propagation loss, to name a few [87, 88]. These adversities make UWA channel as one of the most hostile communication mediums [88, 89]. In these mediums, channel equalization [90, 91] plays a key role in providing reliable high data rate communication [87]. Most recently, orthogonal frequency division multiplexing (OFDM) is employed to mitigate the effects of long and

time varying channel impulse response (CIR) [92]. However, OFDM requires an accurate estimate of the CIR for the channel equalization and to "remove" the effects of time varying CIR. Since the channel is non-stationary due to fast variations in the underwater environment, the the estimation algorithms need to be online and adaptive [87]. Furthermore, the occasional but high amplitude impulsive noise makes the estimation process unstable. Therefore, we thoroughly analyze the hostile UWA channel and seek robust estimation algorithms that can adapt to the variations in UWA channels.

1.2 Outline

The thesis is organized as follows. In Chapter 2, we consider the problem of high dimensional input and nonlinear relationship between input and output in machine learning and regression frameworks. We use a deterministic framework where the algorithms are independent of the statistical distribution and suitable for a non-stationary environment. We propose novel algorithm that incorporate adaptive and dynamic learning of the underlying low dimensional structure of the input space (manifold learning) in an online learning environment [93]. We provide mathematical justification of the performance and computational complexities of the proposed algorithm by the use of theorems. Furthermore, we provide the performance results on several synthetic as well as real high dimensional datasets. We compare the performance of the proposed algorithm, in terms of mean square error (MSE) and processing time, with several state-of-the-art algorithms.

In Chapter 3, we investigate, as real life application of nonlinear modeling, network intrusion detection. Here, we investigate the sequential and contextual learning of the network packets using deep neural networks. We specifically consider intrusion detection in data driven and machine learning framework. We propose to use deep learning algorithms that work on the network payloads. These payloads are sequence of characters where we identify their respective structure, syntax and context in a language model perspective. We use recurrent and multilevel neural networks such as LSTMs and CNNs for character and sequence level encoding of the payload [53, 94, 95]. We experiment with several combinations of parameters such as number and types of RNN and CNN layers, number of units in each layer and the "depth" of the neural network model. We provide the performance measures of our proposed models using several performance metrics such as true/false positive rates, receiver operating characteristics, cross validation and area under the curve (AUC) scores [96, 97].

In Chapter 4, we model a highly nonlinear relationship between customers' data and their decision to churn, considering adaptive and piecewise learning. We apply the algorithms developed in Chapters 2 and 3 to real life cellular network users' data for churn prediction. Furthermore, we use deep learning approach to exploit the sequential nature of the data and predict the churn probability for the future. We specifically use multi-layer LSTM network and logistic regression for the churn analysis. Moreover, there are several missing features throughout the dataset that we address by padding and applying a masking layer along with our RNN layers. In this manner, the parameters of LSTM networks are not updated when the input is masked. We perform several experiments and draw comparisons with the other machine learning and prediction algorithms using metrics such as ROC and AUC scores.

In Chapter 5, we investigate ensemble learning approaches to further improve our nonlinear modeling developed in previous chapters. We consider the ensemble learning methods and boosting approaches for robust and dynamic learning with performance guarantees for online learning . We introduce novel algorithms that are suitable for online big data in a highly nonlinear environment. The resultant methods can dynamically learn the non-stationary structure of the data as well as the regression parameters without any a priori knowledge. We support our claims by strong mathematical justifications and theorems giving performance bounds. We apply the proposed boosting approaches on several real and simulated datasets. We provide results that significantly outperforms the ensemble learning and boosting algorithms in literature.

In Chapter 6, we consider the problem of abrupt changes and non-stationary data in an online learning environment, specifically for the channel estimation in underwater acoustic communication [86–89]. We propose robust algorithms that adapt to the abrupt changes by incorporating various cost functions. We specifically propose to add a logarithmic term for regularization to the conventional cost function such that in the absence of high impulse noise the logarithmic term is added as a correction and yields faster convergence. In the case of impulsive noise, the correction term diminishes and provides better robustness. Furthermore, we investigate first and second order estimation / filtering methods that incorporates logarithmic cost function. In this manner, we develop a family of robust channel estimation algorithms with mathematical justification for error and complexity bounds. We show a significant performance improvement over the previously proposed algorithms by providing results on several realistic simulations.

Finally, we conclude the thesis with closing remarks in Chapter 7.

Chapter 2

Universal Nonlinear Regression on High Dimensional Data using Adaptive Hierarchical Trees

In this chapter, we investigate the problem of online and nonlinear learning in a high dimensional setting. We specifically introduce an adaptive algorithm to escape the curse of dimensionality and approximate the nonlinear model by a non-stationary piecewise model. We study nonlinear regression using high dimensional data assuming that the data lies on a manifold. We partition the regressor space into several regions to construct a piecewise linear model as an approximation of the nonlinearity between the observed and the desired data. However, instead of fixing the boundaries of the regions, we partition the space in a hierarchical manner [98]. We use the notion of context trees [41,42] to represent a broad class of all possible partitions for the piecewise linear models. We specifically introduce an algorithm that incorporates context trees for online learning of the high dimensional manifolds and perform regression on the big data. In this approach, regression directly adapts to the intrinsic lower dimension of the data while operating in the original regressor space. The algorithm achieves the performance of the best partitioning of the regressor space, competing against a broader class of piecewise linear algorithms. This broader class consists of various partitioning methods, region boundaries and regression algorithms for each region as explained later in the chapter.

In the domain of online nonlinear regression, context trees have been used to partition the regressor space hierarchically, and to construct a competitive algorithm among a broader class of algorithms [42]. Although we also use the context tree weighting of Willems et. al [41] as [42], there are major differences

between our method and the method in [42]. In contrast to nonlinear regression using context trees, we use a hierarchical tree structure to track and learn the manifold in a high dimensional setting. We use the regions defined by the tree to learn the underlying low dimensional projection and perform piecewise linear regression whereas in [42], the tree is used to learn the actual piecewise regions where the data lie in a $D-$dimensional space. Furthermore, the regions defined by our algorithm are time varying ellipsoids and a parent region on the tree is not necessarily a union of its children. Unlike [42], the actual data does not belong to the regions defined by the tree and we use a quadratic distance measure to decide on the membership of a data instance to a certain region. Moreover, for the test data, where the desired labels are not available, the algorithm in [42] does not update the tree structure as well as the node estimators. However, in our algorithm, we update the node performance measure by the tracking performance of the submanifold structure in the observed data. We finally use the projection of the actual data on the low dimensional regions for the regression. In addition to solving the problem of high dimensionality by incorporating manifold learning, our algorithm also performs online piecewise regression.

In this chapter, we first introduce an algorithm that is guaranteed to asymptotically achieve the performance of the best combination of a doubly exponential number of different models that can be represented by a depth-K tree with computational complexity only linear in the depth of the tree. We use a tree structure to hierarchically partition the high dimensional regressor space. We then incorporate an approximate Mahalanobis distance as in [27,28,38] to adapt the regressor space to its intrinsic lower dimension.

The Mahalanobis distance is a measure of the distance between a point P and a distribution D, which is unit-less and scale-invariant (unlike the Euclidean distance), and takes into account the correlations in the data set. Therefore, for general classification of data points, the Mahalanobis distance is shown to perform better than the Minkowski distances [102]. Furthermore, we can effectively track the true curvature of each submanifold by using the Mahalanobis distance instead of the Euclidean distance (as a special case of the Minkowski distance [102]), since the Mahalanobis distance takes into account the spreading of data in each direction (resulting in a non-symmetric ellipsoid for each submanifold). Our algorithm also adapts to the corresponding regressors in each region to minimize the final regression error. We then prove that as the data length increases, the algorithm achieves the performance of the best partitioning by providing an upper bound on the performance of the algorithm. We show that the method used is truly sequential and generic in the sense that it is independent of the statistical distribution or structure of the data. Moreover, the algorithm does not presume the structure or variation of the manifolds and adapts to the underlying data sequentially. In this sense, the algorithm learns i) the structure of the manifolds,

ii) the structure of the tree, *iii*) the low dimensional projections in each region, *iv*) the linear regressors in each region, and *v*) the linear combination weights of all possible partitions, to minimize the final regression error. We also show that a perfect manifold tracking for the regression purpose is not only unnecessary but also increases the complexity of the algorithm and may even increase the final error due to overfitting. This is in contrast to [28, 38] where the ultimate goal is to reduce the tracking error as much as possible.

In next section, we formally describe the problem and setting. In Section 2.2, we develop and propose the adaptive hierarchical trees (AHT) algorithm. In Section 2.3, we show the results of several experiments on real as well as synthetic datasets. We conclude the chapter with a short discussion on the performance of proposed algorithm in Section 2.4.

2.1 Problem Description

All vectors used in this chapter are column vectors, denoted by boldface lowercase letters. Matrices are denoted by boldface uppercase letters. For a vector $\boldsymbol{v}$, $\|\boldsymbol{v}\|^2 = \boldsymbol{v}^T\boldsymbol{v}$ is squared Euclidean norm and $\boldsymbol{v}^T$ is the ordinary transpose. $\boldsymbol{I}_k$ represents a $k \times k$ identity matrix.

We investigate online nonlinear regression using high dimensional data, i.e., when the dimension of data $D \gg 1$. We observe a desired sequence $\{y[n]\}_{n\geq 1}$, $y[n] \in \mathbb{R}$, and regression vectors $\{\boldsymbol{x}[n]\}_{n\geq 1}$, $\boldsymbol{x}[n] \in \mathbb{R}^D$, where D denotes the *ambient dimension*. The data $\boldsymbol{x}[n]$ are measurements of points lying on a submanifold $S_{m[n]}$, where the subscript $m[n]$ denotes the time varying manifold, i.e., $\boldsymbol{x}[n] \in S_{m[n]}$. The *intrinsic dimension* of the submanifolds $S_{m[n]}$ are d, where $d \ll D$. The submanifolds $S_{m[n]}$ can be time varying. At each time n, a vector $\boldsymbol{x}[n]$ is observed. The estimate of the desired sequence $y[n]$ is given by:

$$\hat{y}[n] = f_n(\boldsymbol{x}[n]), \tag{2.1}$$

where $f_n(.)$ is a nonlinear, time varying function. The instantaneous regression error is given by: $e[n] = y[n] - \hat{y}[n]$.

The nonlinear model of (2.1) could establish a perfect fit to the underlying relationship between the desired and observed data in certain situations. However, identifying this nonlinear relationship could be challenging, and it may be unnecessary and computationally complex to use the perfect model [32]. Furthermore, the nonlinear model of (2.1) may suffer from overfitting, stability and convergence issues [8]. Therefore, we use a piecewise linear model as an approximation

of the nonlinear relationship between the observed sequence and the desired data. We begin our discussion of piecewise linear modeling with a fixed partitioning of the regressor space, i.e., $\mathbb{R}^D$. We next use the context tree algorithm to include arbitrary partitions from a large class of possible partitions, as explained later in the sub-section 2.1.1. Finally, we extend our results to the case when the input data is high dimensional.

In piecewise linear modeling, we partition the regressor space into J regions, where a linear relationship is assumed between the desired data and the observed data within each region. Since the statistical distribution of data and the dimension of regressor space vary with time, our partitioning method as well as the linear model in each region should be dynamic. To this end, we use a tree structure to hierarchically partition the regressor space. One such tree structure to partition a two dimensional regressor space is shown in Fig. 2.1. Here, a depth-2 tree is used to partition the $\mathbb{R}^2$ regressor space, i.e., $D = 2$ for this figure. We define a "partition" of the D-dimensional regressor space as a specific partitioning $\mathcal{P}_i = \{R_{i,1}, ..., R_{i,J_i}\}$, where $\bigcup_{j=1}^{J_i} R_{i,j} = \mathcal{R}$, $R_{i,j}$ is a region in the D-dimensional regressor space and $\mathcal{R} \in \mathbb{R}^{\mathrm{D}}$ is the complete D-dimensional regressor space. Fig. 2.1 shows all possible partitionings of the two dimensional regressor space with a tree of depth-2. In general, for a tree of depth K, there are as many as 1.5^{2^K} possible partitions, $\mathcal{P}_i$, where $i \in \{1, ..., 1.5^{2^K}\}$. Each of these doubly exponential number of partitions can be used to construct a piecewise linear model.

To clarify the notation, as an example, we consider a sample partition $\mathcal{P}_3$ in Fig. 2.1, where the regressor space is divided into three regions. At j^{th} region of a specific partition, e.g., $\mathcal{P}_3$, we generate the estimate:

$$\hat{y}_j[t] = \boldsymbol{x}^T[t]\boldsymbol{v}_j[t], \tag{2.2}$$

where $\boldsymbol{v}_j[t]$ is the regressor weight vector for the j^{th} region, $t = \{n; \boldsymbol{x}[n] \in R_j\}$, $j \in \{1, ..., J\}$ and $J = 3$ is the number of regions the regressor space is divided into by the partition $\mathcal{P}_3$. The final estimate of $y[n]$ is given by:

$$\hat{y}_{\mathcal{P}_3}[n] = \boldsymbol{x}^T[n]\boldsymbol{v}_j[n], \tag{2.3}$$

for $j \in \{1, 2, 3\}$ when $\boldsymbol{x}[n] \in R_j$. For a fixed partition, e.g., $\mathcal{P}_3$, we try to achieve the performance of the best piecewise linear regressor. We then extend these results to all possible partitions.

We try to achieve the performance of the best piecewise linear model when there is a large class of possible partitions of the regressor space, i.e., over all $\mathcal{P}_i$. We specifically minimize the following regret over any n [42]:

$$\sum_{t=1}^{n}(y[t] - \hat{y}_q[t])^2 - \inf_{\mathcal{P}_i} \sum_{t=1}^{n}(y[t] - \hat{y}_{\mathcal{P}_i}[t])^2, \tag{2.4}$$

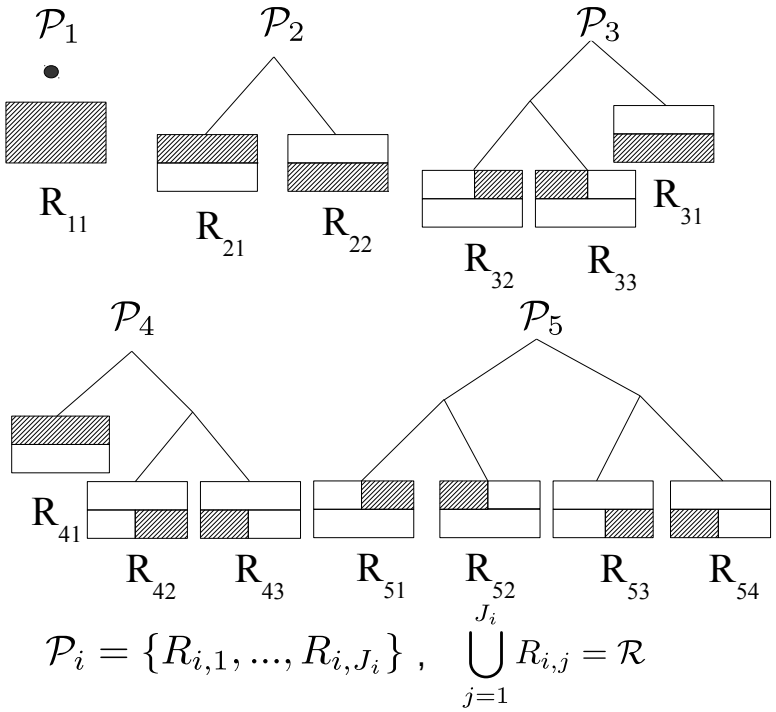


Figure 2.1: A full tree of depth 2 that represents all possible partitions of the two dimensional space, $\mathcal{P} = \{\mathcal{P}_1, ..., \mathcal{P}_{N_K}\}$ and $N_K \approx (1.5)^{2^K}$, where K is the depth of the tree. Here $N_K = 5$.

where $\hat{y}_{\mathcal{P}_i}[t]$ is the estimation of $y[t]$ from the partition $\mathcal{P}_i$, $i = 1, ..., 1.5^{2^K}$ and $\hat{y}_q[i]$ is the estimation from a sequential algorithm. We seek a sequential algorithm that can estimate the desired sequence $y[n]$ from $\boldsymbol{x}[n]$, as well as the best piecewise linear model. However, instead of brute-forcing over all possible partitionings, we seek an algorithm with linear complexity in the depth of the tree. For this purpose, we use a context tree approach [41, 42].

2.1.1 Context Tree Algorithm for Piecewise Linear Regression

The context tree algorithm achieves the performance of the best partition among the doubly exponential class of partitions with a complexity linear in the depth of the tree [42]. We use a full tree of depth K with up to 2^K finest partition bins as shown in Fig. 2.2. Each node $\eta = 1, ..., 2^{K+1} - 1$ on the tree represents a certain region among the $2^{K+1} - 1$ regions. The 2^K nodes corresponding to the finest partitioning of the regressor space are called the leaf nodes. The union

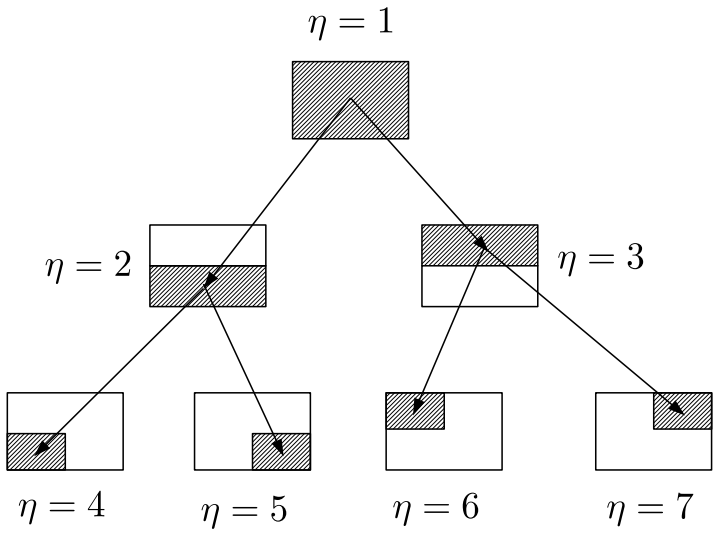


Figure 2.2: A two dimensional context tree of depth 2

of two leaf nodes η_l and η_u is a node one level above these nodes and is called the parent node of η_l and η_u, i.e., $R_{\eta_p} = R_{\eta_l} \cup R_{\eta_u}$. If a data sample $\boldsymbol{x}[n] \in R_{\eta^*}$, where η^* is one of the leaf nodes, it also belongs to the ancestor nodes of η^*. In principle, $\boldsymbol{x}[n]$ is an element of a single node on each level of tree. These $K+1$ nodes are called the "dark nodes". We define the set of dark nodes by $\mathcal{K} \triangleq \{\eta^*, \texttt{fix}(\eta^*/2), \texttt{fix}(\eta^*/4), ..., 1\}$, where we use the MATLAB notation `fix`(.) that rounds off the expression to the nearest integer towards zero. Linear regression is applied on each dark node and the final estimate is computed as a weighted combination of estimates from each of these $K+1$ regressors,

$$\hat{y}[n] = \sum_k \omega_k[n-1]\hat{y}_k[n], \tag{2.5}$$

where $k \in \mathcal{K}$, the weights $\omega_k[n-1]$ correspond to the performance of each node in the past and the regression error is used as a performance measure [42]. Here, $\hat{y}_k[n]$ is the estimate of $y[n]$ by the regressor of node k. As an example, in the beginning of the adaptation when there is a small amount of input data available, the nodes on the upper level of the tree may perform better and are given more weights [41, 42]. The calculation and update of these weights is explained in Section 2.2.2.

However, if the input regressor vectors are high dimensional, i.e. $D \gg 1$, the regression process can be challenging due to the curse of dimensionality [14, 32].

Hence, we dynamically map the high dimensional input vectors to lower dimension for the regression. We introduce an algorithm that performs piecewise linear regression in the high dimensional setting while inherently exploiting the underlying manifold structure. Furthermore, in contrast to the context tree algorithm, where regions of each partition are fixed, we learn these regions dynamically according to the submanifold variation in the data. In the following sections, we propose an algorithm that uses adaptive hierarchical trees tuned to the dynamics of the data and performs piecewise linear regression.

2.2 Manifold Learning and Regression using Adaptive Hierarchical Trees

To escape the curse of dimensionality, we perform regression on high dimensional data by mapping the regressor vectors to low dimensional projections. We assume that the observed data $\boldsymbol{x}[n] \in \mathbb{R}^D$ lies on time varying submanifolds $S_{m[n]}$. We can solve the problem of nonlinear regression by using piecewise linear modeling as explained in Section 2.1.1, where the regressor space, i.e., $\mathbb{R}^D$ can be partitioned into several regions. However, in the new setting, since the data lies on submanifolds with a lower intrinsic dimension, we use the lower dimensional projections instead of the original $\mathbb{R}^D$ regressor space. We define the piecewise regions in $\mathbb{R}^d$ for each node that correspond to the low dimensional submanifolds. However, since the submanifolds are time varying, the regions are not fixed. We define these regions by the subsets [28, 38]:

$$\begin{aligned}\Re_j[n] = \{\boldsymbol{x}[n] \in \mathbb{R}^D : \boldsymbol{x}[n] = \boldsymbol{Q}_j[n]\boldsymbol{\beta}_j[n] + \boldsymbol{c}_j[n], \\ \boldsymbol{\beta}_j^T[n]\boldsymbol{\Lambda}_j^{-1}[n]\boldsymbol{\beta}_j[n] \leq 1, \boldsymbol{\beta}_j[n] \in \mathbb{R}^d\},\end{aligned} \tag{2.6}$$

where each subset $\Re_j[n]$ is a $d-$dimensional ellipsoid assigned to each node of the tree. The matrix $\boldsymbol{Q}_j[n] \in \mathbb{R}^{\mathrm{D}\times\mathrm{d}}$ is the subspace basis in $d-$dimensional hyperplane and the vector $\boldsymbol{c}_j[n]$ is the offset of the ellipsoid from the origin. The matrix $\boldsymbol{\Lambda}_j[n] \triangleq \mathrm{diag}\{\lambda_j^{(1)}[n], ..., \lambda_j^{(d)}[n]\}$ with $\lambda_j^{(1)}[n] \geq ... \geq \lambda_j^{(d)}[n] \geq 0$, contains the eigen-values of the covariance matrix of the data $\boldsymbol{x}[n]$ projected onto each hyperplane. The subspace basis $\boldsymbol{Q}_j[n]$ specify the orientation or direction of the hyperplane and the eigen-values specify the spread of the data within each hyperplane [28, 38].

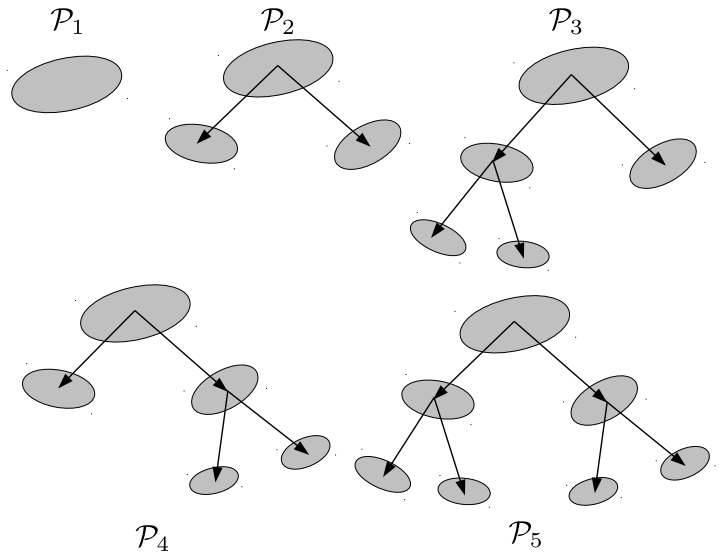
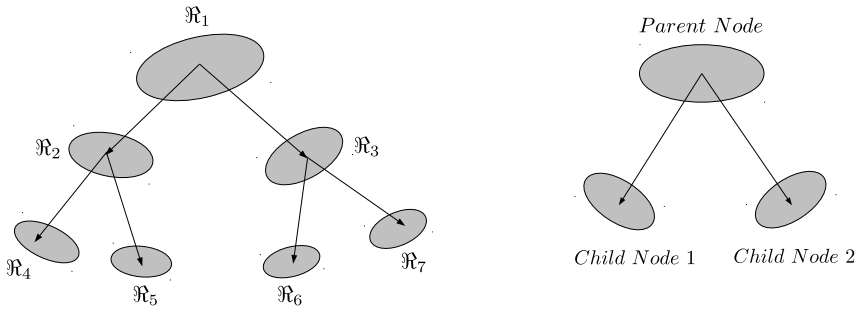


Figure 2.3: A doubly exponential number of partitions defining the piecewise models on time varying submanifold, each leaf node represents a subset defined by (2.2)

The projections of $\boldsymbol{x}[n]$ on the basis $\boldsymbol{Q}_j[n]$ are used as new regression vectors, $\tilde{\boldsymbol{x}}_j[n] = \boldsymbol{Q}_j^T[n]\boldsymbol{x}[n]$ and $\boldsymbol{\beta}_j[n] = \tilde{\boldsymbol{x}}_j[n] - \boldsymbol{c}_j[n]$. We represent these regions by a tree structure, where the leaf nodes correspond to these regions.

We use a tree structure to learn the underlying time varying manifold structure and the best piecewise linear model, however, instead of using Fig. 2.3, we use the notion of context trees given in Fig. 2.2 that represents the doubly exponential class in an efficient manner. We emphasize that each node on the tree is assigned a particular subset $\Re_\eta$, with $\eta \in \{1, ..., 2^{K+1} - 1\}$, in a hierarchical manner. However, unlike a regular tree introduced in Section 2.1.1, in this framework, the subset belonging to the parent node is not the union of its children nodes. The subset belonging to the parent node does not cover the space spanned by its two children and may not be in the same space as shown in Fig. 2.4(a). Moreover, the subsets defined by the nodes of the tree are low dimensional submanifolds while the actual data is of high dimension. In this sense, we can update the tree according to the variation in the observed data. We next use adaptive hierarchical trees (AHT) for the partitioning of high dimensional regressor space and learning the submanifolds. We show that the proposed algorithm significantly improves the performance of piecewise linear regression operating in the high dimensional setting.

The adaptive hierarchical tree (AHT) shown in Fig. 2.4(b) partitions the time varying manifold into $d-$dimensional subsets. However, the regions belonging to



(a) A dynamic hierarchical tree of depth K

(b) A parent node and its two children in an adaptive hierarchical tree.

Figure 2.4: A dynamic hierarchical tree of depth K where each η represents a subset defined by (2.2) and $\eta \in \{1, 2, ..., 2^{K+1} - 1\}$. Each node (subset) is a two dimensional ellipsoid defined by its parameters $\{\boldsymbol{Q}_\eta[n], \boldsymbol{\Lambda}_\eta[n], \boldsymbol{c}_\eta[n]\}$.

each subset are not fixed. Each node of the tree, $\eta \in \{1, ..., 2^{K+1} - 1\}$, corresponds to a $d-$dimensional ellipsoid, $\Re_\eta[n]$, with parameters $\{\boldsymbol{Q}_\eta[n], \boldsymbol{\Lambda}_\eta[n], \boldsymbol{c}_\eta[n]\}$ as defined in (2.2). These subsets are evolving in time according to the dynamics of the data, hence their parameters, $\{\boldsymbol{Q}_\eta[n], \boldsymbol{\Lambda}_\eta[n], \boldsymbol{c}_\eta[n]\}$, are adaptively updated with time. In the following, we explain that instead of updating all the subsets, we only update the recent $K + 1$ subsets that are defined by the dark nodes defined in sub-section 2.1.1. Each subset partially contributes to approximate the underlying submanifold with the leaf nodes representing finer approximations. The levels of the tree are chosen dynamically according to the curvature of the submanifolds. However, at a certain time n, instead of choosing a single level, we use the context tree weighting method [41] to assign weights to the subsets on each level, i.e., we use all possible partitions. We assign more weight to the children node when there is more curvature in the submanifold. The nodes of the tree structure shown in Fig. 2.4(b) represent ellipsoids that may all be in different $d-$dimensional subspaces.

The actual data samples $\boldsymbol{x}[n] \in \mathbb{R}^D$ do not lie in the space of $\Re_\eta[n]$ as $\Re_\eta[n] \in \mathbb{R}^d$. Therefore, we seek to determine the nearest region among the subsets in terms of a certain distance measure. After selecting the nearest regions, we then use the projection of $\boldsymbol{x}[n]$ on the specific region as our regressor input vectors,

$$\tilde{\boldsymbol{x}}_\eta[n] = \boldsymbol{Q}_\eta^T[n]\boldsymbol{x}[n], \tag{2.7}$$

where $\boldsymbol{Q}_\eta[n]$ is the basis for the region $\Re_\eta[n]$. We proceed to use the context tree algorithm for piecewise linear regression, given in [41, 42] that is briefly explained in Section 2.1.1.

With the arrival of a new data sample $\boldsymbol{x}[n]$, we determine which region $\Re_\eta[n]$ among the leaf nodes $\eta \in \{2^K, ..., 2^{K+1}-1\}$ it is nearest to by calculating the quadratic distance between $\boldsymbol{x}[n]$ and $\Re_\eta$, i.e.,

$$\eta^* = \arg\min_{\eta} D_M(\boldsymbol{x}[n], \Re_\eta[n]), \tag{2.8}$$

where $D_M(\boldsymbol{x}[n], \Re_\eta[n])$ is the distance between $\boldsymbol{x}[n]$ and the subset $\Re_\eta[n]$, $\eta = 2^K, ..., 2^{K+1}-1$. We use the approximate Mahalanobis distance [27, 28, 38] as the distance measure. The approximate Mahalanobis distance is defined by,

$$D_M(\boldsymbol{x}, \Re) \triangleq \delta(\boldsymbol{x}-\boldsymbol{c})^T \boldsymbol{Q}_1 \boldsymbol{\Lambda}_1^{-1} \boldsymbol{Q}_1^T (\boldsymbol{x}-\boldsymbol{c}) + \|\boldsymbol{Q}_2^T(\boldsymbol{x}-\boldsymbol{c})\|^2, \tag{2.9}$$

where $\boldsymbol{\Lambda}_1 = \text{diag}\{\lambda^{(1)}, ..., \lambda^{(d)}\}$, $\lambda^{(1)}, ..., \lambda^{(d)}$ are the largest d eigenvalues of the covariance matrix of $\boldsymbol{x} \in \Re$ with mean $\boldsymbol{c}$, and $\boldsymbol{Q}_1$ is the corresponding eigenvector matrix and is the subspace basis for the $d-$dimensional hyperplane, $\delta > 0$ is the average of the remaining eigenvalues, typically a small number and $\boldsymbol{Q}_2$ is the corresponding eigenvector matrix representing the residual subspace basis [28]. We use the minimum distance node, η^*, as the $\{K+1\}^{th}$ dark node in the context tree algorithm [42] and the rest of K dark nodes are the ancestor nodes of η^* till the root node $\eta = 1$. For instance, in Fig. 2.4(b), if $\Re_{\eta^*} = \Re_7$, then the remaining K dark node regions are $\Re_3$ and $\Re_1$. We then project the observed sample $\boldsymbol{x}[n] \in \mathbb{R}^{\text{D}}$ on the basis of each dark node k for $k \in \mathcal{K}$, the set of dark nodes defined in sub-section 2.1.1, i.e.,

$$\tilde{\boldsymbol{x}}_k[n] = \boldsymbol{Q}_k^T[n] \boldsymbol{x}[n], \tag{2.10}$$

where $\tilde{\boldsymbol{x}}_k[n] \in \mathbb{R}^{\text{d}}$. We train a linear regressor using each $\tilde{\boldsymbol{x}}_k[n]$ to learn the regressor weight vectors and estimate $y[n]$:

$$\boldsymbol{w}_k[n] = \boldsymbol{w}_k[n-1] + \nu \tilde{\boldsymbol{x}}_k[n](y[n] - \boldsymbol{w}_k^T[n-1]\tilde{\boldsymbol{x}}_k[n]), \tag{2.11}$$

where ν is the step-size of the Least Mean Square (LMS) algorithm and $\boldsymbol{w}_k[n] \in \mathbb{R}^{\text{d}}$ is the regressor weight vector for node k, $k \in \mathcal{K}$. The estimate of $y[n]$ from each dark node regressor is given by:

$$\hat{y}_k[n] = \boldsymbol{w}_k^T[n] \tilde{\boldsymbol{x}}_k[n]. \tag{2.12}$$

Finally, we use the context tree weighting method to estimate $y[n]$ as a weighted combination of the estimates of the dark nodes using (2.5). The complete algorithm is given in Algorithm 2.

Algorithm 1: Parameters and Initialization

Variables:

$\eta = 1, ..., 2^{K+1} - 1$: All node indices, complete tree of depth K.

$C_\eta[n-1] \triangleq \tilde{C}_\eta(y^{n-1})$: Total node confidence

$E_\eta[n-1] \triangleq \exp\left(-\frac{1}{2a}\sum_{t=1}^{n_\eta - 1}(y_\eta[t] - \boldsymbol{w}_\eta^T[t-1]\tilde{\boldsymbol{x}}_\eta[n])^2\right)$: Prediction performance of node η

$y_\eta[n-1] \triangleq \boldsymbol{w}_\eta^T[n-1]\boldsymbol{\beta}_\eta[n]$: Prediction of node η for $y[n]$

$\boldsymbol{\beta}_\eta[n] = \boldsymbol{Q}_\eta^T[n](\boldsymbol{x}[n] - \boldsymbol{c}_\eta[n])$

$\tilde{\boldsymbol{x}}_\eta[n] = \boldsymbol{Q}_\eta^T[n]\boldsymbol{x}[n]$: Projection of $\boldsymbol{x}$ on the basis of η

$\boldsymbol{\gamma}[n] = (\boldsymbol{I} - \boldsymbol{Q}_\eta[n]\boldsymbol{Q}_\eta^T[n])(\boldsymbol{x}[n] - \boldsymbol{c}_\eta[n])$: Projection residual

$\boldsymbol{w}_\eta[n] = \boldsymbol{w}_\eta[n-1] + \nu\bar{\boldsymbol{x}}_\eta[n](y[n] - \boldsymbol{w}_\eta^T[n-1]\bar{\boldsymbol{x}}_\eta[n])$: regressor weight vectors for each node η. $\boldsymbol{w}_\eta[n] \in R^d$

$\delta, \delta_1, \delta_2$: small positive real constants

$\boldsymbol{d}(k)$: k^{th} component of vector $\boldsymbol{d}$

$\boldsymbol{Q}_\eta$: basis of the subspace with node index η

$\Re_\eta[n]$: Subsets of the regressor space in R^D

$D_M(\boldsymbol{x}[n], \Re_\eta[n]) = \boldsymbol{\beta}_\eta^T[n]\boldsymbol{\Lambda}_\eta^{-1}[n]\boldsymbol{\beta}_\eta[n] + \delta_\eta^{-1}[n]\|\boldsymbol{x}_\perp[n]\|^2$

$\eta_l = 1, ..., 2^K$: Leaf nodes

$\eta^* = \arg\min_{\eta_l} D_M(\boldsymbol{x}[n], \Re_{\eta_l}[n])$

Initialization:

for $\eta = 1$ to $2^{K+1} - 1$; **do**

  $C_\eta[0] = \delta_1^{-1}$

  $E_\eta[0] = \delta_2^{-1}$

  $\hat{y}_\eta[0] = 0$

  $\boldsymbol{w}_\eta[0] = \boldsymbol{0}$ (initial weight vector for node η)

end for

for $k = 1, ..., K+1$; **do**

  $\mu_k[0] = 0, \sigma_k[0] = 0$

  $\boldsymbol{\Lambda}_\eta[0] = diag\{\lambda_1, ..., \lambda_d\}$: here $\lambda_1, ..., \lambda_d$ are the eigen-values of covariance matrix Σ of initial training samples.

  $\boldsymbol{c}_\eta[0] = \mathbb{E}\{\boldsymbol{x}\}$: $\boldsymbol{x}$ are the training samples and $\boldsymbol{x} \in R_\eta$

  $\boldsymbol{Q}_\eta[0]$ = eigen-vector matrix of covariance matrix Σ

end for

Algorithm 2: Main Algorithm

Algorithm:

for $n = 1, ..., N$, **do**
  $\boldsymbol{d} = [\,]$; vector containing indices of dark nodes.
  for $\eta = 2^K, 2^K + 1, ..., 2^{K+1} - 1$: i.e. 2^K leaf nodes **do**
    $D_M(\boldsymbol{x}[n], S_\eta[n-1]) = \boldsymbol{\beta}_\eta^T[n-1]\boldsymbol{\Lambda}_\eta^{-1}[n-1]\boldsymbol{\beta}_\eta[n-1] + \delta_\eta^{-1}[n-1]\|\boldsymbol{\gamma}_\eta[n-1]\|^2$
  end for
  $\boldsymbol{d}(K+1) = \arg\min_\eta D_M(\boldsymbol{x}[n], S_\eta[n-1])$ for $\eta = 2^K, 2^K + 1, ..., 2^{K+1} - 1$
  The remaining K dark nodes are determined by climbing up the tree till the root node:
  $\boldsymbol{d}(K)$: parent node of $\boldsymbol{d}(K+1)$
  $\boldsymbol{d}(K-1)$: parent node of $\boldsymbol{d}(K)$, till
  $\boldsymbol{d}(1)$: root node
  Find weights for each dark node:
  $\sigma_1[n-1] = \frac{1}{2}$
  for $\eta = \boldsymbol{d}(2), ..., \boldsymbol{d}(K+1)$, OR $k = 2, ..., K+1$, **do**
    if $k = K+1$, **then**
      $\sigma_k[n-1] = C_s[n-1]\sigma_{k-1}[n-1]$: s is the sibling node of $\boldsymbol{d}(k)$
    else if $k < K+1$, **then**
      $\sigma_k[n-1] = \frac{1}{2}C_s[n-1]\sigma_{k-1}[n-1]$: s
    end if
    $\mu_k[n-1] = \frac{\sigma_k[n-1]E_k[n-1]}{C_1[n-1]}$
  end for
  Estimation of $y[n]$ from $\boldsymbol{x}[n]$:
  $\tilde{y}_k[n-1] = \boldsymbol{w}_k^T[n-1]\tilde{\boldsymbol{x}}_k[n]$: Prediction of each dark node k
  $\tilde{y}[n] = \sum_{k=1}^{K+1} \mu_k[n-1]\tilde{y}_k[n-1]$: Weighted combination of the individual nodes prediction based on their past performance.
  Update the parameters using Algorithm 3.
end for

The construction of the proposed algorithm is based on the following theorem whereas the complexity of the algorithm is linear in the depth of the hierarchical tree structure.

Theorem 1. *Let* $\{\boldsymbol{x}[n]\}_{n\geq 1}$ *is the observed* $D-$ *dimensional sequence and* $\{y[n]\}_{n\geq 1} \in \mathbb{R}$ *is the desired sequence, where* $|y[n]| \leq A_y$*. Then the Algorithm 2, whose complexity is linear in the depth of the tree, yields*

$$\sum_{n=1}^{N}(y[n] - \hat{y}[n])^2 \leq \min_{\mathcal{P}_i}\left(\sum_{n=1}^{N}(y[n] - \hat{y}_{\mathcal{P}_i}[n])^2 + 8A_y^2\mathbb{C}(\mathcal{P}_i)\ln(2)\right) + O(1), \tag{2.13}$$

for any N, where $\hat{y}[n]$ is the estimate of $y[n]$ as given by (2.5), $\mathcal{P}_i$ is the i^{th} partition from the doubly exponential class of Fig. 2.3 with the cost of partition $\mathbb{C}(\mathcal{P}_i)$, and $\hat{y}_{\mathcal{P}_i}[n]$ is the estimate of $y[n]$ by the partition $\mathcal{P}_i$. The cost of partition $\mathcal{P}_i$ is given by, $\mathbb{C}(\mathcal{P}_i) = J_i + \eta_i - 1$, where J_i is the number of regions in the partition $\mathcal{P}_i$ and η_i is the number of branches of the tree that are not fully grown [42, 99].

This theorem is a basic application of Theorem 1 of [42]. The weights $\omega_k[n-1]$ in (2.5) assigned to each dark node regressors are determined by the performance of these nodes until the current time. Therefore, we sequentially measure the performance of each node that is used for estimation and update them after each iteration. In the next sub-section we explain how to measure the performance of each node in estimating the desired sequence $y[n]$. We then use this performance measure to assign combination weights to each node.

2.2.1 Node Performance Measure

In our method, instead of using fixed partitions for the piecewise linear regression and manifold learning, we use a tree structure that dynamically partitions the regressor space on each level of the tree. We then use the context tree weighting method to linearly combine the estimates of each node. The weights assigned to each node in (2.5) are determined by the node performance in the previous iterations. We assign C_η to each node as a measure of performance, which is an exponential function of the regret $\sum_{i=1}^{n_\eta - 1} (y[i] - \hat{y}_\eta[i])^2$. These C_η are used to calculate the weight or portion of each node regressor in the mixture of (2.5). The universal performance measure, C_u is a weighted combination of all C_η below the root node and $C_r = C_u$, where C_r represents the performance of the root node [42]. We represent the desired data $y[t]$ for $t = 1, ..., n$ by y^n, i.e., $y^n = \{y[1], y[2], ..., y[n]\}$. For a specific partition, $\mathcal{P}_i$, among the doubly exponential class of possible partitions, the performance is measured by [42]:

$$C(y^n|\hat{y}^n, \mathcal{P}_i) \triangleq \exp\left\{-\frac{1}{2a}\sum_{t=1}^{n}(y[t] - \hat{y}[t])^2\right\}, \tag{2.14}$$

which is the performance of the partition $\mathcal{P}_i$ in estimating the desired sequence y^n. Here, a is a constant that depends on $A_y = \max\{|y[n]|\}$ and $a \triangleq 4A_y^2$ [42]. For a given $\mathcal{P}_i$, the best predictor is the one with the minimum loss function, $\sum_{t=1}^{n}(y[t] - \hat{y}[t])^2$, i.e.,

$$C^*(y^n|\mathcal{P}_i) \triangleq \exp\left(-\frac{1}{2a}\min_{\hat{y}^n}\sum_{t=1}^{n}(y[t] - \hat{y}[t])^2\right). \tag{2.15}$$

The best partition can be chosen among all $\mathcal{P}_i$ by maximizing $C^*(y^n|\mathcal{P}_i)$ over all $\mathcal{P}_i$, i.e., $C^*(y^n|\mathcal{P}_i^*) \triangleq \max_{\mathcal{P}_i} C^*(y^n|\mathcal{P}_i)$.

In the context tree algorithm, the performance measure of a leaf node is defined as [42]:

$$\tilde{C}_\eta(y^n) \triangleq \exp\left(-\frac{1}{2a}\sum_{t=1}^{n_\eta}(y[t]-\hat{y}_\eta[t])^2\right), \tag{2.16}$$

where n_η are the number of past input samples closest to the leaf node η. Here, $\hat{y}_\eta[t]$ is the estimate of $y[t]$ from the node η and is given by (2.12). The performance measure of an inner node is defined as [42],

$$\begin{aligned}\tilde{C}_\eta(y^n) \triangleq& \frac{1}{2}\tilde{C}_{\eta_u}(y^n)\tilde{C}_{\eta_l}(y^n) \\ &+ \frac{1}{2}\exp\left(-\frac{1}{2a}\sum_{t=1}^{n_\eta}(y[t]-\hat{y}_\eta[t])^2\right),\end{aligned} \tag{2.17}$$

which is a weighted combination of the performance measures assigned to the node η and its two children nodes, η_u and η_l. This way we define the universal performance measure as a weighted combination of all the leaf and inner nodes. The universal performance measure [42] for $y[n]$, given past observations till $n-1$, is given by:

$$\tilde{C}_u(y[n]|y^{n-1}) = \sum_k \mu_k[n-1]\exp\left(-\frac{1}{2a}l(y^{n-1},\tilde{y}_k^{n-1})\right), \tag{2.18}$$

where $l(y^{n-1},\tilde{y}_k^{n-1}) = (y[n-1]-\tilde{y}_k[n-1])^2$. The weights $\mu_k[n-1]$ are defined as:

$$\mu_k[n-1] \triangleq \frac{\sigma_k[n-1]\exp\left(-\frac{1}{2a}\sum_{t=1}^{n_k-1}(y[t]-\tilde{y}_k[t])^2\right)}{\tilde{C}_u(y^{n-1})} \tag{2.19}$$

where $\sigma_1[n-1] = \frac{1}{2}$ and $\sigma_k[n-1] = \frac{1}{2}\tilde{C}_s[n-1]\sigma_{k-1}[n-1]$ for $k>1$. Here, $\tilde{C}_s$ denotes the performance measure of the sibling node of k. The estimate of $y[n]$ is given by the weighted combination of the regressor outputs from each dark node,

$$\tilde{y}[n] = \sum_k \mu_k[n-1]\hat{y}_k[n] \tag{2.20}$$

which is the same as (2.5) with $\omega_k[n] = \mu_k[n]$ and $\mu_k[n]$ are defined in (2.19).

Alternatively, we can use the approximate Mahalanobis distance (2.9) to define the node performance measure. For the leaf nodes,

$$\tilde{C}_\eta(y^n) \triangleq \exp\left(-\sqrt{D_M(\boldsymbol{x}[n],\Re_\eta[n])}\right), \tag{2.21}$$

and for the inner nodes,

$$\begin{aligned}\tilde{C}_\eta(y^n) =& \frac{1}{2}\tilde{C}_{\eta_u}(y^n)\tilde{C}_{\eta_l}(y^n) \\ &+ \frac{1}{2}\exp\left(-\sqrt{D_M(\boldsymbol{x}[n], \Re_\eta[n])}\right).\end{aligned} \tag{2.22}$$

In the next section, we describe the update mechanism for all the parameters of the submanifolds and the nodes.

2.2.2 Update Node Parameters

There are two layers of parameters which we update before the next data sample $\boldsymbol{x}[n+1]$ arrives, *i*) the submanifold shape parameters $\{\boldsymbol{Q}_\eta, \boldsymbol{c}_\eta, \boldsymbol{\Lambda}_\eta\}$ and *ii*) the context tree and regressor parameters, i.e., C_η and $\boldsymbol{w}_\eta$. Since $\boldsymbol{x}[n]$ only affects the performance measure and regressors associated with the dark nodes of the context tree, we use a greedy approach instead of updating all $2^{K+1}-1$ nodes, and update the tree only for the dark nodes in $K+1$ operations [27, 28, 42]. For the subsets shape parameters associated with each node, i.e. $\boldsymbol{Q}_\eta, \boldsymbol{c}_\eta, \boldsymbol{\Lambda}_\eta$, we update $\boldsymbol{Q}_\eta$ while keeping $\boldsymbol{\Lambda}_\eta$ and mean vectors $\boldsymbol{c}_\eta$ fixed. Then we update $\boldsymbol{\Lambda}_\eta$ and $\boldsymbol{c}_\eta$ by considering $\boldsymbol{Q}_\eta$ fixed and using the data sample $\boldsymbol{x}[n]$ and the projections $\boldsymbol{\beta}_\eta$. These updates are detailed in the Algorithm 3. For instance, we update $\boldsymbol{c}_\eta[n]$ as follows:

$$\boldsymbol{c}_\eta[n] = \alpha\boldsymbol{c}_\eta[n-1] + (1-\alpha)\boldsymbol{x}[n], \tag{2.23}$$

where $0 < \alpha \leq 1$ is a small positive constant that determines the dependence of $\boldsymbol{c}_\eta$ on its past values. In general, α should be close to 1 as in the Recursive Least Squares (RLS) algorithm [4, 5]. We choose small α for the fast evolving manifold and large α for the slow evolving manifold [28].

We update the subspace basis $\boldsymbol{Q}_\eta$ for each dark node using Parallel Estimation and Tracking by REcursive Least Squares with fast orthogonalization (PETRELS-FO) [28, 103]. The details of using the subspace tracking algorithm can be found in [27, 28, 38, 103]. After updating the node parameters for the dark nodes, Algorithm 2 is repeated for the next data sample, $\boldsymbol{x}[n+1]$ to estimate $y[n+1]$. Our algorithm achieves the performance of the best partition among the doubly exponential class with a complexity linear in the depth of the hierarchical tree structure, as given by the Theorem 1.

Remark. *Note that the actual regression value $y[n]$ is needed to update the performance measure C of each node. However, if we use the tree for classification purposes, then we can choose the closest value to $\hat{y}[n]$ among all possible values as*

Algorithm 3: Update Parameters

for $k = K+1, ..., 1$, (Dark nodes of the previous iteration), **do**
  $\boldsymbol{w}_k[n] = \boldsymbol{w}_k[n-1] + \nu \tilde{\boldsymbol{x}}_k[n](y[n] - \boldsymbol{w}_k^T[n-1]\tilde{\boldsymbol{x}}_k[n])$:
  Updated regressor weight vectors for k^{th} node. ν is the step size,
  a small positive constant.
  $E_k[n] = E_k[n-1] \exp\left(-\frac{1}{2a}(y[n] - \tilde{y}_k[n-1])^2\right)$
  Update node performance measure
  if $k = K+1$, **then**
    $C_k[n] = E_k[n]$
  else if $k \neq K+1$, **then**
    $C_k[n] = \frac{1}{2}C_{k_u}[n-1]C_{k_l}[n-1] + \frac{1}{2}E_k[n]$, k is inner node, and k_u and k_l are the two children nodes of k.
  end if
  Update Subspace and manifold parameters
  $\boldsymbol{c}_k[n] = \alpha \boldsymbol{c}_k[n-1] + (1-\alpha)\boldsymbol{x}[n]$
  $\lambda_k^{(m)}[n] = \alpha \lambda_k^{(m)}[n-1] + (1-\alpha)\left(\boldsymbol{\beta}_k[n]\right)_m^2$ where
  $m = 1, ..., d$ and $\left(\boldsymbol{\beta}_k[n]\right)_m$ is the m^{th} component of vector $\boldsymbol{\beta}_k[n]$
  $\delta_k[n] = \alpha \delta_k[n-1] + (1-\alpha)\frac{\|\boldsymbol{\gamma}_k[n-1]\|^2}{(D-d)}$
  $\boldsymbol{Q}_k$ are updated using PETRELS-FO.
end for

the true value of $y[n]$ in the decision directed mode [101]. Nevertheless, even for the regression task, once the training phase is complete, our algorithm still adapts to the structure of the data $\boldsymbol{x}[t]$, i.e., the node shape parameters $\boldsymbol{Q}_\eta, \boldsymbol{\Lambda}_\eta, \boldsymbol{c}_\eta$ are updated based on the new data sample $\boldsymbol{x}[t]$. To update the node performance measure C when $y[n]$ is not available, we use (2.21) *and* (2.22).

Remark. *Note that if we use the Euclidean distance to decide on the membership of the data to the regions, we may end up having two very similar nodes, since we update only the dark nodes at each iteration. However, since we use the Mahalanobis distance, we do not encounter this issue. This is because the Mahalanobis distance takes into account the correlation among data points. Hence, if the means of the data assigned to two different nodes are very similar (i.e., the Euclidean distance between the means is small), the Mahalanobis distance between them is still large enough. Hence, the nodes may not be very similar.*

2.2.3 Initialization and Choice of Parameter Values

The algorithm may be initialized by using a small training set to fix the initial values for $\boldsymbol{Q}_\eta, \boldsymbol{c}_\eta, \boldsymbol{\Lambda}_\eta$ and δ_η. However, for large data lengths, the effect of initialization is negligible and unnecessary, and the algorithm can be randomly initialized. For the first example used in this chapter, we use a small training set

of length $N = 1000$, and use the k-means algorithm to bi-partition the data till the data is divided into 2^K regions. For the rest of the examples, the subspace and the regressor parameters are initialized either randomly or with zeros. We choose the parameter α for updating the subspace parameters that ranges from 0.8 to 0.95 based on the speed of variation in the submanifold. To update the regressor weight vectors, we use RLS [4, 5] with a forgetting factor between 0.5 and 0.75 in different examples.

Remark. *The choices of d and K are very crucial as they are directly related to the performance and complexity of the algorithm. In practice, one can use a few samples of the regressor vectors to compute the covariance matrix and obtain d. To this end, one can compute the Singular Value Decomposition (SVD) of the covariance matrix and put a threshold based on the values of the eigenvalues. Then, for the k^{th} node, the number of eigenvalues greater than the threshold can be used as the dimensionality d_k. Moreover, in order to use the same dimensionality for all nodes, one can choose the largest d_k as the intrinsic dimensionality d. However, we do not determine d using the SVD, since it is computationally prohibitive in big data applications. Instead, we choose a fixed value for d (the same dimensionality for all nodes) and, as shown in the experiments, our algorithm is robust to mismatch in the true and our selected intrinsic dimension. In real life data, the intrinsic dimension as well as the curvature of the manifold is usually not known and can even be time varying. However, a sufficiently large K can also accommodate for the mismatch between the intrinsic dimension of the manifold and the chosen d. An adaptive hierarchical tree of depth K inherently incorporates all the possible trees of depths less than K, therefore in a time varying environment, our algorithm adapts to the fluctuations in data without changing K as shown by the real data tests in Section 2.3. By this, we achieve a significant regression performance over a wide range of time varying data without increasing the complexity of the algorithm.*

2.3 Experiments

In this section, we illustrate the performance of the adaptive hierarchical tree algorithm with several real and synthetic examples. The first set of experiments involves a high dimensional sequence that lies on a time varying submanifold [28]. The dimension of the submanifold is $D = 10$ and the intrinsic dimension is $d_{\text{intrinsic}} = 1$. Let θ be a uniformly distributed random variable, i.e., $\theta \sim \mathcal{U}[-2, 2]$. Let $\boldsymbol{u}[n]$ is a sequence of $D-$dimensional vectors, i.e., $\{\boldsymbol{u}[n]\}_{n\geq 1}$ with j^{th} element of the vector $\boldsymbol{u}[n]$ given by,

$$u_j[n] = \frac{1}{\sqrt{2\pi}} e^{-(\epsilon_j - \theta)^2/(2\kappa^2[n])}, \tag{2.24}$$

where $\epsilon_j = -2 + 4j/D, j = 1, ..., D$, are points regularly spaced between -2 and 2 and $\boldsymbol{u}[n] = [u_1[n], u_2[n], ..., u_D[n]]^T$. The parameter $\{\kappa[n]\}_{n\geq 1}, \kappa[n] \in \mathbb{R}$ manages the fluctuation in the submanifold such as the curvature of the submanifold increases with the value of κ. The parameter $\kappa[n]$ is formally defined by:

$$\kappa[n] = 100 - \kappa_0|(n \bmod 2r) - r|, \text{ for } n = 1, 2, ..., \tag{2.25}$$

where $r = 100/\epsilon_0$ and κ_0 gives the acceleration of the submanifold. Here, $\kappa[n]$ forms a triangular waveform with its minima and maxima at 0 and 100 respectively. The period of the triangular wave is $2r$ and r is the number of points to reach from 0 to 100. We generate the vector sequence $\boldsymbol{x}[n]$ by

$$\boldsymbol{x}[n] = \boldsymbol{u}[n] + \boldsymbol{\rho}[n],$$

where $\boldsymbol{\rho}[n] \in \mathbb{R}^D$ is white Gaussian noise with autocovariance $\boldsymbol{\Sigma}_{\boldsymbol{\rho}} = 4 \times 10^{-4} \boldsymbol{I}_D$. The resultant vector sequence $\boldsymbol{x}[n]$ lies on a time varying submanifold with *ambient dimension*, $D = 10$ while $d_{\text{intrinsic}} = 1$. We generate the desired scalar sequence $\{y[n]\}_{n\geq 1}, y[n] \in \mathbb{R}$ by the following piecewise model,

$$y[n] = \begin{cases} \boldsymbol{w}_1^T \boldsymbol{x}[n] + \varrho_1[n], & \text{if } \kappa[n] \leq 25 \\ \boldsymbol{w}_2^T \boldsymbol{x}[n] + \varrho_2[n], & \text{if } 25 < \kappa[n] \leq 50 \\ \boldsymbol{w}_3^T \boldsymbol{x}[n] + \varrho_3[n], & \text{if } 50 < \kappa[n] \leq 75 \\ \boldsymbol{w}_4^T \boldsymbol{x}[n] + \varrho_4[n], & \text{if } 75 < \kappa[n] \leq 100 \end{cases}, \tag{2.26}$$

where $\boldsymbol{w}_j \in \mathbb{R}^D, j = 1, .., 4$, and ϱ_j for $j = 1, ..., 4$, is zero mean white Gaussian noise with variance $\sigma^2 = 1 \times 10^{-4}$. Note that for this example, there is a piecewise linear structure that can be perfectly modeled by the leaves of the tree. Moreover, the algorithm can perfectly match the underlying time varying submanifold using the adaptive tree structure. The curvature of the submanifold increases and decreases with $\kappa[n]$ in a cyclic manner and yet the algorithm is able to track the variations in the submanifold.

We use $d = 1$ as the dimension of projection and use PETRELS algorithm for subspace tracking [103] with fast orthogonalization, and hence we denote it by PETRELS-FO [28, 38]. We apply piecewise linear regression using the context tree weighting method on $\mathbb{R}^d$ by projecting the observed vectors $\boldsymbol{x}[n]$ on the estimated subspace for each region of the tree with $K = 3$ as described in Algorithm 2 and Algorithm 3. We display the results in Fig. 2.5. Here, we use $\kappa_o = 0.04$ and plot the normalized accumulated mean square errors against 2000 samples of the data. We also calculate the mean square errors using the regression on the finest partitions. The adaptive hierarchical trees algorithm is particularly useful for short data records. As expected, the performance of the

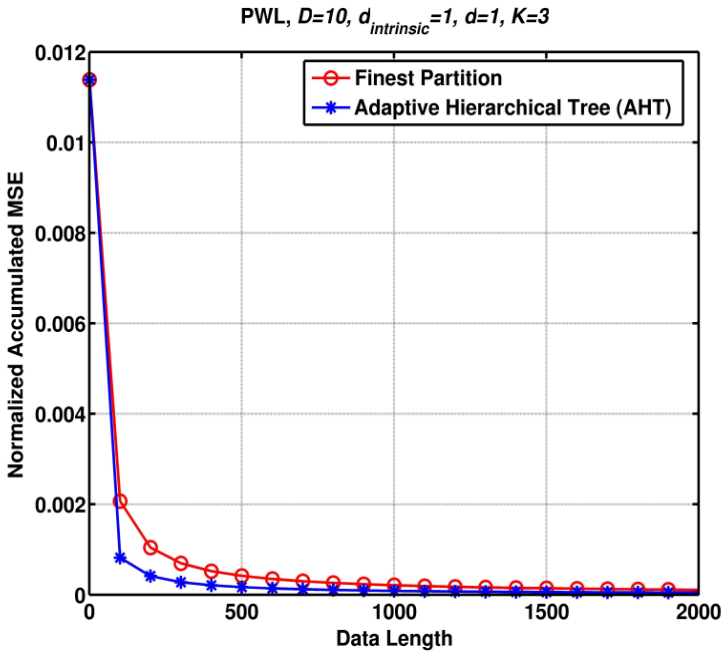


Figure 2.5: Sequential piecewise linear prediction of the desired data $\hat{y}[n]$ of (2.26) with $d = 1$ and tree depth, $K = 3$, sequential piecewise linear predictor with finest partitions on the tree.

finest partition suffers when the data length is small, due to overfitting. Since, our algorithm adaptively combines predictors for each different partition based on their performance, it is able to favor the coarser models with a small number of parameters during the initial phase of the algorithm. This avoids the overfitting problems faced by the sequential algorithms using the finest partition. As the data length increases, both algorithms almost converge to the same minimum error rate. Hence, the tree based adaptation is attractive for adaptive processing in a time varying environments for which a windowed version of the most recent data is typically used.

The performance of the algorithm improves with increasing the depth of the tree, however, it saturates at a specific K, i.e., when the piecewise model of (2.26) can be perfectly modeled by the proposed algorithm with a certain number of regions. In our current example, we observe that choosing $K = 2$ must be sufficient since it divides the regressor space into a maximum four regions. However, due to the time varying nature of the submanifold, choosing $K > 2$ improves the performance. The error rate reaches the saturation point at $K = 3$ and further increasing the depth of the tree results in overfitting. This behavior can be seen in Fig. 2.6.

We next compare the performance of our algorithm for various choices of the dimension of projection using the same dataset. We plot the normalized MSE for

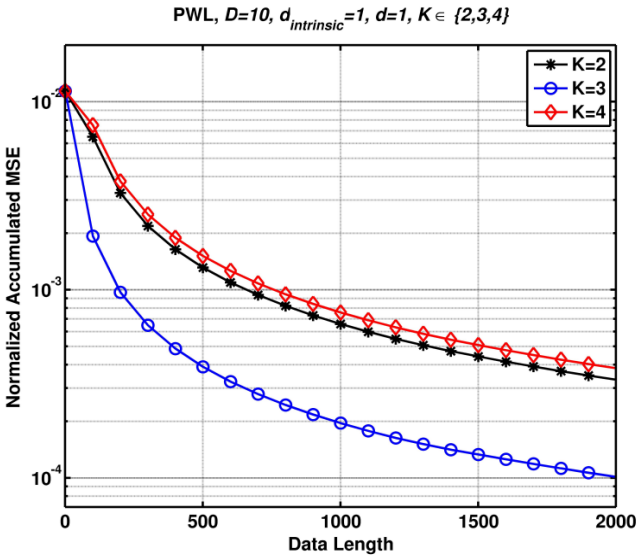


Figure 2.6: Piecewise Linear Prediction on high dimensional, time varying submanifolds using Adaptive Hierarchical Trees. $D = 10, d_{\text{intrinsic}} = 1, d = 1$, and $K = 2, 3, 4$
$d = 1, 2, 3$ in Fig. 2.7.

It is interesting to note that for the dimension of projection higher than $d_{\text{intrinsic}}$, i.e., $d > d_{\text{intrinsic}}$, the performance of the algorithm is slightly improved for the same K. We observe that the tracking of the submanifold gets better with the increase in dimension of projections. From the above two results (Fig. 2.6 and Fig. 2.7), it can be inferred that the algorithm performs better by either increasing d or K, yet at the cost of complexity and memory requirement. However, there are optimal values for both d or K and an increase beyond that may result in overfitting, that in turn degrades the performance. Therefore, as seen from results in the current example, the performance worsens for $d = 3$ as compared to $d = 2$ and $d = 1$. We discover these effects of parameters in further details in the subsequent experiments.

In the next example, we use a $D = 100$ dimensional data that lies on a time varying submanifold with the intrinsic dimension $d_{\text{intrinsic}} = 1$. We generate the datasets in the same manner as eq:subman,eq:pwl. We choose the tree depth, $K = 4$ and plot the normalized mean square errors for various choices of d while $d \geq d_{\text{intrinsic}}$. Fig. 2.8 shows a significant decrease in MSE for $d = 2$ over $d = 1$. Furthermore, the algorithm attains the minimum error rate faster for larger d. This makes the choice of larger d attractive for adaptive processing in time varying environment for which a shorter length of the most recent data is used. However, even with $d = 1$, the algorithm attains almost the same minimum error rate but for a larger data length. Still choosing d as large as possible is not recommended and rather makes the algorithm inefficient since larger d results in overfitting and may even increase the error rate. This is evident from Fig. 2.8,

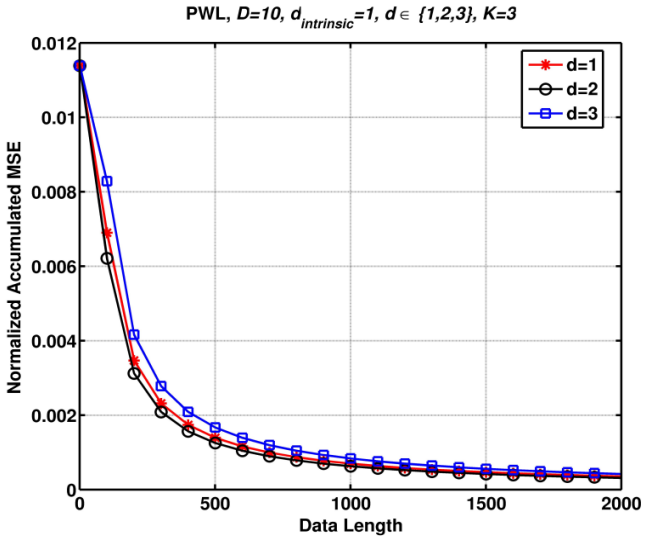


Figure 2.7: Sequential piecewise linear prediction of the desired data $\hat{y}[n]$ of (2.26) with $d = 1, 2, 3$ and tree depth, $K = 3$. The intrinsic dimension is $d_{\text{intrinsic}} = 1$

where choosing $d > 3$ increases the process complexity and memory requirement yet the performance is degraded.

We next compare the performance of Adaptive Hierarchical Trees for a new dataset with dimensionality $D = 200$ and intrinsic dimension $d_{\text{intrinsic}} = 3$ [28,38]. Let the two-dimensional parameters $[f_0, \phi]$ be uniformly and randomly generated with frequency $f_0 \sim \mathcal{U}[1, 100]$ and phase $\phi \sim \mathcal{U}[0, 1]$. Let the j^{th} element of vector $\boldsymbol{u}[n] \in \mathbb{R}^D$ is given by,

$$u_j[n] = \sin\left[2\pi(f_0\tilde{\epsilon}_p + \frac{\tilde{\kappa}[n]^2}{2}\tilde{\epsilon}_j^2 + \phi)\right], \tag{2.27}$$

where $\tilde{\epsilon}_j = 10^{-4}j$, $j \in \{1, 2, ..., 100\}$ are points regularly spaced between 0 and 0.01. The parameter $\tilde{\kappa}[n]$ for $n \in \{1, ..., N\}, N =$ length of the data, controls the variations in the submanifold and is set according to the following equation

$$\tilde{\kappa}[n] = 100|\cos(\theta[n])|, \text{ for } n = 1, 2, ..., \tag{2.28}$$

where $\theta[n] \in [0, 3\pi]$. Then, $\boldsymbol{u}[n] = [u_1[n], u_2[n], ..., u_D[n]]^T$. The observed $D-$dimensional data $\boldsymbol{x}[n]$ is given by

$$\boldsymbol{x}[n] = \boldsymbol{u}[n] + \boldsymbol{\rho}[n],$$

where $\boldsymbol{\rho}[n] \in \mathbb{R}^D$ is a white Gaussian noise with autocovariance $\boldsymbol{\Sigma}_{\boldsymbol{\rho}} = 4\times10^{-4}\boldsymbol{I}_D$. The desired data is generated by (2.26), with four piecewise linear regions.

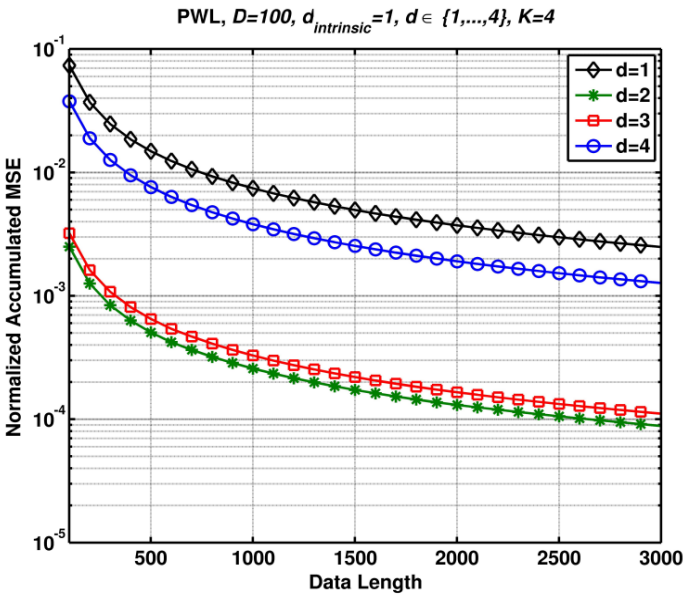


Figure 2.8: Piecewise linear regression on 100− dimensional time varying submanifold with intrinsic dimension $d_{intrinsic} = 1$ using adaptive Hierarchical trees. Normalized MSE are plotted for 3000 samples of online data using $K = 4$ and $d = 1, ..., 4$

In Fig. 2.9, we plot the normalized MSE for the adaptive hierarchical tree algorithm using trees of various depths $K \in \{4, 6, 8\}$ and different values of the dimensionality, $d \in \{1, 2, 3\}$. The results here show that a value of $d = 3$ is optimal in this example. It is interesting to note that our algorithm not only attains the minimum error rate faster but also keeps it stable when the data length increases. Since, our algorithm "always" dynamically updates and combines the weights of the partitions, it is capable of catching up with the sudden changes in the model. This makes the algorithm effective for applications with dynamic environment.

The choice of d affects the performance of the algorithm that can be seen in Fig. 2.9 (b), where $d > 1$ shows great improvement on the performance. In addition, in order to see the effect of the tree depth K on the performance, we have performed the simulations for $K \in \{4, 6, 8\}$, and for $d \in \{2, 3\}$. The results show that we can improve the performance by increasing the depth of the tree for both $d = 2$ and $d = 3$. However, since the depth K directly affects the computational complexity, it should be determined based on the complexity considerations. Moreover, as we observe from the simulations, when we choose $d = 3$, the algorithm can effectively match the underlying data model and reach a significantly low Normalized Accumulated MSE. Hence, if the intrinsic dimension of the submanifold is known, it gives a good initial guess to use for the value of d for d should be at least equal to the intrinsic dimension. It is interesting to note that for an optimal choice of K, a $d < d_{\text{intrinsic}}$ may even be sufficient for the algorithm to track and predict the desired data sequence from the high dimensional time

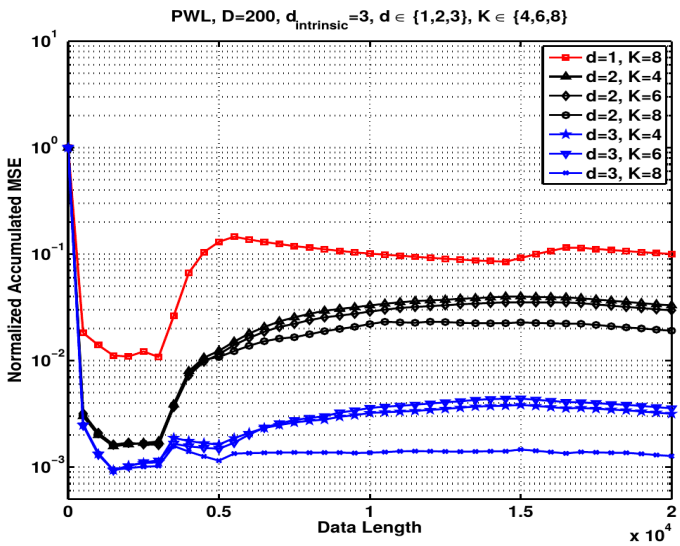


Figure 2.9: Normalized MSE based performance analysis using AHT with $K \in \{4, 6, 8\}$ and $d = \{1, 2, 3\}$, to see the effect of these parameters on the time-varying submanifold tracking performance.

varying submanifold. An optimal K not only determines the modeling strength with respect to the nonlinearity of the model but also the approximation to the true manifold. This is specifically helpful for the real data where the intrinsic dimensionality may be unknown and even varying, then choosing K optimally and adaptively can be used to track the subspace for a short training data. In such case, the best practice would be to start from a simpler model, with small d and K, and adjust these parameters till a certain minimum error rate is achieved or the saturation point occurs. However, this is only required at the training stage. Furthermore, if real online data yet has more fluctuations than initially guessed, there is no need to change K since the weighting coefficients μ_k on each level of the tree already take care of the time varying curvature. For instance, if the curvature of the manifold increases, the algorithm increases the weights of finer nodes. This is in contrast to [28, 38] where the manifold is tracked on a single level at a certain time and if the curvature changes, the tree has to grow or prune. We show by examples that in most real world scenarios, simpler models in terms of d and K works well enough to achieve better results than previous state of the art techniques.

We next illustrate the performance of our algorithm on a number of real world data sets. The public data sets are obtained by measurements on specific parameters related to the underlying system. Then, these measurements are collected into vectors to be used as the regressor vectors in the regression task. However, these measurements are not necessarily independent of each other and there are correlations between the measurements of different parameters. Therefore, it is safe to assume that the observed data lies on a time varying submanifold

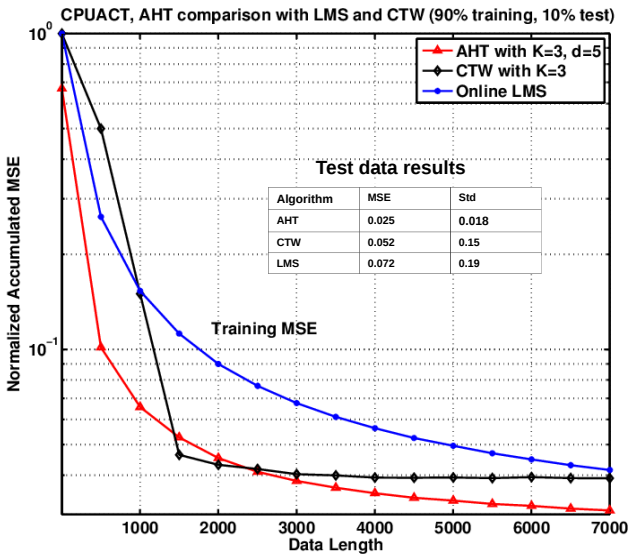


Figure 2.10: Performance Analysis of AHT on real data (Computer Activity) with $d = 5, D = 21$ and $K = 3$. 90% of the data is used for training and the remaining data is used as the test data.

of lower intrinsic dimension. The following simulations demonstrate the superior performance of our algorithm on such data sets, which is consistent with our assumptions.

In the following experiments, we use 90% of the data for training and the remaining 10% data for testing. In the first example, we use the computer activity dataset [104], where we predict the portion of time the CPU runs in user mode from the observed 21 attributes. The results are given in Fig. 2.10. Here we achieve a significantly small MSE and faster convergence as compared to Context Tree Weighting (CTW) [42] and online Least Mean Square (LMS) [4] by using $d = 5$ whereas the actual dimension is $D = 21$. The results demonstrate the superior performance of our algorithm not only in the training phase, but also the in test phase.

In another experiment, we use online news popularity dataset [105]. The target is to determine the popularity of a certain news item, identified by a URL, by the number of shares it received. The input contains 59 numerical attributes containing the statistic about the content of the news articles, e.g., number of words in title and content, time, frequency of words, categories of words as positive, negative or neutral, and the day the article is published. We use the AHT algorithms in regression setting to predict the popularity of news item and compare the performance with online LMS, using normalized MSE. The results are shown in Fig. 2.11. The proposed algorithm significantly outperform the LMS algorithm despite using $d = 1$ and $K = 1$ in AHT. Here, we significantly reduce

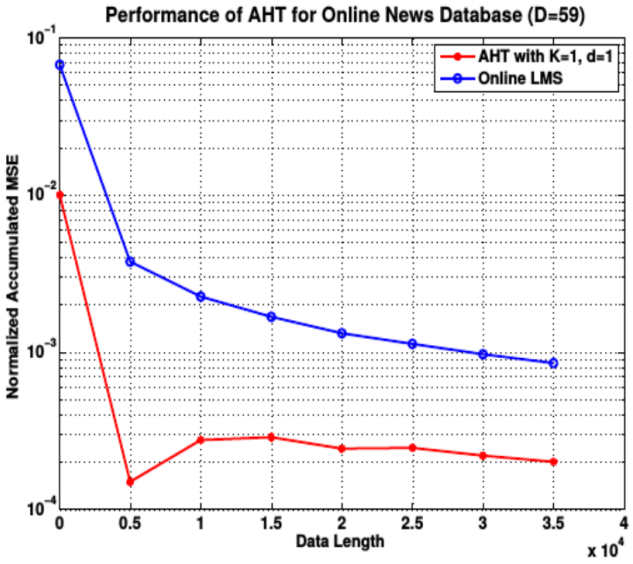


Figure 2.11: Performance Analysis of AHT on real data (Online News Popularity) with $d = 1, D = 59$ and $K = 1$.

the dimensionality of the input space, i.e. from $D = 59$ to $d = 1$, while gaining improvement in the performance.

In the next example, we evaluate the performance of our algorithm on the real world dataset KDD-CUP99 [106]. The task is to design a software to detect network intrusions in order to protect a computer network from unauthorized users, including (perhaps) the insiders. We build a predictive model (i.e., a classifier) capable of distinguishing between "bad" connections, called intrusions or attacks, and "good" connections. A connection is a sequence of TCP packets starting and ending at some well defined times, between which data flows to and from a source IP address to a target IP address under some well defined protocol. The objective is to classify the connections as "normal" or an "attack", based on the corresponding features of that connection. We use the corrected dataset of KDD-CUP99 [106], which consists of 42 features, a mix of categorical and numeric features. We convert the categorical features to numeric by introducing dummy variables and the final dataset has $D = 114$ dimensional data points (feature vectors).

We compare the performance of our algorithm with that of the conventional context tree weighting (CTW) method. For the CTW method, we use a tree with depth $K = 3$, whereas we have done the experiment with $K = 2$ and $K = 3$, and $d = 20$ for our algorithm. As depicted in Fig. 2.12, our algorithm significantly outperforms the CTW method as our algorithm achieves a lower Normalized Accumulated MSE during the training phase. The results also reveal the superior performance of our algorithm during the test phase (see the table included in the Fig. 2.12). This is because our algorithm, unlike the CTW, continues to adapt

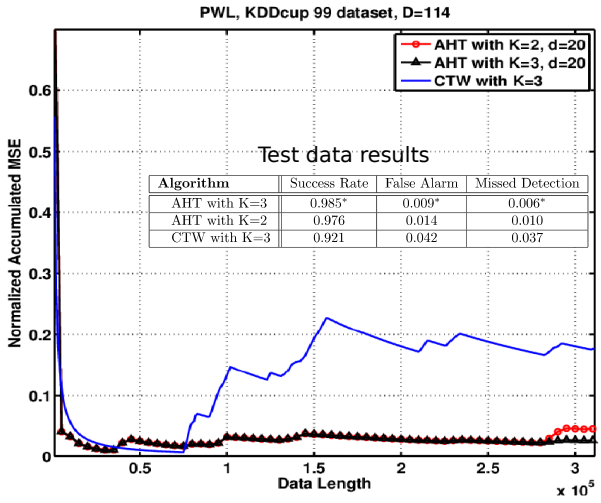


Figure 2.12: Performance Analysis of AHT on real data (KDD CUP99 Database) with $d = 20, D = 114$ and $K \in \{2, 3\}$. 90% of the data is used for training and the remaining data is used for the test.

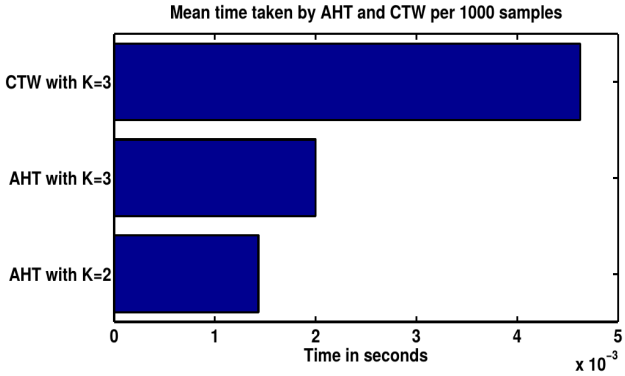


Figure 2.13: The comparison between the time consumption of different algorithms in the KDD CUP99 experiment.

itself to the structure of the data in the test phase. In addition, the results show that when we use $K = 3$, our algorithm can perform slightly better than when we use $K = 2$. Furthermore, Fig. 2.13 shows that our algorithm is remarkably faster than the CTW method.

Finally, we summarize the performance of AHT on other widely used real world datasets and compare with the previous state of the art algorithms in terms of MSE and computational complexity [100]. The results are shown in Table 2.1. We compare the performance and complexity of our algorithm with Decision Adaptive Trees (DAT) [100] and Context Tree Weighting (CTW) [42]. The Kinematics data ($D = 8$) [104] is the simulation of the dynamics of an 8−link all-revolute robotic arm where we predict the distance of the end-effector from the target. In the Elevators dataset with $D = 18$ [107, 108], the desired

Algorithms / Data Sets	DAT $\{O(4^K D)\}$	CTW $\{O(KD)\}$	**AHT** $\{O(Kd)\}$
Kinematics	0.0639	0.0728	**0.0600***
Pumadyn	0.0781	0.0923	**0.0505***
Elevators	0.0091	0.0210	**0.0130***
Bank	0.0511	0.1100	**0.0320***

Table 2.1: Performance of the proposed algorithm in terms of Normalized time accumulated MSE compared with the results of state-of-the-art algorithms. The complexity of each algorithm in terms of order of operations on real numbers is also provided.

task is to take action on the elevators of an F16 aircraft. The Bank dataset ($D = 32$) [108] is a realistic simulation of the queues in a series of banks and the task is to predict the portion of customers who would leave the bank because of full queues. The Pumadyn dataset ($D = 32$) [107] is a realistic simulation of the dynamics of Unimation Puma 560 robot arm where the target task is to predict the angular acceleration of the robot arm's links. Here we use $K = 2$ and $d = 1$ in all the examples and achieve significantly good performance with relatively reduced complexity, i.e., $O(Kd)$ as compared to $O(4^K D)$ in case of DAT and $O(KD)$ in case of CTW, where $d \ll D$.

2.4 Discussion

We consider the problem of piecewise linear regression on high dimensional data from a competitive algorithm perspective. The data is lying on a time varying submanifold and we use a hierarchical tree structure to track the submanifold and escape the curse of dimensionality. We introduce a novel nonlinear regression algorithm using adaptive hierarchical trees that dynamically tracks the subspace of underlying manifolds. The total squared prediction error of the proposed algorithm is bounded by $O(dJ\ln(n/J)) + O(\mathbb{C}(\mathcal{P}_i))$, where J is the number of regions in the best piecewise model. We introduce a method using the context tree notion for the piecewise modeling that competes well against a doubly exponential class of possible partitioning of the regressor space. The algorithm is shown to instantly adapt to the variations in the data and hence, is effective for high dimensional data with short data lengths in an online setting. The resulting algorithms are suitable for rather complex datasets since they have time complexity only linear in the depth of the tree and perform well for a variety of real and synthetic data. Our algorithm can be efficiently used in high performance computing due to the low computational complexity and faster learning.

Chapter 3

Sequential Network Intrusion Detection using Deep Learning

In this chapter, we investigate sequential learning for anomaly detection in an online manner. Here, we use recurrent neural networks (RNN) and convolution neural networks (CNN) to model highly nonlinear relationship in a real life applications. RNNs can be seen as tree networks similar to the hierarchical trees discussed in Chapter 2, that can store the state information. We specifically consider the network intrusion detection [21–23] that has a highly erratic and non-stationary setting for our thesis. We use RNNs to learn the nonlinear relationship between payloads and anomaly label. Intrusion detection in the network loads is a widely studied problem where the relationship between the desired and input data is highly nonlinear and time varying [57, 113]. To model these nonlinear relationship, we introduce highly novel RNNs based algorithms that learn the underlying relationship in an online manner. Hence, this is the first real life application of our online nonlinear algorithms that are trained without any statistical assumptions.

Network intrusion detection systems (IDS) are widely investigated in network and information security literatures [21–23, 48] and form an integral part of the cyber security infrastructure. Fundamentally, an IDS analyzes the network system and the devices and applications attached to it for vulnerabilities against possible intrusions and attacks [47, 109]. Thus, a network system is designed to be robust against the unauthorized access and activities as well as incorporates a mechanism that detects any unwanted but previously unknown efforts to compromise the already secured system. IDSs not only prevent the network from attacks in real time but also provide the information about unknown vulnerabilities. This information can be used to further enhance the systems applications

and hardware abilities for a secure cyber-space [54, 55].

Historically, IDSs use signature based intrusion detection systems where the network traffic is analyzed for known attack types [56, 110, 112]. Signature based detection uses the information and properties of the known attacks and identifies a malicious traffic if it contains such properties. For instance, multiple login attempts, burst of data transmission or request, or the effort to install a malicious software from a remote host might be considered as intrusion [110,111]. However, such systems may fail to identify previously unknown attacks and may consider them as normal traffic. A malicious attack may be initiated by an apparently innocent host such as a regular and genuine member of the network either because the intruder itself is compromised by an attacker or it tries to overuse the authorized services, hence resulting in exhausting the network resources [49, 55]. Due to such issues, a signature based IDS may not detect the intrusion attempts, hence results in a higher misdetection rate [56, 110, 112]. A solution to this problem is to use anomaly detection so that the IDS is trained to identify and pass the "normal" traffic and generate alerts for any activity that falls beyond the "normal" boundaries [44, 57, 113]. Such IDSs are usually suitable for small networks with limited authorized services and designed for specific purposed. For instance, a specific service oriented network such as employee management, if attempted to use for other purposes, may generate an alert of intrusion. Thus, anomaly detection system fails to accommodate novel but legitimate applications. A normal connection that is not in the already designed system or database might be identified as malicious, hence resulting in generating a false alarm. Furthermore, such IDS significantly limits the services and quality of service (QoS) of a network system. Therefore, we seek to design a network intrusion detection mechanism that is universal, robust against novel attacks and achieves significant performance improvement as compared to off the shelf IDS in terms of QoS and intrusion detection rates.

We use online nonlinear models in the classification and anomaly detection framework for the intrusion detection [48]. In this framework, network traffic is analyzed on connection level to decide between "normal" and "attack" traffic. This is achieved by parametrizing the network traffic. The raw data that is transmitted between two connected hosts is extremely complicated and we need a mechanism to generate a set of features that represent such transmission. We then use these features as input to a classifier system that in turn differentiate between normal and malicious connections.

There are various techniques widely investigated for feature generations, e.g., KDD-CUP99 [114]. The KDD-CUP99 dataset uses 41 features to represent each network connection that includes several numeric as well as categorical information. However, the KDD-CUP99 dataset is widely criticized for its lack of ability

to represent a real and comprehensive network traffic, outdatedness and failure to comply with novel internet applications and attacks mechanisms [115, 116]. We investigate online feature extraction, mapping and generation strategies that are efficient and provides as much information as possible to represent the network traffic. These methods need to be universally applicable to any kind of network size, traffic, applications and protocols, without any statistical assumptions on the underlying data, i.e., the network traffic loads.

In [48, 109], the authors propose an IDS where the application payloads can directly be used to detect anomalous behaviors along with other parameters. These payloads can be the HTTP requests and responses, FTP payloads or any other application layer payloads. The payloads consists of binary data or encoded information, e.g., UTF-8 and Base64 [50]. The payloads are dealt similar to a language model extensively used in Natural Language Processing (NLP) applications, and feature mapping techniques are used to generate an input vector or matrix that can be employed to model a classical machine learning system. In this manner, the payloads are analogous to sentences in text documents and the language model is trained so as to classify these payloads as positive (normal) or negative (attack). For the encoding of payloads to generate a language model, one may use $q-$grams along-with frequency counts or term frequency - inverse document frequency (TF-IDF) to generate representative feature vectors [43, 45, 46, 48, 109]. For a instance, in a Base64 encoded payloads with $q = 3$, i.e., tri-grams, a payload is mapped to a sparse feature vector of size $64^3 = 262144$ and each bidirectional connection that consists of two payloads, i.e., source and destination, is encoded by two such huge vectors. This kind of feature mapping results in curse of dimensionality, sparsity and demands higher computational complexity and memory to store and process the information. Furthermore, such encoding techniques, though provide contextual information of the characters in payloads, seem to lack the sequential information. Therefore, we seek encoding and feature mappings that are memory-wise and computationally efficient as well as representative of the contextual and sequential information in the payload. To this end, we use character and payload level encoding using deep learning algorithms. We propose a series of recursive neural network algorithms such as RNNs, CNNs and their combinations to generate a learning model [51, 52, 117].

A big challenge in developing IDS is the lack of availability of benchmark datasets that are essential for training and validation of the model [50]. The datasets need to be representative of different network applications, normal as well as malicious traffic, and mostly known and unknown attack types. Furthermore, we need labeled datasets for the purpose of supervised learning and validation of the trained model even if the training is unsupervised. With the rapid and exponential evolution of internet applications, usage and attacks, it becomes more important that the dataset is recent and futuristic. However,

it is extremely challenging to generate such data either by recording real network traffic or generating it in a controlled environment. For one, the attack instances are extremely rare as compared to normal instances and therefore, it is useless and cumbersome to scan the network in real time to have just a few instances of malicious activity [50]. Furthermore, without any intrusion detection systems, it's not possible to identify and label the attack instances that are required for training the model [50]. A solution is to generate network traffic in a controlled environment that imitates the real network scenario as close as possible. One may generate malicious traffic and use it to launch "attacks" on the hosts, routers or any device on the network using various tools available in the literature [54, 55, 120]. Furthermore, a dataset generated in a certain space and time may not be useful or feasible in another setting because of the scale, application, privacy and outdatedness [119]. Several researchers that use different datasets can not make them public since the data might contain personal information [121]. The FIRST HTTP datasets used in [48] is one such example. LBNL FTP dataset is public since the generators used anonymization to hide IP and physical addresses and other identifiers [121], however, these datasets are unlabeled. Hence, we use ISCX network intrusion dataset because of being modern, representative of and containing instances of various applications, protocols, attack types and scenarios, and publicly available both in form of raw network packets and network flow summaries [50].

In this chapter, we introduce sequential, online and nonlinear anomaly detection algorithms and deep learning techniques for intrusion detection systems. We develop novel algorithms and train our proposed model using ISCX IDS datasets. We provide significant performance improvement with reduced computation complexity. We compare several versions of the recursive neural networks algorithms by using performance measures such as receiver operating characteristics (ROC) curves and the area under the curve (AUC) scores [96, 97]. In Section 3.1, we describe the problem setting in detail. Then, we discuss sequential learning for intrusion detection in Section 3.2. We develop the IDS algorithms step by step in Section 3.3. Finally, we describe the datasets we use in Section 3.4 and provide experimental results with performance evaluation in Section 3.5. We conclude the chapter in Section 3.6.

3.1 Problem Description

All vectors used in this chapter are column vectors denoted by boldface lowercase letters. Matrices are denoted by boldface uppercase letters. For a vector $\boldsymbol{x}$ (or a

matrix vU), $\boldsymbol{x}^T$ ($\boldsymbol{U}^T$) is the ordinary transpose. N is the total number of time-steps and an arbitrary time-step is denoted by t where $0 \leq t < N-1$. The time index of a sequence vector is denoted by t in the subscript, as in $\boldsymbol{x}_t$.

We represent the input (a representation of the payloads) as columns vectors of a matrix $\boldsymbol{X}[n]$, where each column vector is a sequence vector $\boldsymbol{x}_t[n]$ at time t for sample n, i.e., $\boldsymbol{X}[n] = [\boldsymbol{x}_0[n]\ \boldsymbol{x}_1[n]\ \ldots\ \boldsymbol{x}_{N-1}[n]]$. We denote the desired output by a scalar $y[n] \in \{0, 1\}$, where 0 represents "normal" and 1 represents anomaly or "attack". For multi-class labeling where the anomalous traffic is further subdivided in categories, we use the output labels $y \in \{0, 1, ..., C-1\}$, where there are $C-2$ attack types. We use one-hot or categorical labeling for multi-class scenario where in such case the output is a row vector $\boldsymbol{y}[n] = [y_0[n]\ y_1[n]\ \ldots\ y_{C-1}[n]]$ and $\boldsymbol{y}[n] \in \mathbb{R}^{C-1}$.

3.2 Sequential and Contextual Learning for Intrusion Detection

We use a sequential learning framework for the intrusion detection by analyzing the payloads. We consider a connection-wise setup where in connection oriented transmission, such as in the case of TCP, to define a connection by the establishment between two connected hosts [50, 95]. In case of connection-less, e.g., UDP, since a "connection" is not defined implicitly, we use a vague definition for it by considering the hosts IPs, ports, underlying applications and protocols, and the termination point such as time out [50, 95]. In general, a connection is considered as transfer of data between two certain hosts under some defined protocols. Each connection usually contains two payloads, a source payload and a destination payload. For instance, in HTTP a connection request may be considered as source payload and the response received from the requested host is considered as destination payload. We use these connection-wise pairs of payloads to identify network intrusions by analyzing them in a sequential manner. That is, a payload comprising of several characters, is used as input to the IDS in a sequential manner, i.e., one character at a time, in it's natural order. Thus, not only the set of characters gives information about the payload but also their individual positions inside the payload. Furthermore, we investigate a mechanism to extract the contextual information, i.e., using the relative information with respect to the neighboring characters. For this purpose, we use a language model like NLP, by incorporating the grammar and syntax of the payloads. For the purpose of explanation, we use the analogy of text processing, such as each connection-wise pair of payloads is considered a text document that contains two sentences each corresponding to source and destination payloads respectively.

3.3 Recursive Neural Networks

Recursive Neural Networks (RvNN) are tree structures with a neural network at each node [51, 52, 117, 118]. They are mostly useful for the problems with structural dependencies such as NLP, image processing, video streams etc. In RvNN, each layer and the comprising nodes "look" into a different, though not necessarily exclusive, aspect of the problem. For instance, in NLP, a text document comprises of sentences that are made of words, RvNN layers are used for the encoding of document, the sentences, words and even characters in structural manner, hence, using a deep learning framework. Subsets of the RvNN are RNNs and CNNs. We propose a language model that uses a combination RNNs and CNNs for level-wise learning of the network traffic.

3.3.1 Long Short Term Memory Units (LSTMs)

For the sequential and structural learning, we use RNNs that extract the sequential information and to some extent the contextual information from the input. We use LSTM units to build RNNs models [122,125,130]. LSTM based RNNs are useful for long term dependencies since they connect the previous information as well as the current input to the current task. LSTMs are long chains of memory units where each unit takes the input at time t, the information gathered from the previous units till time $t-1$, and generate a candidate for the output as well as a state to be passed to the next unit. In Fig. 3.3.1, we show connected LSTM units that works on the input $\boldsymbol{x}[t]$. We unroll the various operation stages of an LSTM unit in the following.

An LSTM unit takes three inputs, i.e., the cell state, the output of the previous unit and the current input, and generates two outputs, i.e., new cell state (or memory) and an output. In the first step, LSTM decides which information to discard that is propagated from the previous units. This operation is called forget gate and is described by the following equation.

$$\mathbf{f}_t = \sigma\left(\mathbf{W}^{(f)}[\mathbf{x}_t, \mathbf{h}_{t-1}] + \mathbf{b}^{(f)}\right), \tag{3.1}$$

where σ is a logistic function applied on each element of the vector inside braces, $\mathbf{W}^f \in \mathbb{R}^{d\times d}$ are the weights of forget gate with $\mathbf{b}^f$ as bias, and $\mathbf{h}_{t-1}$ is the output of the previous LSTM unit. In the next step, we determine the new information $\mathbf{i}_t \in R^d$ that is going to be stored in the cell state and calculate the candidate for the cell state because of the new information, i.e.,

$$\mathbf{i}_t = \sigma\left(\mathbf{W}^{(i)}[\mathbf{x}_t, \mathbf{h}_{t-1}] + \mathbf{b}^{(i)}\right), \tag{3.2}$$

$$\tilde{\mathbf{C}}_t = \tanh\left(\mathbf{W}^{C}[\mathbf{x}_t, \mathbf{h}_{t-1}] + \mathbf{b}^{C}\right). \tag{3.3}$$

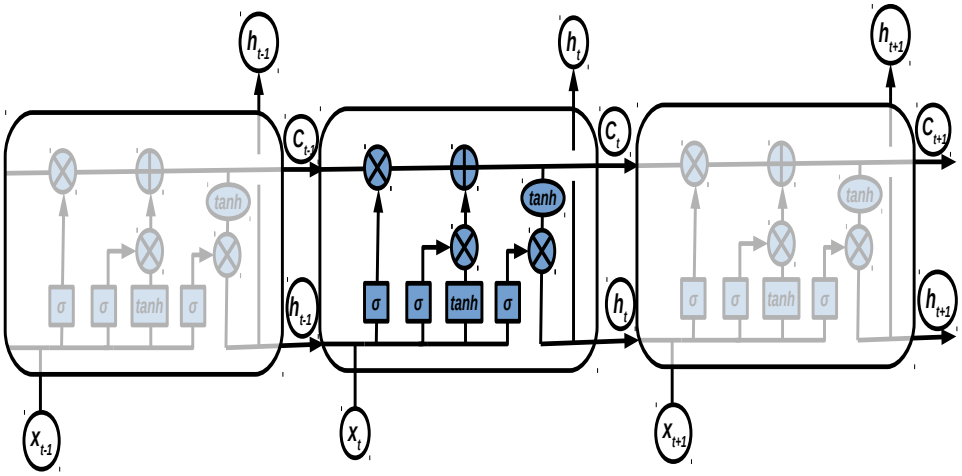


Figure 3.1: Long Short Term Memory Units (LSTM)

Then the cell state is updated using the following equation,

$$\mathbf{C}_t = \mathbf{f}_t * \mathbf{C}_{t-1} + \mathbf{i}_t * \tilde{\mathbf{C}}_t, \tag{3.4}$$

that is the partial information retained from the previous unit added to the new information. Finally, we calculate the current output $\mathbf{h}_t$ of the LSTM unit as follows,

$$\mathbf{o}_t = \sigma\left(\mathbf{W}^{(o)}[\mathbf{x}_t, \mathbf{h}_{t-1}] + \mathbf{b}^{(o)}\right), \tag{3.5}$$

$$\mathbf{h}_t = \mathbf{o}_t * \mathbf{C}_t. \tag{3.6}$$

The wight matrices $\mathbf{W}^f, \mathbf{W}^i, \mathbf{W}^C, \mathbf{W}^o$ and bias vectors $\mathbf{b}^f, \mathbf{b}^i, \mathbf{b}^C, \mathbf{b}^o$ are updated by using back-propagation at each sample input during training. In a single layer LSTM network, the output $\mathbf{h}_t$ of last LSTM unit is further passed through a logistic regression unit to estimate the final output, i.e., the label of the connection or payload. In principle, the LSTM network turns a matrix input $\boldsymbol{X}[n] \in \mathbb{R}^{d \times N}$ to a feature vector $\mathbf{h}_{N-1}[n] \in \mathbb{R}^D$ that is used in a classic machine learning problem. However, in practice multiple layers of LSTMs are used to produce final output. In such case, all the outputs of the previous layers are used as input to the next layer and in the final layer only the last output is used.

LSTM RNNs learn the inherent sequential model as well as the contextual and syntactic information since they incorporates memory from the sequential inputs. However, to make the model further robust specifically to learn the character level encoding and the mutual information, we add CNNs on top of LSTM RNNs. In next, we describe CNN and then proceed to describe the overall recursive learning model.

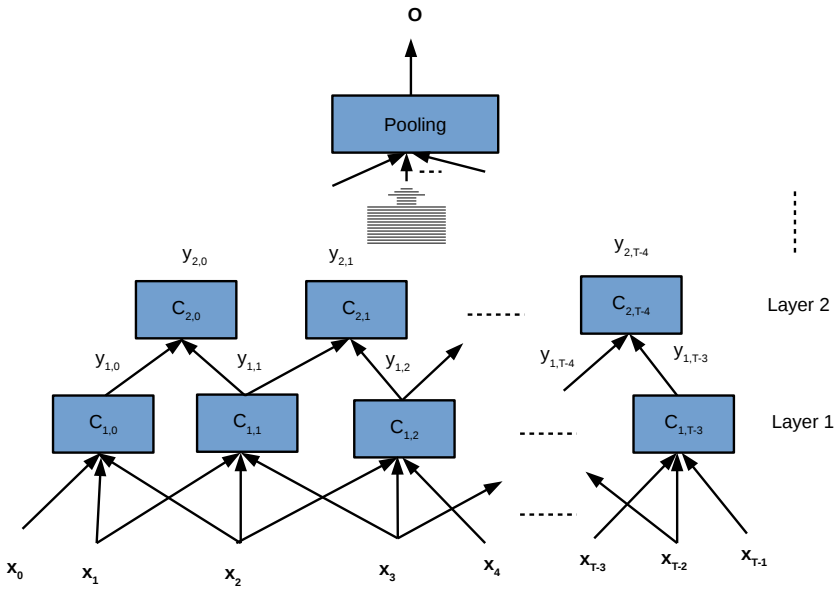


Figure 3.2: Multi-layer convolution neural network

3.3.2 Convolutional Neural Networks (CNNs)

For an input sequence $\boldsymbol{X} = [\boldsymbol{x}_0\ \boldsymbol{x}_1\ \ldots\ \boldsymbol{x}_{T-1}]$, we use a multiple layer CNN such as the one shown in Fig. 3.2. Here, the output of a CNN unit is given as,

$$y_{i,j} = \Sigma_{k=j-r+2}^{j+r-2} \boldsymbol{w}_{i,j,k}^T \boldsymbol{x}_k, \tag{3.7}$$

where r is the order of the convolution unit, e.g., $r = 3$ of the first layer in Fig. 3.2, i is the index of the layer, j is the index of the unit inside the layer, and T is the number of input sequence vectors. The first convolution layer of order r, i.e., $i = 1$, takes an input $\boldsymbol{X}$ comprising of the sequence vectors $\boldsymbol{x}[0], \boldsymbol{x}[1], \ldots, \boldsymbol{x}[T-1]$ and produce outputs scalar outputs $y_{1,0}, y_{1,1}, \ldots, y_{1,T-r}$ or a vector $\boldsymbol{y}_1 \in \mathbb{R}^{T-r+1}$. The weights $\boldsymbol{w}_{i,j,k} \in \{0, 1\}$ are $d-$ dimensional vectors. We use F such convolution filters at each layer, hence, resulting in a $(T - r + 1) \times F$ dimensional matrix or array [53, 126, 128]. The process is repeated L times and hence reducing the number of rows at each layer. We also use p order pooling at intermediate levels, between the layers, i.e.,

$$\mathbf{o} = pool(y_{I,l}) \tag{3.8}$$

where $pool(.)$ is either average pooling, i.e., $pool(z_l) = \frac{1}{L}\Sigma_{l=1}^{L} z_l$, or max pooling, i.e., $pool(z_l) = max_{l=1}^{L}(z_l)$, and L is the number of inputs to the pooling unit. The final output is a sequence array of dimensions $F \times P$ where P is the number of convolution or pooling units at the last layer.

We consider an input sequence, $\boldsymbol{X} \in \mathbb{R}^{d \times T}$ where $\boldsymbol{x}_t \in \boldsymbol{X}$ is a d dimensional sequence vector such that $\boldsymbol{X} = \{\boldsymbol{x}_0, \boldsymbol{x}_1, ...\boldsymbol{x}_{T_i-1}\}$. We use a I layer convolution

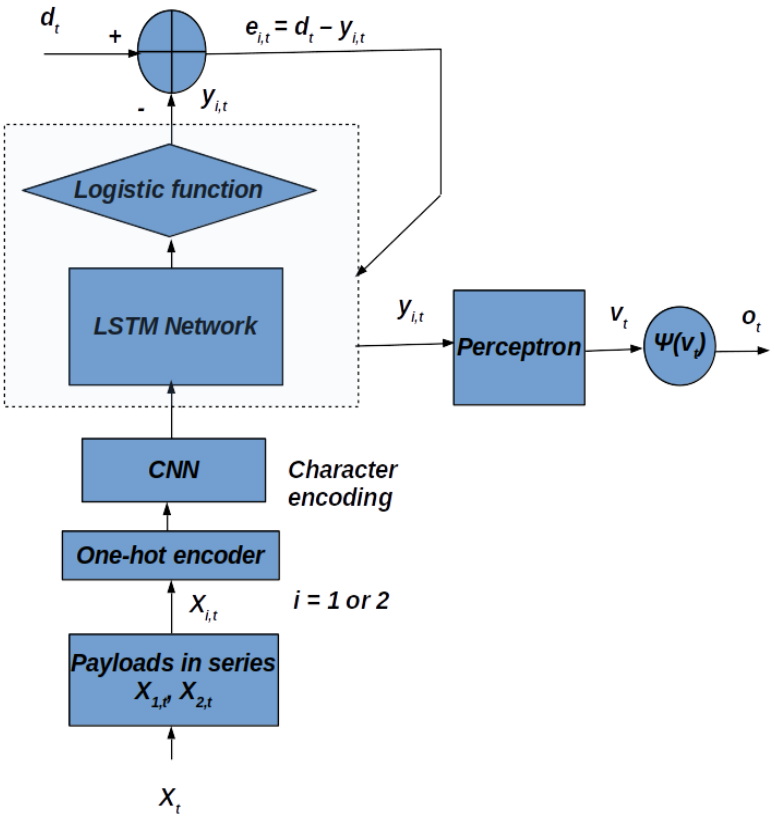


Figure 3.3: The proposed IDS model

network, each comprised of $T_i - r_i - 1$ units where r_i is the order of the convolution unit and T_i is the number of inputs at layer i. We pass the final output $\tilde{\boldsymbol{X}} \in \mathbb{R}^{P \times F}$ to the LSTM network for payload and connection-level encoding, such that the temporal input is $\tilde{\boldsymbol{x}}_t \in \mathbb{R}^P$ where $0 \leq t \leq F - 1$.

3.3.3 Bi-directional LSTMs

We further improve our learning model by introducing bi-directional LSTMs. In the model described in subsection 3.3.1, the input is given to the LSTM network in its natural sequential order, i.e., $\boldsymbol{x}_0 \longrightarrow \boldsymbol{x}_1 \longrightarrow \ldots \longrightarrow \boldsymbol{x}_{T-1}$, where the output at time t depends on the current input $\boldsymbol{x}_t$ as well as the information learned till $t-1$. However, in a natural language model, the meaning of a word in a sentence, might depend on the previous as well as the following words. Therefore, we also use an LSTM network in the backward direction, i.e., $\boldsymbol{x}_{T-1} \longrightarrow \boldsymbol{x}_{T-2} \longrightarrow \ldots \longrightarrow \boldsymbol{x}_0$. We show in Section 3.5 that these bi-directional LSTMs significantly improve the performance of our model.

The overall model for intrusion detection of connection-wise payloads and character-level as well as payload level recognition is shown in Fig. 3.3. To this end, we propose a deep learning model for online and nonlinear intrusion detection of network payloads. Each connection-wise payload is passed sequentially through multiple CNN layers for character-level and contextual recognition. The encoded outputs of CNNs are then used as sequential inputs to the LSTMs network. We finally use a sigmoid function to determine the probability of intrusion based on the current payload. Since most of network connections are bidirectional and contains two payloads, the process is repeated for both the payloads. In this case, the final output is determined by a linear combination of each output.

3.4 Datasets

Here we introduce and describe our benchmark datasets. We use ISCX 2012 intrusion detection datasets [50,127]. These datasets are generated in a controlled and simulated environment that greatly imitates a real network scenario with several application and attack types.

The ISCX 2012 intrusion detection evaluation dataset consists of the following 7 days of network activity (normal and malicious):

- Friday, 11/6/2010, Normal Activity. No malicious activity.
- Saturday, 12/6/2010, Normal Activity. No malicious activity.
- Sunday, 13/6/2010, Infiltrating the network from inside + Normal Activity.
- Monday, 14/6/2010, HTTP Denial of Service + Normal Activity.
- Tuesday, 15/6/2010, Distributed Denial of Service using an IRC Botnet.
- Wednesday, 16/6/2010, Normal Activity. No malicious activity.
- Thursday, 17/6/2010, Brute Force SSH + Normal Activity.

The dataset is available both in form of raw network packets (pcap) files and connection level network flow summaries. We extract the payloads and labels from the flow summaries and use them to train our IDS models. The payloads are Base64 encoded and comprise of sequence of 64 unique characters, i.e., $\{A, B, ..., Z, a, b, ..., z, 0, 1, ..., 9, +, \backslash\}$.

3.4.1 Pre-processing

We use one-hot encoding to convert each character in the input sequence to a $64-$ dimensional vector. The payloads are of varying lengths, upto 2800, we use 99 percentile to limit the length of payloads that is $N = 1358$. If the length of a payloads is $l < N$, we pad the one-hot encoded output by $N - l$ vectors comprising of -1's. We then use a masking layer in the deep learning model such that when the input is -1, the weights are not updated. In this manner, each sample input is comprised of two 64 dimensional arrays, one for each payload.

3.5 Experiments and Results

In the following, we provide our experimental results and analysis using the proposed techniques on the ISCX 2012 datasets.

We use various settings and combinations of algorithms and models to perform several experiments. We split the dataset in $80\% - 20\%$ for training and testing in all the experiments. The 80% datasets is further split for $5-$fold cross-validation during training before applying the learned model to the test dataset. Then we test our trained model on the 20% data and calculate the true positive, false negative, false positive and true negative rates for several thresholds. We compare the performance of learned models under different settings by plotting the receiver operating characteristics and determining the area under the curve (AUC) scores. In the experiments where CNNs are used, we choose $I = 3$,$F_i = \{256, 192, 128\}$ and $r_i = \{5, 3, 3\}$ for $i = \{1, 2, 3\}$ respectively. Furthermore, for LSTM network, we use three layers with 128 units in the hidden (middle) layer.

In the first experiment, we compare the performance of models with and without character level encoding using CNN and with using pooling in the CNN layers. The results in Fig. 3.4 show that employing CNN significantly improves the performance. Furthermore, max pooling seems to have better results than average pooling.

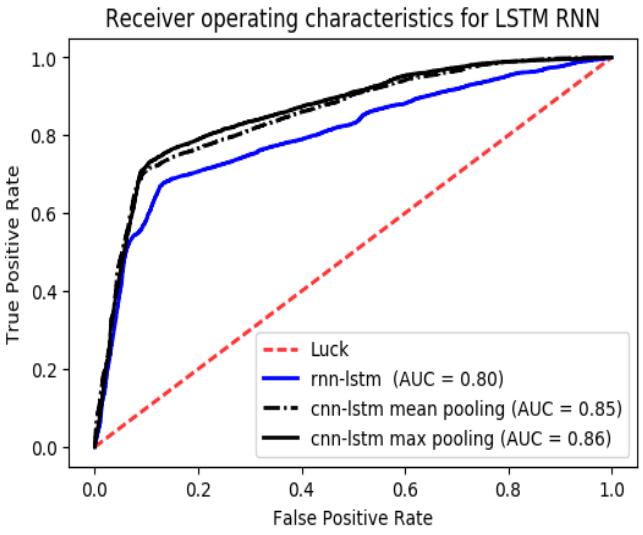


Figure 3.4: Receiver operating characteristics (ROC) for intrusion detection using CNN and RNNs-LSTM

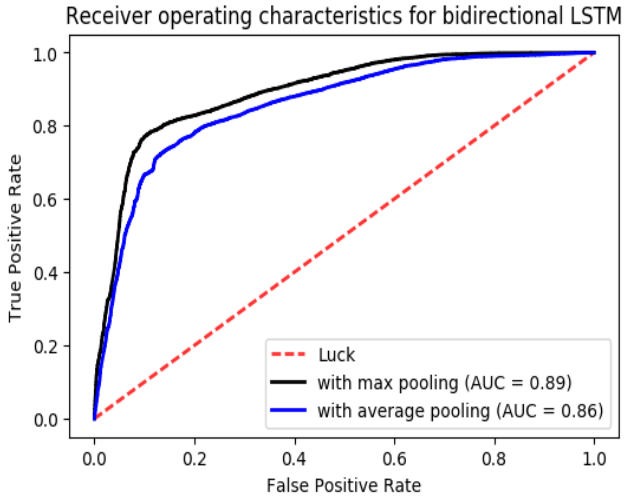


Figure 3.5: Receiver operating characteristics (ROC) for intrusion detection using CNN and RNNs-BiLSTM

In the first LSTM layer, we use all the outputs $\boldsymbol{h}_t$ as input to the next layer, whereas in the final LSTM layer, we only use the final output, i.e., $\boldsymbol{h}_{T-1}$ where T is the number of hidden units. We then use $\boldsymbol{h}_{T-1}$ vector as input to a perceptron and determine the final output via a sigmoid function, $\sigma_f(y) = \frac{1}{1+exp^{-y}}$.

In the next experiment, we use bi-directional LSTMs at the RNNs layer on top of CNNs. There is a significant further improvement in the performance as can be seen by the AUC scores in Fig. 3.5. Here, we plot ROC curves for the choices of average and max pooling at the CNN step. Furthermore, we compare the performance of deep learning algorithms using uni-directional and bi-directional

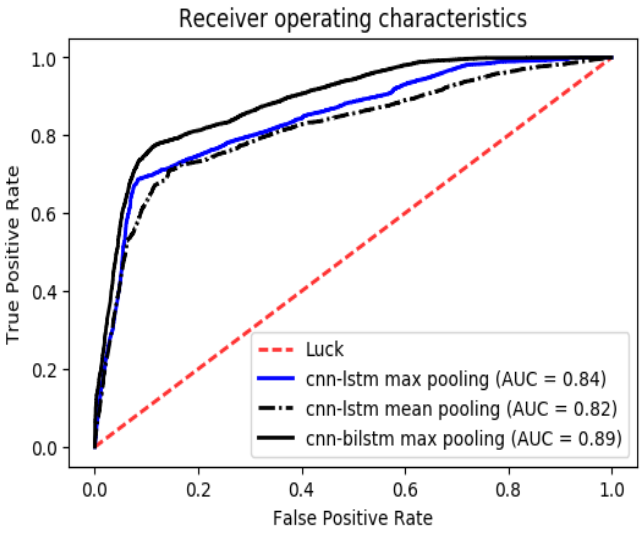


Figure 3.6: Receiver operating characteristics (ROC) for comparison of unidirectional and bidirectional LSTM
LSTM units and show our results in Fig. 3.6.

We next replace the LSTM units with gated recurrent units (GRU) for the recurrent learning [123, 124]. We use bidirectional RNNs in both case and the results in Fig. 3.5 show that LSTM units perform slightly better than GRU.

Finally, we summarize the performance of our deep sequential learning architectures by comparing them with classic machine learning algorithm. We use $q-$grams with and TF-IDF that maps the Base-64 encoded payloads to 64^q dimensional, highly sparse feature vectors. We then use these vectors to train random forests, support vector machines and decision tree classifiers. The results are shown in Table 3.1. Note that, the $q-$grams encoding is inefficient in terms of computational complexity and memory requirement since, even for online setting, requires huge amount of memory.

3.6 Discussion

We investigate online nonlinear modeling of the network IDSs. The network traffic is highly non-stationary and erratic. Therefore, we introduce a structural learning approach using deep neural networks to model the IDS. We propose sequential learning by analyzing the application payloads and develop a language model that is independent of the network applications. We specifically use a combination of recursive neural networks to learn the payloads pattern for character, sequence and syntax level learning. As a result, we achieve significant performance, in term

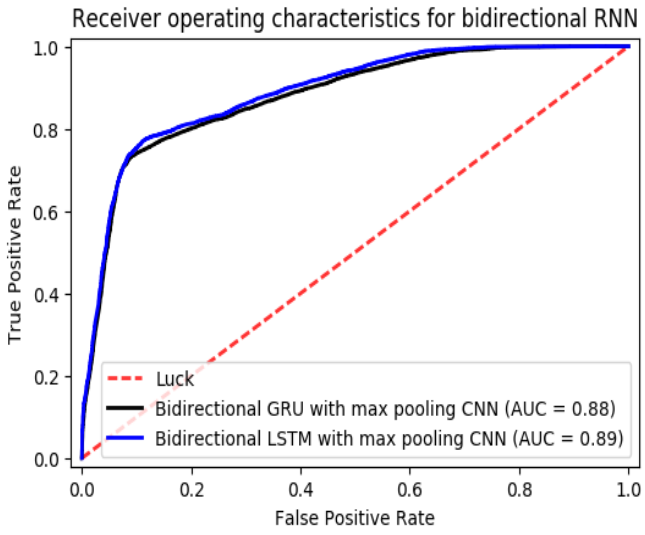


Figure 3.7: Receiver operating characteristics (ROC) for comparison of bidirectional RNNs with LSTM and GRU

Algorithms	Encoding	RNNs layers	RNNs units	CNN layers	CNN pooling	AUC score
LSTM RNNs	One-hot	2	128	0	N/A	0.8021
CNN-LSTM 1	One-hot	2	128	3	Average	0.8492
CNN LSTM 2	One-hot	2	128	3	Max	0.8632
CNN BiLSTM	One-hot	2	128	3	Average	0.0.8605
CNN Bi-GRU	One-hot	2	128	3	Max	0.8841
Decision Trees	trigram, TF-IDF					0.6955
Random Forests	bigram, TF-IDF					0.75
SVM	bi-gram, TF-IDF					0.9013

Table 3.1: Intrusion Detection scores on ISCX dataset

of detection of any malicious activity, by performing the proposed methods on a high quality and representative of the real networks dataset.

Chapter 4

Sequential Churn Analysis of Cellular Network Users

In this chapter, we investigate online churn detection as a real life application of nonlinear learning models, in big data perspective. We consider the problem of customer churn analysis by analyzing cellular network users data. We specifically use online learning algorithms introduced in Chapter 2 and Chapter 3 for the churn prediction. In customer relationship management, retaining the existing users is of utmost importance [131]. Attrition or churn is when a customer leaves an organization or service for the competitor. We seek to analyze cellular customer data for churn analysis and prediction [59]. Generally, a customer would decide to join or leave a network based on several different reasons, for instance, services and their quality, cost, availability of service, customer support and so on. Customer churn is usually a big issue for the organizations specially when key customers decide to leave for a better a competitor service provider [132]. Therefore, we use the current and past customers data to analyze the common grounds for customer attrition. The customer churn prediction helps in identifying and solving the issues that results in attrition. The analysis can be used for possible retention of the current customers.

The cellular customers data contains two different kinds of information or features, i.e., static and sequential. The static features remain constant for a customer for a certain period of time, e.g., gender, age of the customer, start date, etc. The sequential features may vary over time, e.g, usage statistics, bill payments, location, number and type of services etc. Therefore, the network service providers record and keep customer data for every month. We seek to analyze the customers data to identify potential churners.

The problem of churn analysis can further be divided into following three aspects.

- Binary churn detection, i.e., to decide whether a certain customer would churn or not based on his/her available data. This makes it a classical binary classification problem that can be dealt with using machine learning
- Multi-class churn prediction - to identify in which month a customer would churn in a series of future months.
- A churn probability analysis to determine the probability of churn of a certain customer in a certain period of time, i.e., a regression problem.

The state-of-the-art classification algorithms, such as the Gradient Boosting Trees, Random Forests etc., are inadequate for churn detection due to the sequential nature of the data since these methods cannot directly incorporate the temporal information [133, 134]. In [60, 135], the authors demonstrate that using a sequential classification algorithm has an improved performance in churn detection when compared to Logistic Regression and Multi-Layer Perceptron. These findings lead to further investigation in sequential learning for boosting the churn detection. Furthermore, the customer dataset needs to cleaned and preprocessed for a more robust analysis [136]. Some further issues with the customers data include missing information, the heterogeneous nature of the data types, i.e., numeric, categorical, binary etc., and the existence of features which are unnecessary or less important as compared to others [136]. Therefore, for a robust analysis we focus on the pre-formatting of the data prior to apply the machine learning algorithms.

Furthermore, we study online sequential logistic regression for churn detection in cellular networks when the feature vectors lie in a high dimensional space on a time varying manifold. We escape the curse of dimensionality by tracking the subspace of the underlying manifold using a hierarchical tree structure proposed in Chapter 2. We use the projections of the original high dimensional feature space onto the underlying manifold as the modified feature vectors. We provide several results with real life cellular network data for churn detection.

The outline of the chapter is as follows: In Section 4.1 we formally describe the problem setting. Furthermore, we explain the issues and challenges with the real life datasets and their generic solutions. In Section 4.2 we introduce our real life cellular network dataset. We clean and preprocess the dataset and explain the procedure. Once the dataset is ready, we use it for training and validation of several state-of-the-art classification algorithms. We perform several experiments and give the results and comparison using various performance measures, e.g.,

Receiver Operating Characteristics (ROC) curves and area under these curves (AUC) in Section 4.3 [96, 97]. Finally, we briefly describe our conclusions of our contributions.

4.1 Problem Description

All vectors used are column vectors, denoted by boldface lowercase letters. Matrices are denoted by boldface uppercase letters. For a vector $\boldsymbol{v}, \| \boldsymbol{v} \|^2 = \boldsymbol{v}^T\boldsymbol{v}$ is squared Euclidean norm and $\boldsymbol{v}^T$ is the ordinary transpose. $\mathbf{I}_k$ represents a $k \times k$ identity matrix. We investigate online logistic regression using high dimensional data, i.e., when the dimension of data $D \gg 1$. We observe a desired label sequence $\{y[n]\}_{n\geq 1}, y[n] \in \{-1, 1\}$, and regression vectors $\{\boldsymbol{x}[n]\}_{n\geq 1}, \boldsymbol{x}[n] \in \mathbb{R}^D$, where D denotes the ambient dimension. The data $\boldsymbol{x}[n]$ are measurements of points lying on a submanifold $S_{m[n]}$, where the subscript $m[n]$ denotes the time varying manifold, i.e. $\boldsymbol{x}[n] \in S_{m[n]}$. The intrinsic dimension of the submanifolds $S_{m[n]}$ are d, where $d \ll D$. The submanifolds $S_{m[n]}$ can be time varying. At each time n, a vector $\boldsymbol{x}[n]$ is observed. Then $z[n]$ is given by:

$$z[n] = f_n(\boldsymbol{x}[n]), \tag{4.1}$$

where $f_n(.)$ is a nonlinear, time varying function and $z[n] = 0$ is a separating hyperplane between the two classes. The instantaneous regression error is given by: $e[n] = y[n] - z[n]$. The estimate of the desired label is calculated by the following logistic function,

$$\hat{y}[n] = h(z[n]), \tag{4.2}$$

where $h(.)$ is a signum function, i.e.,

$$\begin{aligned} h(z) &= \ 1, \quad if \ z \geq 0, \\ &= -1, \quad if \ z < 0. \end{aligned}$$

We approximate the nonlinear function $f_n(.)$ by piece-wise linear models such that the $\mathbb{R}^D$ regressor space is divided into various regions. We assume that there is a linear relationship between $\boldsymbol{x}[n]$ and $z[n]$ in each region. We use a hierarchical tree structure that partitions the regressor space into various regions.

In the sequential setting, the input $\boldsymbol{x}_t[n] \in \mathbb{R}^d$ is temporal sequence and for user n, the input is a matrix $\boldsymbol{X}_n \in \mathbb{R}^{d\times M}$, i.e.,

$$\boldsymbol{X}[n] = [\boldsymbol{x}_0[n] \ \boldsymbol{x}_1[n] \ \ldots \boldsymbol{x}_{M-1}[n]]. \tag{4.3}$$

Here, the columns of matrix $\boldsymbol{X}[n]$, i.e., $\{\boldsymbol{x}_0[n], \boldsymbol{x}_1[n], \ldots, \boldsymbol{x}_{M-1}[n]\}$ represent the temporal sequence vectors with elements representing the features. For instance,

in a $j \times k$ matrix, there are j features and k time instances (months). In the sequential setting, the input can be a single column vector, representing the current value of each feature for a certain user, or a k columns matrix containing the data of current and $k-1$ previous months.

We summarize the churn detection from raw input data as follows:

1. Feature extraction for the users in the training set.
2. Preprocess for categorical feature encoding and sequential features.
3. Batch learning model to train the classifier using known labels.
4. Sequentially update the trained model with new data.
5. Use cross-validation and unlabeled data to verify the model persistence.

In the sequential learning, we use the instantaneous input data, intermediate prediction of the desired label (probability of churn) and the accuracy measure of the decision to train the classifier. We use Keras and Scikit-Learn python libraries in our experiments.

4.2 Datasets

We use the AVEA Telekom (currently TURK Telekom) real life dataset for our experiments [58,59]. The dataset consists of sequential features of 6 months. The total number of users are 10000 each having a 40 dimensional feature vector. The first feature, user ID is irrelevant while another feature "tariff type" is constant for all users. The remaining feature set consists of 25 time-varying features, such as the ones related with billing, network usage etc, and 13 stationary features, e.g., device type, age, gender, start date, expiry date etc. Among the stationary features, seven are categorical while the rest are numeric. All the sequential features are numeric. The detailed description of all the features is given in Fig. 4.1.

4.2.1 Preprocessing and Cleaning

We preprocess the dataset for reshaping to be used for the sequential algorithms, conversion of categorical to numerical variables and imputing missing data. For

Features	Description
Categorical No. of features = 7	Gender Device Type Home Change Work Change CRM segment Value segment Lifestyle segment
Stationary Numerical No. of feature = 6	User Age Last reload date Expiry date Hotline date Last reload amount Age of line
Sequential Numericals No. of features =25	Minutes per month with subscription Mintues per month without subscription Duration Payment Outstanding bills internet usage statistics Call drop rate No. of call drops Helpline usage etc.
Output Labels	Churn No Churn

Figure 4.1: Summary of the features of AVEA dataset

the batch setting, all the features of a certain users are taken as a $D-$ dimensional vector $\boldsymbol{x}_n$, where D for our dataset is $25\times6+13 = 163$, with 25 sequential features for 6 months and 13 stationary features. In the sequential setting, the features of a certain user n are represented by matrix $\boldsymbol{X} \in \mathbb{R}^{M\times d}$, where $M = 6$ and $d = 38$. The stationary features are repeated for each month. Furthermore, we convert the categorical features into numeric features using one-hot encoding. Finally, we process the dataset for missing values by inserting the mean of each feature and adding an extra feature to denote the missingness [136]. The final cleaned and processed dataset has 310 features in the batch setting and represented by 72×6 matrix in the sequential setting, i.e., a 72 dimensional vector for each month.

The available output data consists of "Churn" labels for 11 months including the 6 month period for which the input data is recorded and further 5 months in the future. "Churn" is represented by 1 while "no churn" is represented by 0. For binary classification, i.e., where the aim is to identify churn status for a certain customer, the output bits are added resulting in 1 if the user churn during any month and 0 otherwise. For multi-class churn prediction, the labels are encoded by 12 distinct classes $\{0, 1, \ldots, 11\}$, each class number representing the month during which the churn occurred and 0 representing "no churn".

4.3 Results

In this section, we first use the adaptive hierarchical trees (AHT) algorithm described in Chapter 2 for the online churn detection using high dimensional cellular network data of $N = 10000$ users. The original dimension of the regressor vectors $\boldsymbol{x}[n]$ is $D = 164$ and the desired labels $y[n] \in \{0, 1\}$. The dataset contains missing values and we observe a subset of the D features for each user. The dataset is also imbalanced as the "no churn" class contains approximately 77% of samples. Therefore, we perform oversampling on the minority class to balance the dataset [1]. We perform online logistic regression choosing $d \in \{10, 20, 30, 40, 50, 60\}$. We plot the success rate versus d for $K \in \{1, 2, 3\}$ in Fig. 4.2.

The logistic regression in $\mathbb{R}^d$ shows great improvement in performance for churn detection in high dimensional cellular data for $d << D$. In Fig. 4.2, we show that choosing a larger d improves the performance of the algorithm, however, it reaches saturation and further increasing d results in over- fitting and increased complexity. Hence, using only $d = 40$ features instead of $D = 164$, we achieve a success rate of over 90% with simple linear discriminant functions within each piecewise region. It is interesting to note that for smaller K, the algorithm performs worse for $d < 40$, however, it reaches the same success rate as $K > 2$ as we choose $d > 40$. We also perform logistic regression on the original $\mathbb{R}^D$ feature vectors using online linear discriminants and the success rate is as low as 32%. Therefore, the proposed algorithm not only reduces the computational complexity but also produces outstanding success rate.

We next compare the performance of AHT algorithm with well known classification algorithms, i.e., SVM and classification trees while using these algorithms offline. We show that our algorithm produces comparable results to batch SVM and classification trees with much less computational and time complexity as our algorithm uses linear discriminants in the online setting. The adaptive hierarchical trees (AHT) algorithm for logistic regression uses ensemble learning [1] as it linearly combines several linear discriminants and produces outstanding results with much less complexity than SVM and Classification trees (CTrees). We also use a combination of our algorithm and SVM by using SVM within each piecewise region instead of linear discriminants (AHT-SVM) and achieved 99% success rate with a false alarm rate = 0.01. However, this approach is much complex than the original AHT and is not suitable for online learning. The results are shown in Table 4.1.

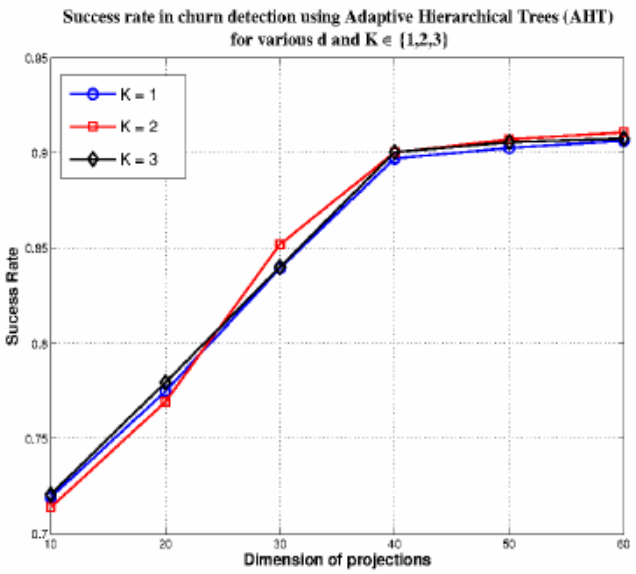


Figure 4.2: Success rate in Churn detection using Adaptive hierarchical Trees algorithm for logistic regression with various choices of d and K

Algorithm	Success rate	False alarm rate	Detection rate
AHT	0.91	0.04	0.93
SVM	0.985	0.03	0.97
CTrees	0.94	0.04	0.95
AHT-SVM	0.99	0.01	0.99

Table 4.1: Success rate, false alarm rate and detection rate for AHT ($d = 40, K = 3$), SVM, Classification Trees and AHT-SVM

In the next set of experiments, we use the AVEA dataset for bi-class churn detection while considering the batch setting. In Fig. 4.3, we show the receiver operating characteristics of RandomForest classifiers with several choices of number of estimators [138]. We show that as the number of estimators increases, the classifier performance gets better on an unlabeled dataset. However, after using a certain number of estimators, the saturation occurs as in this example, there is no further increase in the AUC scores using more than 50 estimators. We also use $5-$fold cross-validation for the Random Forest classifier with 50 estimators and plot the ROC curves for each fold in Fig. 4.4.

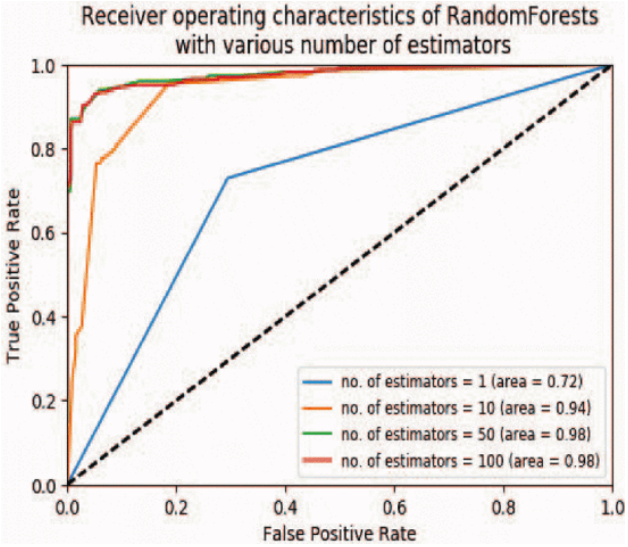


Figure 4.3: Receiver operating characteristics (ROC) for random forest classifiers

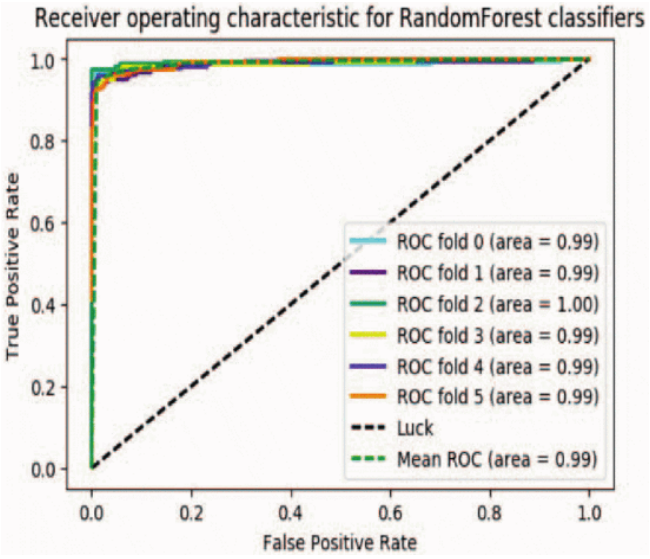


Figure 4.4: Receiver operating characteristics (ROC) for random forest classifiers with 5− fold cross-validation

Algorithm	Mean AUC scores	Prediction Accuracy	Remarks
SVM	0.89	0.92	
RFC	0.985	0.99	50 estimators
GBC	0.98	0.99	10 estimators
VRNN	0.93	0.97	3 layers, 32 hidden neurons
LSTM	0.96	0.96	32 input, 16 hidden layers

Figure 4.5: AUC scores and prediction accuracy of churn predictors

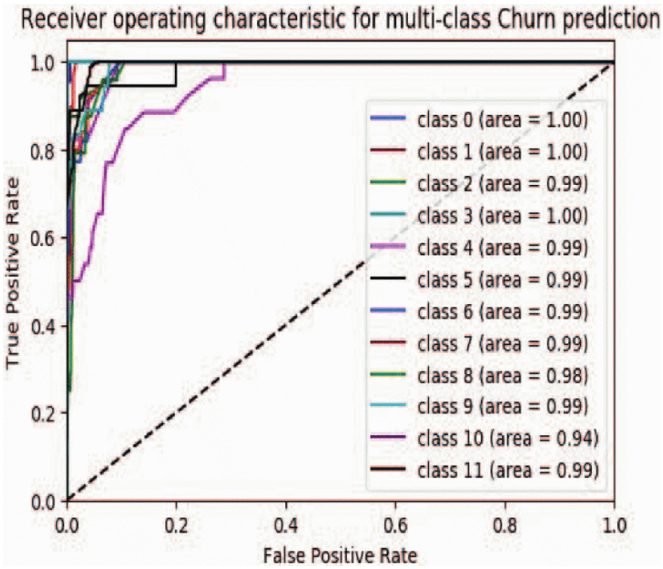


Figure 4.6: Receiver operating characteristics (ROC) for multi-class churn prediction using random forest classifiers

In the following experiment, we compare the performance of several simple and ensemble classifiers namely Support Vector Machine (SVM), Random Forests, Gradient Boost [137], Vanilla RNN and LSTM [122]. For the Vanilla RNN and LSTM, we use the sequential setting as described earlier. The input is taken as a 6 dimensional sequence vector for each feature where each dimension represents a certain month. This makes the complete input dataset as an array of matrices $\boldsymbol{X}[n]$ where $\boldsymbol{X}[n] \in \mathbb{R}^{6\times 72}$. Each user data in the training set is used to train the RNN model sequentially (monthly data). The average AUC scores for each algorithm are shown in Fig. 4.5 where we use the same batch setting as the previous experiment for the first three classifier models while sequentially training the Vanilla RNN with 60 neurons at the input and 30 at the middle hidden layer.

The results in the last two experiments seem promising, however, we assume that all the training data is available in advance (batch). The real and more practical scenario may be different where we are more interested in classification models that are trained instantaneously and sequentially. An adaptive classifier model is able to train on instantaneous input and can incorporate the time varying nature of the dataset and input-output relationship in a more robust manner. Furthermore, sequential algorithms are computationally efficient since they do not necessarily store and process all the previous data.

Finally, we perform experiments to predict the churn probability of a user during each month. Among the 11 months for which the churn labels are available, the input data is recorded only for the first 6. Therefore, our goal here is not only to detect the churn in those six months but also predict churn in the next five months based on the history. In this experiment, we consider churn in each

Future Month	RandomForest	GradientBoost	SVM
1	0.98	0.99	0.94
2	0.985	0.99	0.945
3	1.0	0.99	0.98
4	0.98	1.0	0.89
5	0.96	0.98	0.92
no churn	0.99	0.992	0.975

Figure 4.7: AUC scores for monthly churn predictions

month as a separate class that in turns makes it a 12-class classification problem including the cases when "no churn" is recorded. We use Gradient Boosting Trees algorithms measure the prediction performance for each class. The results are shown in Fig. 4.6.

The results in Fig. 4.6 describe that an optimally trained and tuned ensemble of classifiers can be used to predict the churn of customers well in advance with high accuracy. However, the essential part of learning is correct features selection and suitable manipulation from the users data. Moreover, because of the temporal variation and evolution of the dataset, the previous values can be discarded and in turn make the learning recursive. In addition, several features are linearly dependent on others and a dimensionality reduction is both essential and efficient in terms of performance boost and computations [93]. We compare the prediction performance of Random Forest, GradientBoost and SVM for each of the five future months (the last five months) for which the input data is not available. The AUC scores are shown in Fig. 4.7.

4.4 Discussion

In this chapter, we investigate online churn detection as a real life application of nonlinear, non-stationary models. We consider the issue of customer churn and its prediction from the history and other information available to the service provider from a machine learning and data analysis perspective. We use real life cellular network users' dataset and work on a sequential feature set for each user. Note that the data is highly erratic with missing information and non-stationary behavior. We describe various type of features, e.g., categorical, numerical, sequential etc. and analyze their role. The sequential data together with the information of previous churners is used to build and train classification

models that can predict the churn probability of customer well in advance. We use several classification algorithm and ensembles and compare the performance of each by using the ROC curves and AUC scores. Our analysis show that we can not only predict whether a customer is prune to churn or not, but also can predict the time of churn well in advance with more than 97% accuracy. The sequential nature of cellular customers data makes this problem a suitable candidate for adaptive and real time learning and prediction.

Chapter 5

Online Learning Algorithms with Boosting under Strong Theoretical Bounds

In this chapter, we investigate online nonlinear learning in a highly adverse and non-stationary environment. We seek to improve our proposed techniques in Chapter 2 and Chapter 3 for online and nonlinear modeling by introducing a boosting approach [61–63]. We specifically introduce online mixture-of-experts boosting algorithms for classification and regression, in a deterministic setting.

Ensemble learning methods are extensively investigated in the machine learning and pattern recognition literatures specifically when a single yet complex model is inadequate. One of the most important ensemble method, boosting, is accomplished by finding a linear combination of weak learning algorithms in order to minimize the total loss over a set of training data commonly using a functional gradient descent [79, 139]. Boosting is successfully applied to several different problems in the machine learning literature including classification [61, 79, 140], regression [139–141], and prediction [142, 143]. However, significantly less attention is given to the idea of boosting in online regression framework. To this end, our goal is (a) to introduce a new boosting approach for online regression, (b) derive several different online regression algorithms based on the boosting approach, (c) provide mathematical guarantees for the performance improvements of our algorithms, and (d) demonstrate the intrinsic connections of boosting with the adaptive mixture-of-experts algorithms [84, 144] and data reuse algorithms [145].

Specifically, the input vectors are received sequentially by M parallel running adaptive filters (weak learners (WL)) [77]. Each WL uses a recursive or stochastic

gradient update method, e.g., least mean squares (LMS) or recursive least squares (RLS), as base learners. The update depends on the target of the applications or problem constraints [4]. After receiving the input vector, each base learner produces its output and calculates the instantaneous error with respect to the target. Generally, the input vector and the corresponding error in estimating the output are used to adjust/update the parameters of the base learner in order to minimize a predefined loss loss function. The loss function is usually some measure of the error such as square error loss or absolute error. The update mechanism is repeated for each of M WLs. The algorithm for individual WLs at each time proceed in rounds from top to bottom, starting from the first WL to the last one to achieve the "boosting" effect [146]. Furthermore, unlike the usual mixture approaches [84, 144], in the online boosting, these adaptations for each individual WL are not independent from each other, i.e., the update of each WL depends on the previous WLs in the mixture. As an example, at each time n, after the m^{th} WL calculates a performance metric based on the estimation error, over $(\boldsymbol{x}[n], y[n])$ pair, and passes it to the $(m+1)^{th}$ WL. This performance metric quantifies the error of the constituent WLs from 1^{st} to m^{th} made on the current $(\boldsymbol{x}[n], y[n])$ pair. In this manner, the performance metric gives a different importance to $(\boldsymbol{x}[n], y[n])$ pair in its adaptation in order to rectify the mistake of the previous WLs.

The proposed idea for online boosting is clearly related to the adaptive mixture-of-experts algorithms widely used in the adaptive signal processing and machine learning literatures, where several parallel running adaptive algorithms are combined to improve the performance. Since each adaptive algorithm has a different view or advantage, the mixture methods provide a diversity and hence result in a performance improvement [144]. This diversity is exploited to yield a final combined algorithm, which achieves a performance better than any of the algorithms in the mixture. Although the online boosting approach is similar to mixture approaches [144], there are significant differences. In the online boosting notion, the parallel running algorithms are not independent, i.e., one deliberately introduces the diversity by updating the WLs one by one from the first WL to the m^{th} WL for each new sample based on the performance of all the previous WLs on this sample. In this sense, each adaptive algorithm, say the $(m+1)^{th}$ WL, receives feedback from the previous WLs, i.e., 1^{st} to m^{th}, and updates its inner parameters accordingly. As an example, if the current $(\boldsymbol{x}[n], y[n])$ is well modeled by the previous WLs, then the $(m+1)^{th}$ WL performs minor update using $(\boldsymbol{x}[n], y[n])$ and may give more emphasis (importance weight) to the later arriving samples that may be worse modeled by the previous WLs. Thus, by boosting, each adaptive algorithm in the mixture can concentrate on different parts of the input and output pairs achieving diversity and significantly improving the gain.

Moreover, we introduce the "random updates" method for boosting, which significantly reduces the computational complexity, while achieving the performance of the mixture methods. This is because in this scenario, the kth filter is updated with probability $\lambda^{(m)}[n]$, which depends on the performance of other filters.

The linear online learning algorithms, such as LMS or RLS, are among the simplest as well as the most widely used learning algorithms in the real-life applications [4]. However, for applications with nonlinear modeling, although linear learners have a low complexity, piecewise linear learning deliver a significantly superior performance [93, 100], with a comparable complexity. These piecewise linear models mitigate the overfitting, stability and convergence issues tied to nonlinear models [4, 8], while effectively improving the modeling power relative to linear filters [93, 100]. Therefore, we implement our boosting algorithms on piecewise linear filters. To this end, we first apply the boosting notion to several parallel running piecewise linear RLS-based filters, and introduce three different approaches to use the importance weights [146], namely "weighted updates","data reuse", and "random updates". In the first approach, we use the performance metrics directly to produce certain weighted LMS algorithms. In the second approach, we use the performance metrics to construct data reuse adaptive algorithms [81]. However, the data reuse in boosting, such as [81], is significantly different from the usual data reusing approaches in adaptive filtering [145]. As an example, in boosting, the performance metric coming from the m^{th} WL determines the data reuse amount in the $(m+1)^{th}$ WL, i.e., it is not used for the m^{th} filter, hence, achieving the diversity. In the third approach we use the performance metrics as a parameter of the Bernoulli distribution to determine whether to update the constituent WLs. This "random update" approach is specifically suitable and effective for big data processing [147] due to the reduced complexity. Then we use a linear combination of the individual outputs of each WL to produce the final output. The combination weights are updated using the RLS algorithm [144]. The boosting idea is then extended to parallel running piecewise linear LMS-based algorithm similar to the RLS case. For this case, we combine the outputs of the constituent filters using a linear filter, which is trained using the LMS algorithm. For all these different cases, we derive the corresponding mean squared error (MSE) results and provide performance bounds in an individual sequence manner [157, 159].

This chapter is organize as follows. We start our discussions by introducing the problem setup and background in Section 5.1, where we provide individual sequence as well as MSE convergence results for the RLS and LMS algorithms. We introduce our first boosted online learning algorithm using the RLS update in Section 5.2. Three different updates are introduced in this section. We then continue with the boosted LMS algorithms in Section 5.3. Then, in Section 5.4 we provide the mathematical analysis for the MSE performance and computational

complexity of the proposed algorithms. We provide results with extensive sets of experiments over the well known benchmark data sets and simulation models widely used in the machine learning and adaptive signal processing literatures to demonstrate the significant gains achieved by the boosting notion. We conclude the chapter with a brief discussion on the achievement of our proposed algorithms for online nonlinear learning in terms of performance gain and computational complexity.

5.1 Problem Description and Background

All vectors are column vectors and represented by bold lower case letters. Matrices are represented by bold upper case letters. For a vector $\boldsymbol{a}$ (or a matrix $\boldsymbol{A}$), $\boldsymbol{a}^T$ (or $\boldsymbol{A}^T$) is the transpose and $\mathrm{Tr}(\boldsymbol{A})$ is the trace of the matrix $\boldsymbol{A}$. Here, $\boldsymbol{I}_k$ and $\boldsymbol{0}_k$ represent the identity matrix of dimension $k \times k$ and the all zeros vector of length k, respectively. We enclose the time index in brackets, i.e., $\boldsymbol{x}[n]$ is the sample at time n. For the simplicity of notation, we work with real data. We denote the mean of a random variable x as $E[x]$. Furthermore, $|S|$ represents the cardinality, i.e., the number of elements in a set S.

We sequentially receive d-dimensional input (regressor) vectors $\{\boldsymbol{x}[n]\}_{n\geq 1}$, $\boldsymbol{x}[n] \in \mathbb{R}^d$, and desired data $\{y[n]\}_{n\geq 1}$, and the estimate of $y[n]$ by $\hat{y}[n] = f_n(\boldsymbol{x}[n])$, where $f_n(.)$ is an online, time varying learning algorithm. At each time n, the estimation error is given by $e[n] = y[n] - \hat{y}[n]$ and is used to update the parameters of the WLs. For presentation purposes, we assume that $y[n] \in [0, 1]$, however, our derivations hold for any bounded but arbitrary desired data sequences, $y[n] \in \mathbb{R}$. In our framework, we do not use any statistical assumptions on the input feature vectors or on the desired data such that our results are guaranteed to hold in an individual sequence manner [155]. However, we also provide steady-state, tracking and transient MSE analysis of the algorithms under widely used statistical models in the signal processing literature [4] for completeness.

Nonlinear models usually suffer from overfitting and the learning process may cause convergence and stability issues [42, 93, 100]. Moreover, the modeling of nonlinear systems for real life applications is restricted due to the higher computational requirements [42, 93, 100]. Therefore, piecewise linear learning is advised as an approximation of the nonlinearity to mitigate the overfitting and stability issues. Piecewise linear modeling achieves a comparable modeling performance to the nonlinear models with much lower complexity [42, 100]. In this chapter, we use piecewise linear modeling as an elegant alternative to linear models. Nevertheless, for illustration, we first explain the basic principles of linear models,

and their extension to the piecewise linear case. Then, in Sections 5.2 and 5.3, we introduce our algorithm in a piecewise linear model framework.

The linear methods are considered as the simplest online modeling or learning algorithms, which estimate the desired data $y[n]$ by a linear model as $\hat{y}[n] = \boldsymbol{w}^T[n]\boldsymbol{x}[n]$, where $\boldsymbol{w}[n]$ is the linear algorithm's coefficients at time n. Note that the previous expression also covers the affine model if one includes a constant term in $\boldsymbol{x}[n]$ such that the new $\boldsymbol{x}[n] \in \mathbb{R}^{d+1}$. Hence, we use the purely linear form for the simplicity of notation. When the true $y[n]$ is revealed, the algorithm updates its coefficients $\boldsymbol{w}[n]$ based on the error $e[n]$. For instance, in the basic implementation of the online least square algorithms, e.g., RLS, the coefficients are selected to minimize the accumulated squared regression error up to the time $n-1$ as

$$\begin{aligned} \boldsymbol{w}[n] &= \arg\min_{\boldsymbol{w}} \sum_{i=1}^{n-1} (y[i] - \boldsymbol{x}^T[i]\boldsymbol{w})^2, \\ &= \left(\sum_{i=1}^{n-1} \boldsymbol{x}[i]\boldsymbol{x}^T[i] \right)^{-1} \left(\sum_{i=1}^{n-1} \boldsymbol{x}[i]y[i] \right), \end{aligned} \tag{5.1}$$

where $\boldsymbol{w}$ is a fixed vector of coefficients. The RLS algorithm enjoys several optimality properties under different statistical settings [4]. Furthermore, apart from these results and more related to the framework of this chapter, the RLS algorithm is also proved to be rate optimal in an individual sequence manner [156]. In [156] (Section V), the authors show that when applied to any sequence $\{\boldsymbol{x}[n]\}_{n\geq 1}$ and $\{y[n]\}_{n\geq 1}$, the accumulated squared error of the RLS algorithm is as small as the accumulated squared error of the best batch least squares (LS) method that is directly optimized for these realizations of the sequences, i.e., for all L as the length of the sequence, $\{\boldsymbol{x}[n]\}_{n\geq 1}$ and $\{y[n]\}_{n\geq 1}$, the RLS algorithm achieves

$$\sum_{l=i}^{L} (y[i] - \boldsymbol{x}^T[i]\boldsymbol{w}[i])^2 - \min_{\boldsymbol{w}} \sum_{i=1}^{L} (y[i] - \boldsymbol{x}^T[i]\boldsymbol{w})^2 \leq O(\ln L). \tag{5.2}$$

The RLS algorithm uses the best performing linear model up to time $n-1$ to predict $y[n]$, hence is a Follow-the-Leader type algorithm [157] (Section 3). Therefore, (5.2) is consistent with the direct application of the online convex optimization results [158] after regularization. The convergence rate (or the rate of the regret) of the RLS algorithm is also shown to be optimal so that $O(\ln L)$ in the upper bound cannot be further improved [159]. It is also shown in [159] that one can reach the optimal upper bound (with exact scaling terms) by using a slightly modified version of (5.1)

$$\boldsymbol{w}[n] = \left(\sum_{i=1}^{n} \boldsymbol{x}[i]\boldsymbol{x}^T[i] \right)^{-1} \left(\sum_{i=1}^{n-1} \boldsymbol{x}[i]y[i] \right). \tag{5.3}$$

This extension, given by (5.3), of (5.1) is a forward algorithm (Section 5 of [160]) and one can show that, in the scalar case, the predictions of (5.3) are always bounded (unlike (5.1)) [159].

We emphasize that in the basic application of the RLS algorithm, all data pairs $(\boldsymbol{x}[i], y[i])$, $i = 1, \ldots, L$, receive equal "importance" or weight in (5.1). Although there exists exponentially weighted or windowed versions of the basic RLS algorithm [4], these methods weigh (or concentrate on) the most recent samples for better modeling of the non-stationarity [4]. However, in the boosting framework [79], each sample pair receives a different weight based on not only the recency, but also the performance of the boosted algorithms on the sample pair under consideration. As an example, if a WL performs worse on a sample, the next WL concentrates more on this example to better rectify this mistake. In the following sections, we use this framework to derive different boosted online learning algorithms.

We use a piecewise linear online learning method, such that the desired signal is predicted as

$$\hat{y}[n] = \sum_{j=1}^{J} s_j[n]\boldsymbol{w}_j^T[n]\boldsymbol{x}[n], \tag{5.4}$$

where $s_j[n]$ is the indicator function of the j^{th} region, i.e.,

$$s_j[n] = \begin{cases} 1 & \text{if } \boldsymbol{x}[n] \in \mathcal{R}_j \\ 0 & \text{if } \boldsymbol{x}[n] \notin \mathcal{R}_j. \end{cases} \tag{5.5}$$

Note that at each time n, only one of the $s_j[n]$'s is nonzero, which indicates the region in which $\boldsymbol{x}[n]$ lies. Thus, if $\boldsymbol{x}[n] \in \mathcal{R}_j$, we update only the j^{th} linear estimator. As an example, consider 2-dimensional input vectors $\boldsymbol{x}[n]$, as depicted in Fig. 5.1. Here, we construct the piecewise linear function f_n such that

$$\begin{aligned} \hat{y}[n] = f_n(\boldsymbol{x}[n]) &= s_1[n]\boldsymbol{w}_1^T[n]\boldsymbol{x}[n] + s_2[n]\boldsymbol{w}_2^T[n]\boldsymbol{x}[n] \\ &= s[n]\boldsymbol{w}_1^T[n]\boldsymbol{x}[n] + (1 - s[n])\boldsymbol{w}_2^T[n]\boldsymbol{x}[n], \end{aligned} \tag{5.6}$$

Then, if $s[n] = 1$ we shall update $\boldsymbol{w}_1[n]$, otherwise we shall update $\boldsymbol{w}_2[n]$, based on the amount of the error, $e[n]$.

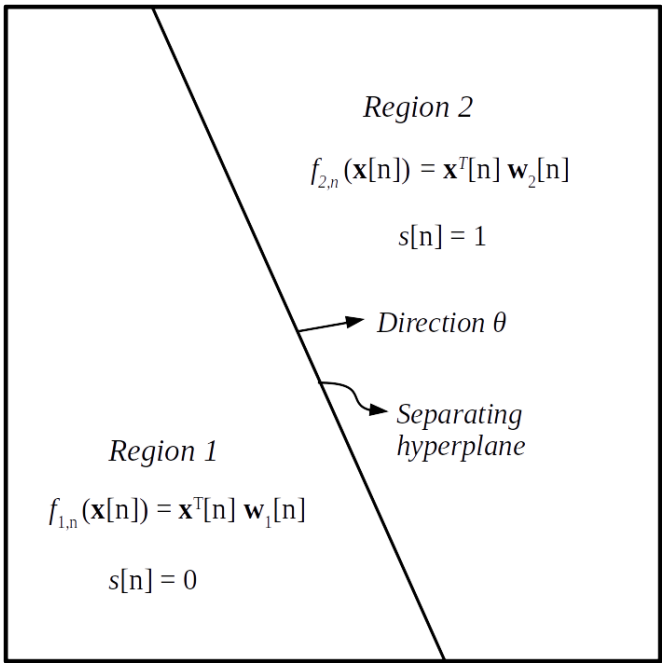


Figure 5.1: A 2-region partitioning of the input vector (i.e., $\boldsymbol{x}[n] \in \mathbb{R}^2$) space. The indicator function $s[n]$ determines the region to which $\boldsymbol{x}[n]$ belongs, i.e., $s[n] = 0$ represents that $\boldsymbol{x}[n]$ is in Region 1 and $s[n] = 1$ represents that $\boldsymbol{x}[n]$ is in Region 2

5.2 Boosted RLS Algorithms

As shown in Fig. 5.2, at each iteration n, we have M parallel running WLs with estimating functions $f_n^{(m)}$, producing estimates $\hat{y}^{(k)}[n] = f_n^{(m)}(\boldsymbol{x}[n])$ of $y[n]$, $m = 1, \ldots, M$. As an example, if we use M "linear" learners, $\hat{y}^{(m)}[n] = \boldsymbol{x}^T[n]\boldsymbol{w}^{(m)}[n]$ is the estimate generated by the m^{th} constituent WL, and if we use piecewise linear learners (each of which with J different regions), then $\hat{y}^{(m)}[n] = \sum_{j=1}^{J} s_j[n]\boldsymbol{x}^T[n]\boldsymbol{w}_j[n]$. The outputs of these M WLs are then combined using the linear weights $\boldsymbol{z}[n]$ to produce the final estimate as $\hat{y}[n] = \boldsymbol{z}^T[n]\boldsymbol{y}[n]$ [144], where $\boldsymbol{y}[n] \triangleq [\hat{y}^{(1)}[n], \ldots, \hat{y}^{(M)}[n]]^T$ is the vector containing the outputs of each WL as its elements. Once the desired signal $y[n]$ is available, the M parallel running WLs will be updated for the next iteration. Moreover, the linear combination coefficients $\boldsymbol{z}[n]$ are also updated using ordinary RLS method, as detailed later in Section 5.2.4.

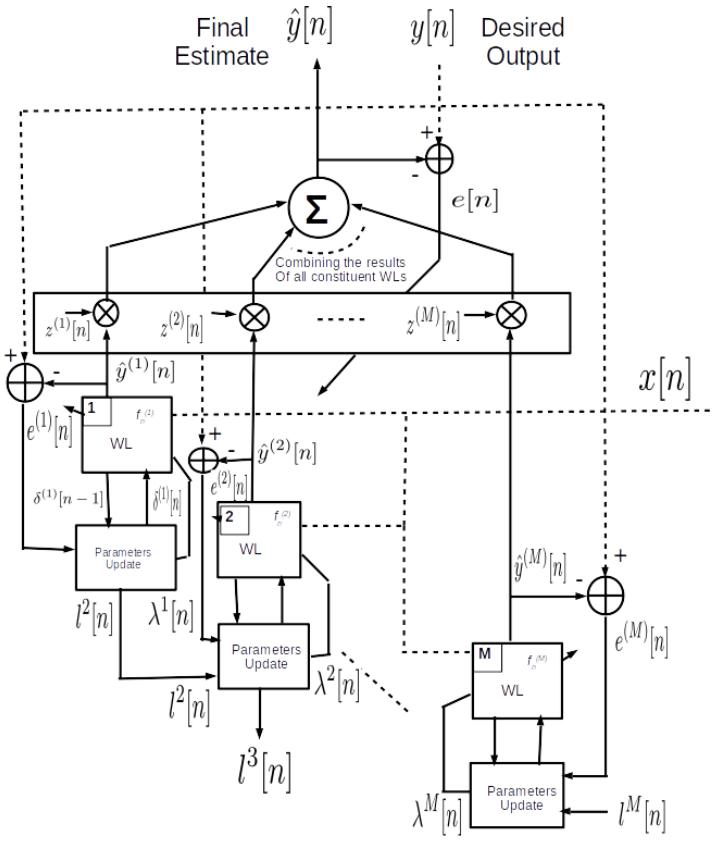


Figure 5.2: The block diagram of online boosting algorithm that combines M WLs. Each WL is a piecewise linear model that produces an estimate of the output $y[n]$ for the input vector $\boldsymbol{x}[n]$. The final estimate $\hat{y}[n]$ is generated by linear combinations of the constituent WLs. With each iteration, the parameters of WLs as well as the combination weights are updated based on $\lambda^{(m)}[n]$ and $e^{(m)}[n]$

After $y[n]$ is revealed, the constituent WLs, $f_n^{(m)}$, $m = 1, \ldots, M$, are consecutively updated as shown in Fig. 5.2 from top to bottom, i.e., first $m = 1$ is updated, then, $m = 2$ and finally $m = M$ is updated. However, to enhance the performance, we use a boosted updating approach [79], such that, the $(m+1)^{th}$ WL receives a "total loss" parameter, $l^{(m+1)}[n]$, from the WL $f_n^{(m)}$, as (Refer to Fig. 5.3)

$$l^{(m+1)}[n] = l^{(m)}[n] + \left[\sigma^2 - \left(y[n] - f_n^{(m)}(\boldsymbol{x}[n])\right)^2\right], \tag{5.7}$$

to compute a weight $\lambda^{(m)}[n]$. The total loss parameter $l^{(m)}[n]$, indicates the sum of the differences between the desired MSE (σ^2) and the squared error of the first $m-1$ WLs at time n, as shown in Fig. 5.3. Then, the difference $\sigma^2 - (e^{(m)}[n])^2$ is added to $l^{(m)}[n]$, to generate $l^{(m+1)}[n]$, and $l^{(m+1)}[n]$ is passed to the next constituent WL as shown in Fig. 5.2. Here, $\left[\sigma^2 - \left(y[n] - f_n^{(m)}(\boldsymbol{x}[n])\right)^2\right]$

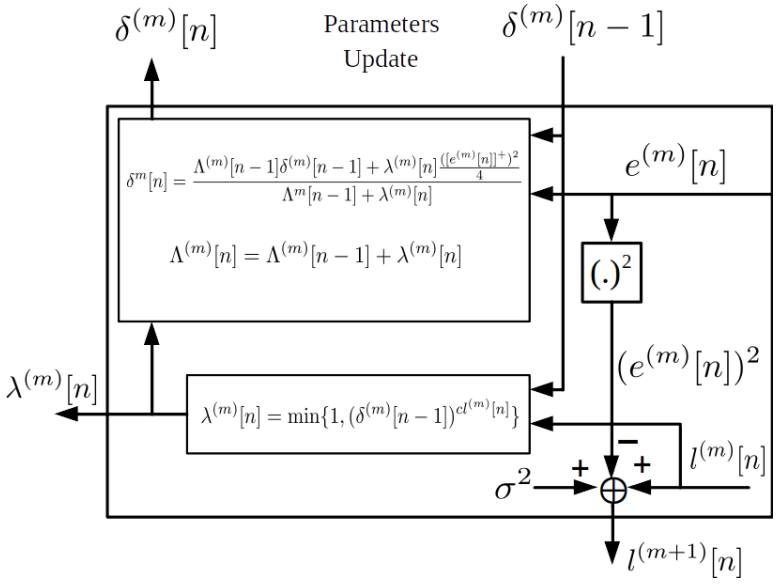


Figure 5.3: Parameters update block of the mth constituent WL, which is embedded in the mth WL block as depicted in Fig. 5.2. This block receives the parameter $l^{(m)}[n]$ provided by the $(m-1)$th WL, and uses that in computing $\lambda^{(m)}[n]$. It also computes $l^{(m+1)}[n]$ and passes it to the $(m+1)^{th}$ WL. The parameter $[e^{(m)}[n]]^+$ represents the error of the thresholded estimate as explained in (5.9), and $\Lambda^{(m)}[n]$ shows the sum of the weights $\lambda^{(m)}[1], \ldots, \lambda^{(m)}[n]$. The WMSE parameter $\delta^{(m)}[n-1]$ represents the time averaged weighted square error made by the m^{th} WL up to time $n-1$.
measures the deviation of m^{th} WL from the target MSE. As an example, for $y[n] = f(\boldsymbol{x}[n]) + \nu[n]$ where $\nu[n]$ is the observation noise and $f(\cdot)$ is a nonlinear function, then the variance of the noise process $\nu[n]$ can be selected as the target MSE σ^2. Consequently, $l^{(m)}[n]$ can be seen as a performance metric of the m^{th} WLs on the $(y[n], \boldsymbol{x}[n])$ pair with respect to the target performance.

We next update the m^{th} WL by either of the "weighted updates", "data reuse", or "random updates" methods using the weight $\lambda^{(m)}[n]$. We design our algorithm such that for large errors made by $m-1$ WLs, the $\lambda^{(m)}[n]$ would become large, and hence the m^{th} WL would prioritize and focus more on the current input pair $(y[n], \boldsymbol{x}[n])$ to boost the performance of the overall system. We next demonstrate the construction of these performance weights $\lambda^{(m)}[n]$ where $0 < \lambda^{(m)}[n] \leq 1$. We start by initializing the weights for 1^{st} WL as $\lambda^{(1)}[n] = 1$, for all n, and determine the remaining weights as in [146, 149]. We define the weights as

$$\lambda^{(m)}[n] = \min\left\{1, \left(\delta^{(m)}[n-1]\right)^{c\, l^{(m)}[n]}\right\}, \tag{5.8}$$

where $\delta^{(m)}[n-1]$ is the estimation MSE of the m^{th} WL on the pair $\{\boldsymbol{x}[n], \{y[n]\}_{n\geq 1}$ while $c \geq 0$ is the "dependence" factor of each WL update on the performance of

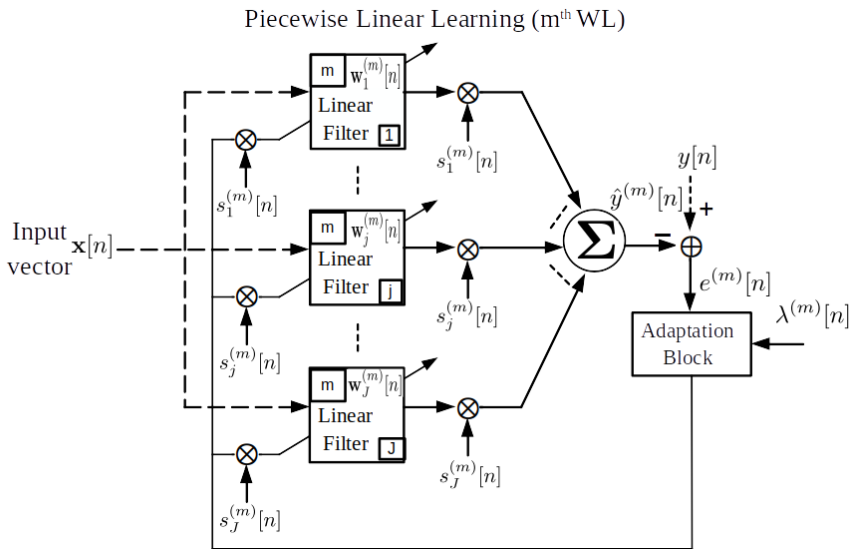


Figure 5.4: A sample piecewise linear adaptive filter, used as the m^{th} constituent WL in the system depicted in Fig. 5.2. This WL consists of J linear filters, one of which produces the estimate at each iteration n. Based on where the input vector at time n, $\boldsymbol{x}[n]$, lies in the input vector space, one of the $s_j^{(m)}[n]$'s is 1 and all others are 0. Hence, at each iteration only one of the linear filters is used for estimation and updated correspondingly.

previous WLs, i.e., $c = 0$ means all WLs work independently of each other [144]. In the conventional online boosting algorithms [146, 149], $\left(1 - \delta^{(m)}[n-1]\right)$ is designed to achieve better performance of the WLs [149], however, the performance gain is targeted to be the same for each WL. In contrast, we seek the algorithm to be thoroughly adaptive abstentious of using any a priori information. Therefore, we choose $\delta^{(m)}[n-1]$ as the weighted and thresholded MSE of the m^{th} WL up to time $n-1$ as

$$\begin{aligned}\delta^{(m)}[n] &= \frac{\sum_{\eta=1}^{n} \frac{\lambda^{(m)}[\eta-1]}{4} \left(y[\eta] - \left[f_\eta^{(m)}(\boldsymbol{x}[\eta])\right]^+\right)^2}{\sum_{\eta=1}^{n} \lambda^{(m)}[\eta]} \\ &= \frac{\Lambda^{(m)}[n-1]\delta^{(m)}[n-1] + \frac{\lambda^{(m)}[n]}{4} \left(y[n] - \left[f_n^{(m)}(\boldsymbol{x}[n])\right]^+\right)^2}{\Lambda^{(m)}[n-1] + \lambda^{(m)}[n]} \end{aligned} \tag{5.9}$$

where $\Lambda^{(m)}[n] \triangleq \sum_{\eta=1}^{n} \lambda^{(m)}[\eta]$, and $\left[f_\eta^{(m)}(\boldsymbol{x}[\eta])\right]^+$ thresholds $f_\eta^{(m)}(\boldsymbol{x}[\eta])$ into the range $[0, 1]$. This thresholding is necessary to assure that $0 < \delta^{(m)}[n] \leq 1$, which guarantees $0 < \lambda^{(m)}[n] \leq 1$ for all $m = 1, \ldots, M$ and n. We point out that (5.9) can be recursively calculated as in Fig. 5.3.

According to the definition of $\delta^{(m)}[n]$ and $\lambda^{(m)}[n]$, if the m^{th} WL is performing

"well", i.e., if $\delta^{(m)}[n]$ is small enough, the algorithm passes less weight to the next WLs, such that those WLs can concentrate more on the other samples. Hence, the WLs can focus on different parts of the data and increase the diversity [144]. Furthermore, following this idea, in (5.8), when the most of the WLs are performing poorly and producing large errors, the weights $\lambda^{(m)}[n]$ are larger and close to one. On the other hand, if the pair $(y[n], \boldsymbol{x}[n])$ is correctly modeled by the previous WLs, i.e., the WLs $m+1, \ldots, M$, the weights $\lambda^{(m)}[n]$ shrink. We next demonstrate three approaches to update the combinations weights and the constituent WLs based on these parameters. Again, here the constituent WLs consist of piecewise linear models as discussed earlier in Section 5.1, and individually updated using RLS algorithm. The three approaches are (1) directly using λs as the performance metrics, i.e., the dependence factor on the previous WLs, (2) data reuse approach where the constituent WLs are updated a number of times depending on λ, and (3) the random update where we choose to update or leave a certain WL based on a probability distribution that is dependent on λ.

5.2.1 Directly Using λ's as Sample Weights

As depicted in Fig. 5.4, each constituent WL uses a piecewise linear model that in turn consists of J linear filters. At each time n, all of the constituent WLs (shown in Fig. 5.2) estimate the desired output $y[n]$ in parallel, and the final estimate is a linear combination of the results generated by the constituent WLs. However, at each time n, exactly one of the J linear filters in each constituent WL is used for estimating $y[n]$. Correspondingly, updating the constituent WLs, we follow a greedy approach by updating only the filter (in the piecewise model) that has been used for the estimation. To this end, we use $s_j^{(m)}[n]$ to denote the indicator function for the j^{th} linear filter embedded in the mth constituent WL, as has been explained in Section 5.1. Therefore, at each time n, only a single filter whose indicator function equals 1, will be updated. Furthermore, as soon as the mth constituent WL receives the weight $\lambda^{(m)}[n]$, it updates the linear coefficients $\boldsymbol{w}_j^{(m)}[n]$, assuming that $\boldsymbol{x}[n]$ lies in the j^{th} region of the m^{th} constituent WL, i.e., piecewise model. We consider $\lambda^{(m)}[n]$, representing the error in estimating the output, as the weight for the observation pair $(y[n], \boldsymbol{x}[n])$ and apply a weighted RLS update on $\boldsymbol{w}_j^{(m)}[n]$. For the weighted RLS algorithm, we define the autocorrelation matrix and the cross correlation vector as

$$\boldsymbol{R}_j^{(m)}[n+1] \triangleq \alpha \boldsymbol{R}_j^{(m)}[n] + \lambda^{(m)}[n]\boldsymbol{x}[n]\boldsymbol{x}^T[n], \tag{5.10}$$

$$\boldsymbol{p}_j^{(m)}[n+1] \triangleq \alpha \boldsymbol{p}_j^{(m)}[n] + \lambda^{(m)}[n]\boldsymbol{x}[n]y[n], \tag{5.11}$$

where α is the forgetting factor [4] and $\boldsymbol{w}_j^{(m)}[n+1] = \left(\boldsymbol{R}_j^{(m)}[n+1]\right)^{-1} \boldsymbol{p}_j^{(m)}[n+1]$ can be calculated in a recursive manner as

$$
\begin{aligned}
& e^{(m)}[n] = y[n] - \boldsymbol{x}^T[n]\boldsymbol{w}_j^{(m)}[n], \\
& \boldsymbol{g}_j^{(m)}[n] = \frac{\lambda^{(m)}[n]\boldsymbol{P}_j^{(m)}[n]\boldsymbol{x}[n]}{\alpha + \lambda^{(m)}[n]\boldsymbol{x}^T[n]\boldsymbol{P}_j^{(m)}[n]\boldsymbol{x}[n]}, \\
& \boldsymbol{w}_j^{(m)}[n+1] = \boldsymbol{w}_j^{(m)}[n] + e^{(m)}[n]\boldsymbol{g}_j^{(m)}[n], \\
& \boldsymbol{P}_j^{(m)}[n+1] = \alpha^{-1}\left(\boldsymbol{P}_j^{(m)}[n] - \boldsymbol{g}_j^{(m)}[n]\boldsymbol{x}^T[n]\boldsymbol{P}_j^{(m)}[n]\right).
\end{aligned} \tag{5.12}
$$

Here, $\boldsymbol{P}_j^{(m)}[n] \triangleq \left(\boldsymbol{R}_j^{(m)}[n]\right)^{-1}$, and $\boldsymbol{P}_j^{(m)}[0] = v^{-1}\boldsymbol{I}$ for $j = 1, \ldots, J$, and $0 < v \ll 1$.

Remark 1: We emphasize that one can be inclined to use a single boosted RLS algorithm instead of running M RLS algorithms in parallel due to the computational complexity considerations. In the single RLS implementation, the calculated importance weight can be used as a weight in the next time instant $n+1$ so that a single RLS algorithm may concentrate more on wrongly regressed samples. However, running M RLS based WLs with boosting provides "diversity" [144], where each constituent algorithm concentrates on different parts of the data according to the boosting weights in (5.8). Hence, by this boosting and then final mixture-of-experts combination [144], we achieve a significantly improved learning performance.

5.2.2 Data Reuse Approaches Based on the Weights

Another approach follows Ozaboost [81]. In this approach, from $\lambda^{(m)}[n]$, we generate an integer, say $\mathbb{c}_n^{(m)} = \text{ceil}(K\lambda^{(m)}[n])$, and apply the RLS update on the $(y[n], \boldsymbol{x}[n])$ pair repeatedly $\mathbb{c}_n^{(m)}$ times, i.e., run the RLS update on the same $(y[n], \boldsymbol{x}[n])$ pair $\mathbb{c}_n^{(m)}$ times consecutively. Here, K is an integer, as an example, we use $K = 2$ in our simulations. The final $\boldsymbol{w}^{(m)}[n]$ is calculated after $\mathbb{c}_n^{(m)}$ RLS updates. As a major advantage, clearly, this reusing approach can be readily generalized to other adaptive algorithms in a straightforward manner.

Remark 2: We emphasize that the data reuse approach is exactly equal to the approach in Section 5.2.1, where the cross correlation vector and auto correlation

matrix are defined as

$$\boldsymbol{R}^{(m)}[n+1] = \boldsymbol{R}^{(m)}[n] + \mathbb{c}_n^{(m)} \boldsymbol{x}[n]\boldsymbol{x}^T[n],$$
$$\boldsymbol{p}^{(m)}[n+1] = \boldsymbol{p}^{(m)}[n] + \mathbb{c}_n^{(m)} \boldsymbol{x}[n]y[n].$$

Hence, for the RLS recursion, these two approaches are equivalent up to ceil() operation to generate an integer from $\lambda^{(m+1)}[n]$. However, such an equivalence is not usually correct for other type of updates such as the LMS algorithm, hence we also provide the online boosting algorithm with LMS based WLs for completeness.

Unlike the Ozaboost [81] data reuse strategy, where $\lambda^{(m)}[n]$ is used a parameter of a Poisson distribution to generate the integer $\mathbb{c}_n^{(m)}$, our data reuse approach generates $\mathbb{c}_n^{(m)}$ as $\mathbb{c}_n^{(m)} = \text{ceil}(K\lambda^{(m)}[n])$.

5.2.3 Random Updates Approach Based on The Performance Metrics

In this approach, we simply use the performance metric $\lambda^{(m)}[n]$ as a probability of updating the m^{th} WL at time n. To this end, we generate a Bernoulli random variable, which is 1 with probability $\lambda^{(m)}[n]$ and is 0 with probability $1-\lambda^{(m)}[n]$. Then, in each of the constituent WLs, we update one of the linear filters that is involved in estimation process, only if the Bernoulli random variable equals 1. With this method, we significantly reduce the computational complexity of the algorithm. Moreover, due to the dependence of this Bernoulli random variable on the MSE performance of the previous constituent WLs, this method does not degrade the MSE performance severely, while offering a considerably lower complexity, i.e., when the MSE is low, there is no need for further updates, hence, the probability of an update is low, while this probability is larger when the MSE is high. In other words, if the complexity of the underlying model, i.e., the relationship between the input and output, varies with time, the proposed learning algorithm with random updates adapts accordingly to use optimal computational resources.

5.2.4 The Final Algorithm

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

yielding [144]

$$\begin{aligned}\boldsymbol{R}[n+2] &= \lambda \boldsymbol{R}[n] + \boldsymbol{y}[n]\boldsymbol{y}^T[n],\\ \boldsymbol{p}[n+1] &= \lambda \boldsymbol{p}[n] + \boldsymbol{y}[n]y[n],\end{aligned}$$

and

$$\begin{aligned}e[n] &= y[n] - \boldsymbol{z}^T[n]\boldsymbol{y}[n],\\ \boldsymbol{g}[n] &= \frac{\boldsymbol{P}[n]\boldsymbol{y}[n]}{\lambda + \boldsymbol{y}^T[n]\boldsymbol{P}[n]\boldsymbol{y}[n]},\\ \boldsymbol{z}[n+1] &= \boldsymbol{z}[n] + e[n]\boldsymbol{g}[n],\\ \boldsymbol{P}[n+1] &= \lambda^{-1}\boldsymbol{P}[n] - \lambda^{-1}\boldsymbol{g}[n]\boldsymbol{y}^T[n]\boldsymbol{P}[n],\end{aligned} \tag{5.13}$$

where $0 < \lambda \leq 1$ is the exponential weighting. The complete algorithm is given in Algorithm 4 with the weighted RLS implementation in (5.12).

5.3 Boosted LMS Algorithms

In this case, as shown in Fig. 5.2, we have M parallel running piecewise linear WLs, each of which is updated using the LMS algorithm with a different learning rate. For instance, if the input vector $\boldsymbol{x}[n]$ lies in the j^{th} region of the m^{th} WL partition, $s_j^{(m)}[n] = 1$, we use $\boldsymbol{w}_j^{(m)}[n]$ to estimate $y[n]$, and update this linear filter with its own learning rate $\mu_j^{(m)}$. The learning rates are based on the performance metrics given in (5.8) that in turn depend upon the total loss and the MSE parameters given in the equations (5.7) and (5.9), respectively. We next introduce three LMS based online boosting algorithms, similar to those introduced in Section 5.2.

5.3.1 Directly Using λ's to Scale the Learning Rates

We note that since $0 < \lambda^{(m)}[n] \leq 1$ according to (5.8), we can directly use these performance metrics to scale the learning rates for the LMS updates. When the m^{th} WL receives the performance metric $\lambda^{(m)}[n]$, it updates its filter coefficients $\boldsymbol{w}_j^{(m)}[n], j = 1, \ldots, J$, as

$$\boldsymbol{w}_j^{(m)}[n+1] = \left(\boldsymbol{I} - \rho_j^{(m)}\lambda^{(m)}[n]\boldsymbol{x}[n]\boldsymbol{x}^T[n]\right)\boldsymbol{w}_j^{(m)}[n] + \rho_j^{(m)}\lambda^{(m)}[n]\boldsymbol{x}[n]y[n], \tag{5.14}$$

where $0 < \rho_j^{(m)}\lambda^{(m)}[n] \leq \rho_j^{(m)}$.

Algorithm 4: Boosted RLS with the weighting scheme in (5.12)

Input: $(\boldsymbol{x}[n], y[n])$ (data stream), M (number of RLS piecewise linear constituent WLs running in parallel) and σ^2 (the desired MSE, upper bound on the error variance).
Initialize the regression coefficients $\boldsymbol{w}_j^{(m)}[1]$ for each RLS filter; and the combination coefficients as $\boldsymbol{z}[1] = \frac{1}{M}[1, 1, \ldots, 1]^T$; and for all m set $\delta^{(m)}[0] = 0$.
for $n = 1$ **to** L **do**
  Receive the input data instance $\boldsymbol{x}[n]$;
  Compute the indicator functions $s_j^{(m)}[n]$ for all m's
  Compute the constituent WL outputs $\hat{y}^{(m)}[n] = \sum_{j=1}^{J} s_j^{(m)}[n] \boldsymbol{x}^T[n] \boldsymbol{w}_i^{(m)}[n]$;
  Produce the final estimate $\hat{y}[n] = \boldsymbol{z}^T[n][\hat{y}^{(1)}[n], \ldots, \hat{y}^{(M)}[n]]^T$;
  Receive the true output $y[n]$ (desired data);
  $\lambda^{(1)}[n] = 1$; $l^{(1)}[n] = 0$;
  for $m = 1$ **to** M **do**
    Update the regression coefficients $\boldsymbol{w}_j^{(m)}[n]$ by using RLS and the performance metrics $\lambda^{(m)}[n]$ based on one of the introduced algorithms in Section 5.2;
    $e^{(m)}[n] = y[n] - \hat{y}^{(m)}[n]$;
    $\lambda^{(m)}[n] = \min\left\{1, \left(\delta^{(m)}[n-1]\right)^{c\, l^{(m)}[n]}\right\}$;
    $\delta^{(m)}[n] = \frac{\Lambda^{(k)}[n-1]\delta^{(k)}[n-1] + \frac{\lambda^{(m)}[n]}{4}\left(y[n] - \left[f_n^{(m)}(\boldsymbol{x}[n])\right]^+\right)^2}{\Lambda^{(m)}[n-1] + \lambda^{(m)}[n]}$;
    $\Lambda^{(m)}[n] = \Lambda^{(m)}[n-1] + \lambda^{(m)}[n]$
    $l^{(m+1)}[n] = l^{(m)}[n] + \left[\sigma^2 - \left(e^{(m)}[n]\right)^2\right]$;
  end for
  $e[n] = y[n] - \boldsymbol{z}^T[n]\boldsymbol{y}[n]$;
  $\boldsymbol{g}[n] = \frac{\boldsymbol{P}_{[n]}\boldsymbol{y}_{[n]}}{\lambda + \boldsymbol{y}^T{}_{[n]}\boldsymbol{P}_{[n]}\boldsymbol{y}_{[n]}}$;
  $\boldsymbol{z}[n+1] = \boldsymbol{z}[n] + e[n]\boldsymbol{g}[n]$;
  $\boldsymbol{P}[n+1] = \lambda^{-1}\boldsymbol{P}[n] - \lambda^{-1}\boldsymbol{g}[n]\boldsymbol{y}^T[n]\boldsymbol{P}[n]$;
end for

Note that we can choose $\rho_j^{(m)} = \rho_j$ for all m, since the adaptive algorithms work consecutively from top to bottom, and the j^{th} linear filters of different constituent WLs will have different learning rates $\rho_j \lambda^{(m)}[n]$.

5.3.2 A Data Reuse Approach Based on the Performance Metric

In this scenario, for updating $\boldsymbol{w}_j^{(m)}[n]$, we use the LMS update $\mathbb{c}_n^{(m)} = \text{ceil}(K\lambda^{(m)}[n])$ times to obtain the $\boldsymbol{w}_j^{(m)}[n+1]$ as

$$\begin{aligned}
&\boldsymbol{r}^{(0)} = \boldsymbol{w}_j^{(m)}[n], \\
&\boldsymbol{r}^{(b)} = \left(\boldsymbol{I} - \rho_j^{(m)}\boldsymbol{x}[n]\boldsymbol{x}^T[n]\right)\boldsymbol{r}^{(b-1)} + \rho_j^{(m)}\boldsymbol{x}[n]y[n],\ b = 1, \ldots, \mathbb{c}_n^{(m)}, \\
&\boldsymbol{w}^{(m)}[n+1] = \boldsymbol{r}^{\left(\mathbb{c}_n^{(m)}\right)}.
\end{aligned} \tag{5.15}$$

Here, $\rho_j^{(m)} > 0$ is the step size (learning rate) of LMS update for the m^{th} WL.

5.3.3 Random Updates Based on the Weights

Again, in this scenario, similar to the RLS case, we use the metric $\lambda^{(m)}[n]$ to generate a random number from a Bernoulli distribution, which equals 1 with probability $\lambda^{(m)}[n]$ and equals zero with probability $1 - \lambda^{(m)}[n]$. Then, if this number is 1, we perform the ordinary LMS update on $\boldsymbol{w}_j^{(m)}[n]$, otherwise we skip to update the parameters of that certain WL.

5.3.4 The Final Boosting Algorithm with LMS update

Once the desired data $y[n]$ is available, we update the constituent WLs as well as the combination weights $\boldsymbol{z}[n]$. To update the combination weights, we again employ an LMS algorithm yielding

$$\boldsymbol{z}[n+1] = \left(\boldsymbol{I} - \rho\boldsymbol{y}[n]\boldsymbol{y}^T[n]\right)\boldsymbol{z}[n] + \rho\boldsymbol{y}[n]y[n], \tag{5.16}$$

where $\rho > 0$ is the learning rate and $\boldsymbol{y}[n] = [\hat{y}^{(1)}[n], \ldots, \hat{y}^{(M)}[n]^T$.

The complete final algorithm is similar to Algorithm 4, but replacing the weighted RLS with the weighted LMS implementation in (5.14), i.e., for updating the constituent WLs (line 11 in Algorithm 4), we use the LMS recursion given in (5.14) (for weighted updates), and for updating the combination weights $\boldsymbol{z}[n]$, we use the LMS update given in (5.16).

Remark 3: Similar updates can be employed for all different types of online

learning methods.

Remark 4: To this end, we assumed that each constituent WL is a piecewise model built upon a fixed partition, i.e., the partitioning of the input space is not updated during the algorithm. However, one can use a method similar to that in [93, 100] to make the partitioning adaptive. As an example, let each constituent WL is defined on a 2-region partition, as shown in Fig. 5.1, the regions of which are separated using a hyper-plane $\boldsymbol{\theta}^{(m)}[n]$ that itself is varying with time. Then we can update $\boldsymbol{\theta}^{(m)}[n]$ based on the instantaneous error. Then the indicator function to decide the region in which $\boldsymbol{x}[n]$ lies, is defined as follows

$$s^{(m)}[n] = \frac{1}{1+\exp(-\boldsymbol{\theta}^T[n]\boldsymbol{x}[n])}, \tag{5.17}$$

and the estimate made by the m^{th} WL is given by

$$\hat{y}^{(m)}[n] = s^{(m)}[n]\hat{y}_1^{(m)}[n] + \left(1 - s^{(m)}[n]\right)\hat{y}_2^{(m)}[n], \tag{5.18}$$

which yields the following ordinary LMS update for $\boldsymbol{\theta}^{(m)}[n]$ [100]

$$\boldsymbol{\theta}^{(m)}[n+1] = \boldsymbol{\theta}^{(m)}[n] + \rho_{\boldsymbol{\theta}} e^{(m)}[n]\left(\hat{y}_1^{(m)}[n] - \hat{y}_2^{(m)}[n]\right) s^{(m)}[n]\left(1 - s^{(m)}[n]\right)\boldsymbol{x}[n]. \tag{5.19}$$

Then, in the "random updates" case, we update the partitioning hyperplane with probability $\lambda^{(m)}[n]$. For the "weighted updates" case, we update $\boldsymbol{\theta}^{(m)}[n]$ as follows,

$$\boldsymbol{\theta}^{(m)}[n+1] = \boldsymbol{\theta}^{(m)}[n] + \rho_{\boldsymbol{\theta}}\lambda^{(m)}[n]e^{(m)}[n]\left(\hat{y}_2^{(m)}[n] - \hat{y}_1^{(m)}[n]\right) s^{(m)}[n]\left(1 - s^{(m)}[n]\right)\boldsymbol{x}[n]. \tag{5.20}$$

For the "data reuse" case, we perform this update $\mathbb{c}_n^{(m)} = \text{ceil}(K\lambda^{(m)}[n])$ times, along with updating the rest of the parameters, that is calculated as

$$\begin{aligned}
\kappa^{(b+1)} &= \kappa^{(b)} + \rho_{\boldsymbol{\theta}}\varepsilon^{(b)}\boldsymbol{x}[n]\boldsymbol{x}^T[n]\left(\boldsymbol{r}_1^{(b)} - \boldsymbol{r}_2^{(b)}\right)\phi^{(b)}\left(1-\phi^{(b)}\right), \\
\boldsymbol{r}_1^{(b+1)} &= \boldsymbol{r}_1^{(b)} + \rho_j^{(m)}\phi^{(b)}\varepsilon^{(b)}\boldsymbol{x}[n], \\
\boldsymbol{r}_2^{(b+1)} &= \boldsymbol{r}_2^{(b)} + \rho_j^{(m)}(1-\phi^{(b)})\varepsilon^{(b)}\boldsymbol{x}[n], \\
\phi^{(b+1)} &= \frac{1}{1+\exp(-\boldsymbol{\vartheta}^T[n]\boldsymbol{x}[n])}, \\
\varepsilon^{(b+1)} &= y[n] - \left(\phi^{(b+1)}\boldsymbol{r}_1^{(b+1)} + (1-\phi^{(b+1)})\boldsymbol{r}_2^{(b+1)}\right)\boldsymbol{x}[n],
\end{aligned} \tag{5.21}$$

where $b = 0, \ldots, (\mathbb{c}_n^{(m)} - 1)$, $\boldsymbol{\vartheta}^{(0)} = \boldsymbol{\theta}^{(m)}[n]$, $\varepsilon^{(0)} = e^{(m)}[n]$, $\phi^{(0)} = s^{(m)}[n]$, and $\boldsymbol{r}_j^{(0)} = \boldsymbol{w}_j^{(m)}[n]$ for $j = 1, 2$. Also, the updated values are $\boldsymbol{\theta}^{(m)}[n+1] = \boldsymbol{\vartheta}^{(\mathbb{c}_n^{(m)})}$, and $\boldsymbol{w}_j^{(m)}[n+1] = \boldsymbol{r}_j^{(\mathbb{c}_n^{(m)})}$ for $j = 1, 2$.

5.4 Analysis Of The Proposed Algorithms

In this section we provide the MSE as well as complexity analysis for the proposed algorithms. We show an upper bound for the performance metric $\lambda^{(m)}[n]$, which is significantly less than 1. This bound proves that the complexity of the "random updates" algorithm is significantly less than other proposed algorithms, and slightly greater than that of a single piecewise linear model. Hence, it shows the considerable advantage of "boosting with random updates" in processing high dimensional data. We also use this bound in the MSE analysis of the algorithms. In addition, for the sake of simplicity, we have chosen the "dependence parameter" $c = 1$. Nevertheless, the results are readily extendable to the general case.

5.4.1 Complexity Analysis

Here, we compare the complexity of the proposed algorithms by providing an upper bound for the random updates case (introduced in Section 5.2.3 for RLS, and in Section 5.3.3 for LMS updates). We show that the computational complexity is significantly reduced as compared to the two other approaches. As a reference, we define computational complexity as the number of operations on real numbers per input sample. As an example, for input $\boldsymbol{x}[n] \in \mathbb{R}^d$, each constituent WL performs $O(d)$ computations to generates its estimate. Furthermore, the algorithm requires $O(d^2)$ computations when updated using RLS algorithm due to updating the matrix $\mathbf{R}_j^{(m)}[n]$, while it needs $O(d)$ computations when updated using the LMS algorithm (in their most basic implementations).

In the following, we first derive the computational complexity of using the RLS updates in different boosting scenarios. In the first case, where all M WLs are updated, the computational cost is $O(Md^2)$ per sample at n. However, in "random updates", the mth WL may or may not be updated with probabilities $\lambda^{(m)}[n]$ and $1 - \lambda^{(m)}[n]$ respectively, yielding

$$C_n^{(m)} = \begin{cases} O(d^2) & \text{with probability } \lambda^{(m)}[n] \\ O(d) & \text{with probability } 1 - \lambda^{(m)}[n], \end{cases} \tag{5.22}$$

where $C_n^{(m)}$ indicates the complexity of running the m^{th} WL at sample n. Therefore, the total computational complexity C_n is given by $C_n = \sum_{m=1}^{M} C_n^{(m)}$, which yields

$$E\left[C_n\right] = E\left[\sum_{m=1}^{M} C_n^{(m)}\right] = \sum_{m=1}^{M} E[\lambda^{(m)}[n]]O(d^2). \tag{5.23}$$

Hence, if $E\big[\lambda^{(m)}[n]\big]$ is upper bounded as $\tilde{\lambda^{(m)}} < 1$, the average computational complexity of the random updates method is,

$$E\left[C_n\right] < \sum_{m=1}^{M} \tilde{\lambda}^{(m)} O(d^2). \tag{5.24}$$

In Theorem 1, we provide sufficient constraints to have such an upper bound.

Furthermore, we can use such a bound for the "data reuse" mode as well. In this case, for each WL $f_n^{(m)}$, we perform the RLS update $K\lambda^{(m)}[n]$ times, resulting a computational complexity of order $E\left[C_n\right] < \sum_{m=1}^{M} K\ E[\tilde{\lambda}^{(m)}](O(d^2))$. Similarly, for the LMS updates, we obtain the computational complexities as $O(md)$, $O\big(\tilde{\lambda}^{(m)}d\big)$, and $O\big(K\tilde{\lambda}^{(m)}d\big)$, for the "weighted samples", "random updates", and "data reuse" modes respectively.

The following theorem determines the upper bound $\tilde{\lambda}^{(m)}$ for $E\big[\lambda^{(m)}\big]$.
Theorem: *If the learning algorithm converges and achieves a sufficiently small MSE (according to the proof following this Theorem), the following upper bound is obtained for $\lambda^{(m)}[n]$, given that σ^2 is chosen properly,*

$$E\left[\lambda^{(m)}[n]\right] \leq \tilde{\lambda}^{(m)} = \left(\beta^{-2\sigma^2}(1+2\psi^2\ln\beta)\right)^{\frac{1-m}{2}}, \tag{5.25}$$

where $\beta \triangleq E\left[\delta^{(m)}[n-1]\right]$ and $\psi^2 \triangleq E\left[\left(e^{(m)}[n]\right)^2\right]$.
It can be shown in a straightforward manner that, this bound is less than 1 for appropriate choices of σ^2, and reasonable values for the MSE according to the proof. This theorem directly implies that if we choose σ^2 such that it is achievable, i.e., the learning algorithm can achieve a slightly lower MSE than σ^2, the probability of updating the WLs in the random updates scenario will decrease. This is of course our desired results, since if the WLs are performing sufficiently well, there is no need for additional updates. Moreover, if σ^2 is chosen such that the WLs cannot achieve a MSE equal to or less than σ^2, the WLs have to be updated at each iteration, which would increase the complexity.

Proof: For simplicity, in this proof, we assume $c = 1$, however, the results are readily extendable to the general values of c. We construct our proof based on the following assumption:
Assumption: Let $e^{(m)}[n]$'s are independent and identically distributed (i.i.d) zero-mean Gaussian random variables with variance ψ^2.

We have,

$$E\left[\lambda^{(m)}[n]\right] = E\left[\min\left\{1, \left(\delta^{(m)}[n-1]\right)^{l^{(m)}[n]}\right\}\right]$$
$$\leq \min\left\{1, E\left[\left(\delta^{(m)}[n-1]\right)^{l^{(m)}[n]}\right]\right\}. \quad (5.26)$$

Next, we show that under certain conditions, $E\left[\left(\delta^{(m)}[n-1]\right)^{l^{(m)}[n]}\right]$ is less than 1, hence, we can get an upper bound for $E\left[\lambda^{(m)}[n]\right]$. We define $\delta \triangleq \ln(\delta^{(m)}[n-1])$, yielding

$$E\left[\left(\delta^{(m)}[n-1]\right)^{l^{(m)}[n]}\right] = E\left[E\left[\exp\left(\delta\ l^{(m)}[n]\right)\middle|\delta\right]\right] = E\left[\mathbb{M}_{l^{(m)}[n]}(\delta)\middle|\delta\right], \quad (5.27)$$

where $\mathbb{M}_{l^{(m)}[n]}(.)$ is the moment generating function of the random variable $l^{(m)}[n]$. From the Algorithm 4, $l^{(m)}[n] = (m-1)\sigma^2 - \sum_{k=1}^{m-1}\left(e^{(k)}[n]\right)^2$. According to our prior assumption, $\frac{e^{(k)}[n]}{\psi}$ is a standard normal random variable. Therefore, $\sum_{k=1}^{m-1}\left(e^{(k)}[n]\right)^2$ has a Gamma distribution as $\Gamma\left(\frac{m-1}{2}, 2\psi^2\right)$ [162], and hence results in the following moment generating function for $l^{(m)}[n]$,

$$\mathbb{M}_{l^{(m)}[n]}(\delta) = \exp\left(\delta(m-1)\sigma^2\right)\left(1+2\psi^2 s\right)^{\frac{1-m}{2}}$$
$$= \left(\delta^{(m)}[n-1]\right)^{(m-1)\sigma^2}\left(1+2\psi^2\ln\left(\delta^{(m)}[n-1]\right)\right)^{\frac{1-m}{2}}. \quad (5.28)$$

In the above equation, $\delta^{(m)}[n-1]$ is a random variable, with the mean denoted by β. We specify that β approaches to ψ^2 in convergence. We next define a function $\Psi(.)$ such that $E\left[\lambda^{(m)}[n]\right] = E\left[\Psi\left(\delta^{(m)}[n-1]\right)\right]$, and seek to find a condition for $\Psi(.)$ to be a concave function. Then, by using Jensen's inequality for concave functions, we have

$$E\left[\lambda^{(m)}[n]\right] \leq \Psi(\beta). \quad (5.29)$$

Motivated by (5.28), we define $B\left(\delta^{(m)}[n-1]\right) \triangleq \delta^{(m)}[n-1]^{-2\sigma^2}\left(1+2\psi^2\ln\left(\delta^{(m)}[n-1]\right)\right)$ and $\Psi\left(\delta^{(m)}[n-1]\right) \triangleq \left(B\left(\delta^{(m)}[n-1]\right)\right)^{\frac{1-m}{2}}$. By these definitions, we obtain

$$\Psi''\left(\delta^{(m)}[n-1]\right) = \frac{1-m}{2}\left(B\left(\delta^{(m)}[n-1]\right)\right)^{\frac{-m-3}{2}}\left[\left(\frac{-m-1}{2}\right)\left(B'\left(\delta^{(m)}[n-1]\right)\right)^2\right.$$
$$\left. + \left(B\left(\delta^{(m)}[n-1]\right)\right)^2 B''\left(\delta^{(m)}[n-1]\right)\right]. \quad (5.30)$$

For $m > 1$, in order for $\Psi(.)$ to be concave, it is sufficient to have

$$\left(B\left(\delta^{(m)}[n-1]\right)\right)^2 B''\left(\delta^{(m)}[n-1]\right) > \left(\frac{m+1}{2}\right)\left(B'\left(\delta^{(m)}[n-1]\right)\right)^2, \quad (5.31)$$

which results in the following necessary and sufficient conditions:

$$\frac{\left(\delta^{(m)}[n-1]\right)^{2\sigma^2}}{\left(1+2\psi^2\ln\left(\delta^{(m)}[n-1]\right)\right)^2} < \frac{\left(1+2\sigma^2\right)^2}{4(m+1)}, \tag{5.32}$$

and

$$\frac{(1-\chi_1)\sigma^2}{1-2\sigma^2\ln\left(\delta^{(m)}[n-1]\right)} < \psi^2 < \frac{(1-\chi_2)\sigma^2}{1-2\sigma^2\ln\left(\delta^{(m)}[n-1]\right)}, \tag{5.33}$$

where

$$\chi_1 = \frac{\gamma^2(1+2\sigma^2)+\gamma\sqrt{(1+2\sigma^2)^2\gamma^2-4(m+1)(\delta^{(m)}[n-1])^{2\sigma^2}}}{2(m+1)(\delta^{(m)}[n-1])^{2\sigma^2}},$$

$$\chi_2 = \frac{\gamma^2(1+2\sigma^2)-\gamma\sqrt{(1+2\sigma^2)^2\alpha^2-4(m+1)(\delta^{(m)}[n-1])^{2\sigma^2}}}{2(k+1)(\delta^{(m)}[n-1])^{2\sigma^2}},$$

and

$$\gamma \triangleq 1+2\psi^2\ln\left(\delta^{(m)}[n-1]\right).$$

Under these conditions, $\Psi(.)$ is concave, therefore, by substituting $\Psi(.)$ in (5.29), we achieve the bound in (5.25). This concludes the proof of the Theorem. □

5.4.2 MSE Analysis

We provide the MSE analysis for the "weighted updates" mode, which can be readily extended to the other scenarios. Let each constituent WL consists of J different linear adaptive filters, as shown in Fig. 5.1. Based on the region in which the input vector $\boldsymbol{x}[n]$ lies, one of these filters is used for estimating $y[n]$. Here, we consider a fixed partition over the space of the input vectors. Suppose that in each region, say j^{th} region, which is corresponding to the j^{th} linear model of the constituent WL m, there is a vector $\boldsymbol{w}_{o,j}^{(m)}$ such that

$$y[n] = \sum_{j=1}^{J} s_j^{(m)}[n]\left[\left(\boldsymbol{w}_{o,j}^{(m)}\right)^T\boldsymbol{x}[n]+\upsilon[n]\right], \quad m=1,\ldots,M$$

where we assume that the estimation error $\upsilon[n]$ is independent of the region to which $\boldsymbol{x}[n]$ belongs, and has a variance of σ_υ^2. Moreover, suppose that the input vector distributes over the regions of the m^{th} constituent WL, with a probability distribution $\varpi^{(m)}$, i.e., $\boldsymbol{x}[n]$ lies in the j^{th} region of the m^{th} constituent WL with probability $\varpi_j^{(m)}$. Then for the LMS case, under the general separation

assumptions (see [4], chapter 6), and given that all linear filters converge, the MSE of the m^{th} constituent WL at the steady state can be expressed as [4],

$$\epsilon^{(m)} = \sum_{j=1}^{J} \varpi_j^{(m)} \sigma_v^2 \left(1 + \frac{\nu \tilde{\lambda}^{(m)} \mathrm{Tr}(\boldsymbol{R}_{\boldsymbol{x}_j})}{2 - \nu \tilde{\lambda}^{(m)} \mathrm{Tr}(\boldsymbol{R}_{\boldsymbol{x}_j})} \right). \tag{5.34}$$

Here, $\boldsymbol{R}_{\boldsymbol{x}_j}$ is the covariance matrix of the input vectors lying in the j^{th} region. Assuming that $\boldsymbol{x}[n]$ is i.i.d and has the covariance matrix $\boldsymbol{R}_{\boldsymbol{x}}$, we have the following MSE for the m^{th} constituent WL at the steady state,

$$\epsilon_{LMS}^{(m)} = \sigma_v^2 \left(1 + \frac{\nu \tilde{\lambda}^{(m)} \mathrm{Tr}(\boldsymbol{R}_{\boldsymbol{x}})}{2 - \nu \tilde{\lambda}^{(m)} \mathrm{Tr}(\boldsymbol{R}_{\boldsymbol{x}})} \right). \tag{5.35}$$

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

$$\mathrm{Tr}\left(\boldsymbol{R}_{\boldsymbol{y}}\right) = E\left[\boldsymbol{y}^T\boldsymbol{y}\right] = \sum_{m=1}^{M} \left(\sigma_d^2 + \epsilon^{(m)}\right). \tag{5.36}$$

We further assume that the $y[n]$ can be modeled as $y[n] = \boldsymbol{y}^T \boldsymbol{z}_o + \varrho[n]$, in which, $\varrho[n]$ is the estimation noise with the variance σ_ϱ^2, and in general different from $\upsilon[n]$. From [4] (chapter 6), we get

$$\varsigma_{LMS} = \left(\frac{\nu_{\boldsymbol{z}} \mathrm{Tr}(\boldsymbol{R}_{\boldsymbol{y}})}{2 - \nu_{\boldsymbol{z}} \mathrm{Tr}(\boldsymbol{R}_{\boldsymbol{y}})} + 1 \right) \sigma_\varrho^2, \tag{5.37}$$

where $\nu_{\boldsymbol{z}}$ is the step size used for updating the combination coefficients $\boldsymbol{z}[n]$.

We next consider the RLS case for MSE analysis, where we use (5.12) for updating each $\boldsymbol{w}_j^{(m)}[n]$. We use the same model and assumptions as the LMS case. In this case, although the input vectors $\boldsymbol{x}[n]$ do not appear with the same weight in computing the autocovariance matrix $\mathbf{R}_j^{(m)}[n]$, it is reasonable to assume the same weight for all input vectors in steady state. We assume that the weights converge to the upper bound obtained in the Theorem. Thus, from [4] (chapter 6), we get

$$\epsilon_{RLS}^{(m)} = \left(\frac{(1-\gamma)d}{2\tilde{\lambda}^{(m)} - (1-\gamma)d} + 1 \right) \sigma_\epsilon^2, \tag{5.38}$$

where d is the dimension of $\boldsymbol{x}[n]$. Hence, in this case the overall MSE is given as,

$$\varsigma_{RLS} = \left(\frac{(1-\lambda)m}{2 - (1-\lambda)m} + 1 \right) \sigma^2. \tag{5.39}$$

This concludes our analysis for MSE of the proposed algorithms under theoretical bounds.

5.5 Experiments

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

Throughout this section, "PLMS" represents the piecewise linear LMS-based learning algorithm, "SPLMS" represents the piecewise linear LMS-based with soft partitioning (explained in Section 5.3, Remark 3), "BPLMS" represents the boosted piecewise linear LMS-based learning algorithm, and "BSPLMS" represents the boosted piecewise linear LMS-based learning algorithm with soft partitioning. Similarly, "PRLS" represents the piecewise linear RLS-based learning algorithm, and "BPRLS" represents the boosted piecewise linear RLS-based learning algorithm. In addition, we use the suffixes "-WU", "-RU", or "-DR" to denote the "weighted updates", "random updates", or "data reuse" modes, respectively. Also, "LMS-MIX" and "RLS-MIX" denote the conventional LMS-based and RLS-based mixture methods, which are a special case of our methods, i.e., the case when $c = 0$.

In our simulations, we set the step sizes for the LMS update to be 0.02 in all algorithms, except the SPLMS. For SPLMS, the step size for updating the filter coefficients is set to 0.1, while the step size for the regions boundaries update is set to 0.5. In addition, for boosted RLS algorithm, we set the forgetting factor $\alpha = 0.99$ for weighted updates and random updates, and $\alpha = 0.995$ for data reuse updates. In all of the RLS-based algorithms, we set $\lambda = 0.999$ as the forgetting factors of PRLS and updating the combination weights in the boosting algorithms. Moreover, we choose σ^2 as the desired MSE parameter, $K = 2$ for data reuse approach, $c = 1$ for all boosting algorithms, and $M = 5$ as the number of constituent WLs, in all our experiments, except the experiments where we investigate the effects of these parameters.

We compare the performance of our methods with that of the best constituent WL. To this end, we provide the Accumulated Square Error (ASE) results as well as the relative improvements in the values of the ASE as our performance measure.

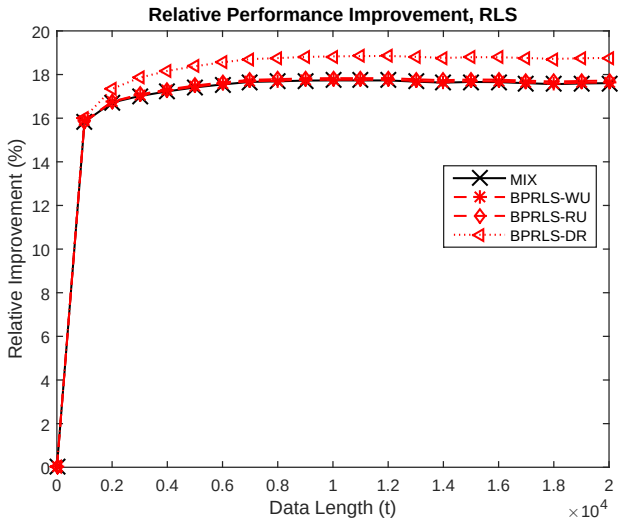


Figure 5.5: The relative improvement in ASE performance of the RLS-based algorithms in the stationary data experiment.

We define the relative improvement in the ASE as $\varphi_{alg} \triangleq 100 \times \dfrac{ASE_0 - ASE_{alg}}{ASE_0}$, where, ASE_0 denotes the ASE of the single piecewise linear algorithm that is to be boosted, e.g., ASE of PLMS, and ASE_{alg} indicates the ASE of proposed boosting algorithm, e.g., BPLMS. All of the results are averaged over 30 rounds.

5.5.1 Stationary Data

In this experiment, we consider the case where the desired data is generated by a piecewise linear model with 3 regions. The input vectors $\boldsymbol{x}[n] = [x_1 \ x_2 \ 1]$ are $3-$dimensional, and $[x_1 \ x_2]$ is drawn from a jointly Gaussian random process, and then scaled such that $[x_1 \ x_2]^T \in [0 \ 1]^2$. We include 1 as the third entry of $\boldsymbol{x}[n]$ to consider affine models. Specifically, the desired data is generated by the following model,

$$y[n] = \begin{cases} [1 \ 1]\boldsymbol{x}[n] + \upsilon[n] & \text{if } \Phi_0^T \boldsymbol{x}[n] < 0.5 \\ 0.5 + \upsilon[n] & \text{if } 0.5 \leq \Phi_0^T \boldsymbol{x}[n] < 0.8 \\ [1 \ -0.5]\boldsymbol{x}[n] + \upsilon[n] & \text{if } \Phi_0^T \boldsymbol{x}[n] \geq 0.8 \end{cases} \tag{5.40}$$

where $\Phi_0 = [1 \ 1]^T$, and $\upsilon[n]$ represents random Gaussian noise. We use 5 piecewise linear models, each with 2 regions, as the constituent WLs in boosting. The regions corresponding to each WL are different, and for the linear model within each region, the initial values are set to zero.

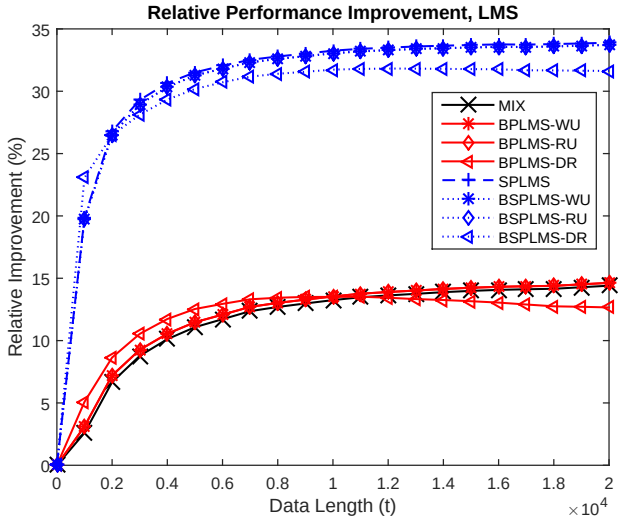


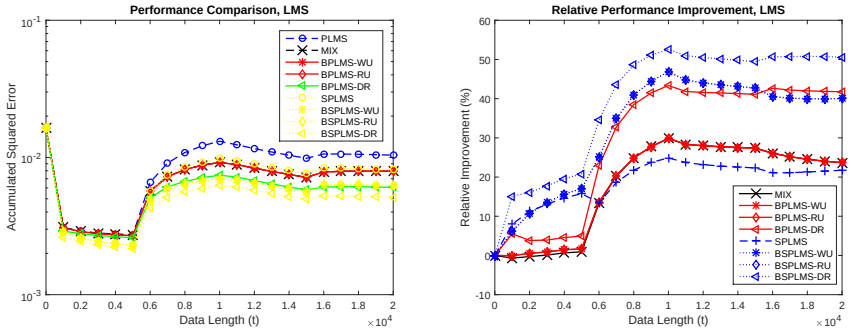
Figure 5.6: The relative improvement in ASE performance of the RLS-based algorithms in the stationary data experiment.

As depicted in Fig. 5.5 and Fig. 5.6, our proposed methods significantly boost the performance of a single piecewise linear model, in both LMS and RLS cases. It is interesting to note that, the random updates method achieves the same performance as the weighted updates method and the conventional mixture method, with a significantly lower computational complexity. Furthermore, the results show that the proposed algorithms with the data reuse boosting approach, perform better than the conventional mixture method. However, since the complexity of data reuse updates is considerably higher than other approaches, one may prefer to use the other boosting approaches. Specifically, as shown in Fig. 5.5 and Fig. 5.6, the random update method, provides a satisfactory performance, while its complexity is much lower than other approaches.

5.5.2 Non-stationary Data

In the next experiment we generate the training data using a non-stationary and piecewise linear model, such that the total data interval is divided into four exclusive intervals of length $N/4$ and N is the total number of samples. We generate $\boldsymbol{x}[n] = [x_1 \ x_2]^T \in [0 \ 1]^2$ as in the stationary data experiment. During each of the data intervals, we assume a different $3-$regions piecewise linear model.

We use the proposed online boosting algorithm with $M = 5$ WLs and each WL is a piecewise learners with $2-$regions. We choose a target MSE of 0.01 and update our algorithm's parameters using gradient descent, e.g., LMS, with a learning rate 0.02. We partition the input space by a separating hyperplane given



(a) ASE performance

(b) relative performance improvement

Figure 5.7: The performance results of the boosting approaches in the non-stationary experiment.

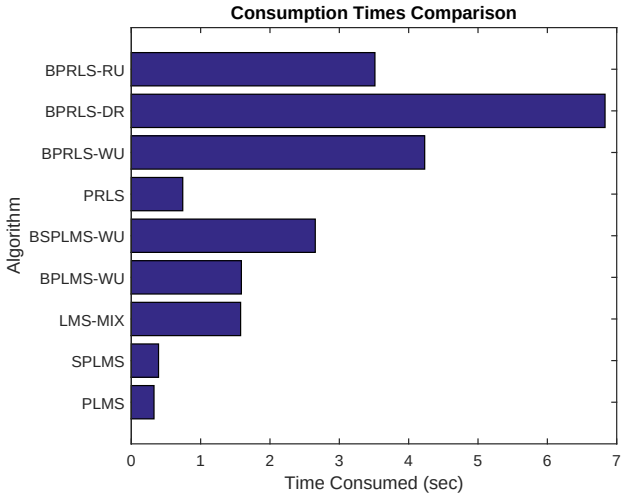


Figure 5.8: The time consumed by each algorithm in the non-stationary data experiment.

by $\boldsymbol{\theta} = [\theta_1\ \theta_2\ -\theta_3]^T$, which is used as the initial direction vector in "SPLMS" and "BSPLMS" cases as well. Each element of vector $\boldsymbol{\theta}$ is a zero mean random variable. We demonstrate in Fig. 5.7 that the proposed boosting approaches achieve significant performance gain over the single piecewise learner and ordinary mixture methods. Specifically, the data reuse approach gives the best performance while the random update methods achieves a comparable performance despite much lower computational time as seen from the Fig. 5.8. The lower complexity makes the random update approach suitable for big data processing and streaming applications. We repeated the experiments on non-stationary data with the RLS-based algorithms and get significantly good results, which shows that the proposed boosting methods can perform well with RLS updates too.

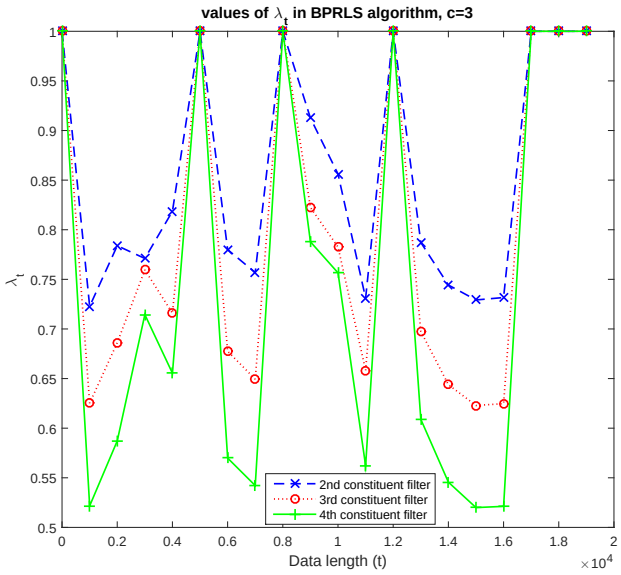
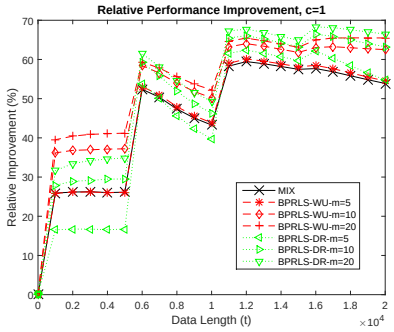


Figure 5.9: The changing of the weights in BPRLS-WU algorithm in the non-stationary data experiment. All experiments have been done on the same computer.

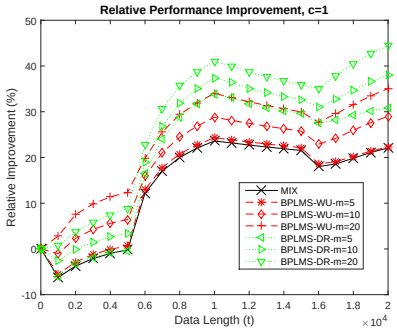
Moreover, from the Fig. 5.9, we analyze the approximate changes of the weights, in a BPRLS algorithm implemented on the non-stationary data. As shown in this figure, the weights do not change monotonically, that shows the capability of our algorithm in effective tracking of the non-stationary data. Furthermore, since we update the WLs in an ordered manner, i.e., we update the $\{km+1\}^{th}$ WL after the m^{th} WL is updated, the weights, i.e., the performance metric, assigned to the last WLs are generally smaller than the weights assigned to the previous WLs. As an example, in Fig. 5.9 we see that the weights assigned to the second constituent WL are larger than those of the third and fourth WLs. Moreover, note that in this experiment, we have set $c = 1$ as the dependency parameter. We should mention that increasing this parameter, in general, causes the lower weights, hence, it can considerably reduce the complexity of random updates and make it a useful method for big data processing tasks.

5.5.3 The Effect of Parameters

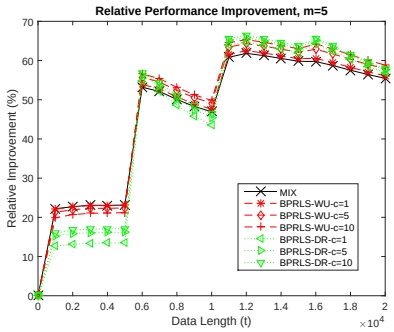
In this section, we investigate the effects of the dependence parameter c as well as the number of constituent WLs, on the boosting performance of our methods in the non-stationary data experiment, explained in Section 5.5.2. From the results in Fig. 5.10a and Fig. 5.10b, we observe that, increasing the number of constituent filters can improve the performance significantly. However, as shown



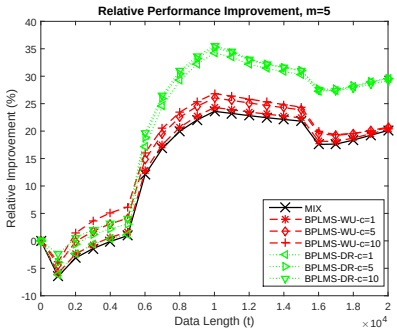
(a) The effect of the parameter m



(b) The effect of the parameter m



(c) The effect of the parameter c



(d) The effect of the parameter c

Figure 5.10: The effect of the number of WLs m and dependency parameter c on the relative performance improvement of the RLS and LMS-based algorithms in the non-stationary data experiment.

in Fig. 5.10c and 5.10d, in this experiment, for the short-length data, increasing the dependency parameter cannot improve the performance. Nevertheless, in this experiment, as the data length increases, we can get a better performance by using a larger value for dependency parameter.

5.5.4 Benchmark Real and Synthetic Data Sets

In this section, we demonstrate the efficiency of the introduced methods over some widely used real and synthetic regression data sets. The data sets we have used can be found in [108]. We have normalized the data to the interval $[-1, 1]$ in all algorithms. These experiments show that our algorithms can successfully improve the performance of single piecewise linear filters, and in some cases, even outperform the conventional mixture method. We now describe the experiments and provide the results:

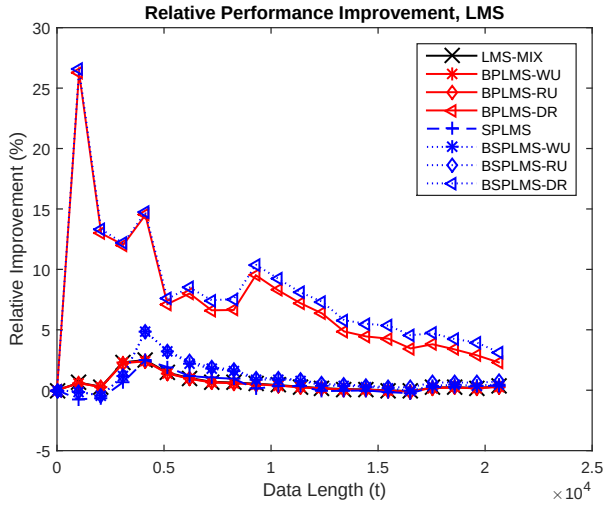


Figure 5.11: The performance of LMS-based boosting methods on the California Housing data set.

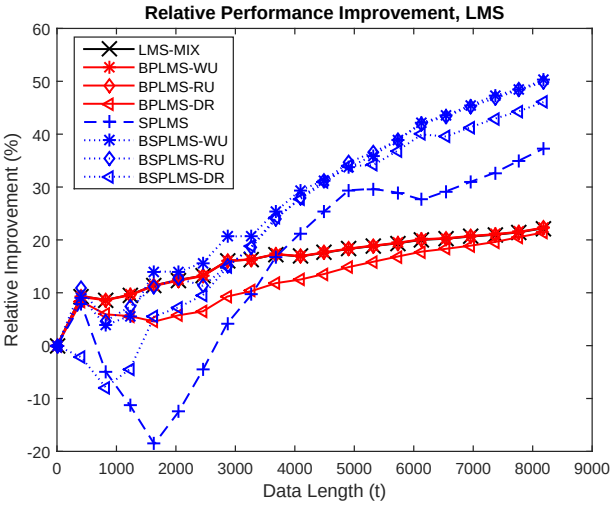


Figure 5.12: The performance of LMS-based boosting methods on the CompAct data set.

1. California Housing: This data set has been obtained from StatLib repository. They have collected information on the variables using all the block groups in California from the 1990 Census. We are going to find the house median values, based on the given attributes. For further description one can refer to [108]. In this experiment we have used the same parameters as the non-stationary experiment, for the filters. The results in Fig. 5.11 shows that our methods can perform well on this data set.

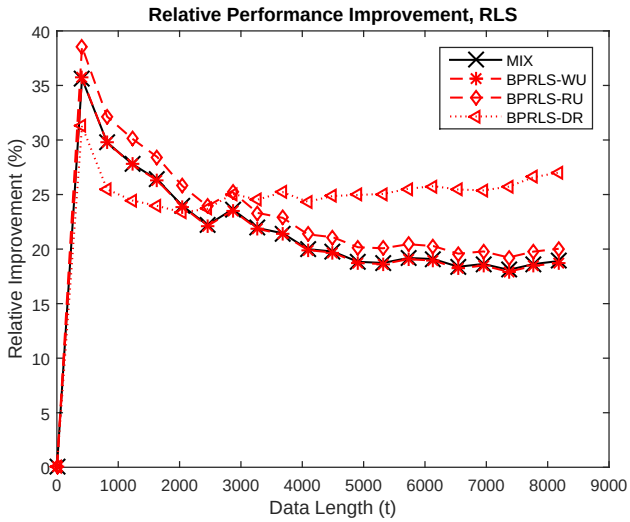


Figure 5.13: The performance of RLS-based boosting methods on the CompAct data set.

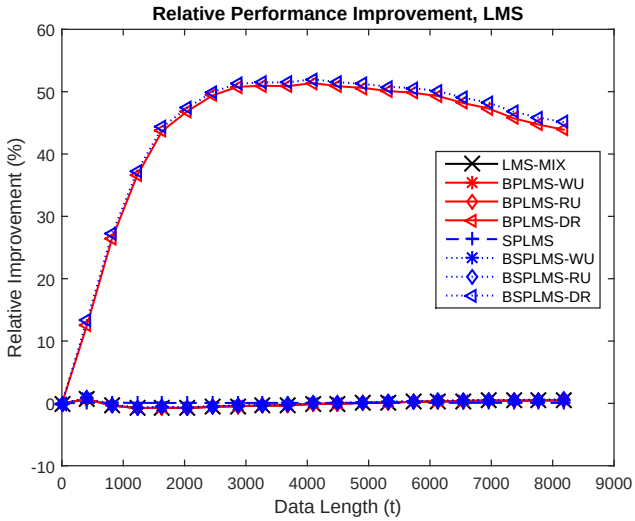


Figure 5.14: The performance of LMS-based boosting methods on the Bank data set.

2. Computer Activity (CompAct): This real data set is a collection of computer systems activity measures [108]. The task is to predict usr, the portion of time that CPUs run in user mode from all attributes [108]. In our simulation for this data set, we have set $J = 2$, $M = 5$, and $c = 1$. Also, for all of LMS based algorithms we have used 0.03 as the step size for linear filter coefficients updates, except for data reuse scenarios. For BPLMS-DR we have used a step size of 0.01, and for BSPLMS-DR we have used a step size of 0.05. The results shown in Fig. 5.12 and Fig. 5.13 indicate the superior performance of our algorithms.

3. Bank: This data set is generated from a simulation of how bank-customers

choose their banks [108]. The algorithm should predict the fraction of bank customers who leave the bank because of full queues. We used the variant with m=32 (This is related to the data set, and one should not confuse this m with the number of constituent filters.) [108]. We have used the same parameters as the non-stationary data experiment described in Section 5.5.2 for this experiment. The results are provided in Fig. 5.14.

5.6 Discussion

In this chapter, we investigate online and nonlinear modeling in a deterministic setting. We further improve upon the adaptive and piecewise nonlinear modeling by incorporating the boosting approach. We introduce a novel family of boosted online regression algorithms and proposed three different boosting approaches, i.e., weighted updates, data reuse, and random updates, which can be applied to different online learning algorithms. We provide theoretical bounds for the computational complexity and MSE performance of our proposed methods in a strong mathematical sense. We emphasize that while using the proposed techniques, we do not assume any prior information about the statistics of the desired data or feature vectors. We show that by the proposed boosting approaches, we can significantly improve the MSE performance of the conventional online linear algorithms, e.g linear regression with LMS and RLS updates, and the nonlinear or piecewise learning algorithms using LMS and RLS as linear algorithms at each region of the piecewise model. Moreover, we provide an upper bound for the weights generated during the algorithm that leads us to a thorough analysis of the computational complexity of these methods. The computational complexity of the random updates method is remarkably lower than that of the conventional mixture-of-experts and other variants of the proposed boosting approaches, without degrading the performance. Therefore, the boosting using random updates approach is an elegant alternative to the conventional mixture-of-experts method when dealing with real life large scale problems. Furthermore, the proposed boosting techniques provide further robustness to the online nonlinear models discussed in Chapter 2 without considerably increasing the complexity. We provide several results that demonstrate the strength of the proposed algorithms over a wide variety of synthetic as well as real data.

Chapter 6

Online Channel Estimation for Underwater Acoustic Communication

In this chapter, we investigate the channel estimation of underwater acoustic communication as a real life application of nonlinear learning. We use online learning to estimate the channel impulse response (CIR) of communication media, in a non-stationary environment. We specifically propose a novel family of online, robust and nonlinear estimation algorithms for highly non-stationary underwater acoustic (UWA) channels. The underwater environment is highly challenging because of the non-stationarity and effects of impulsive noise. Therefore, we seek online processing algorithms that are based on the minimization of a logarithmic cost function. Fundamentally, we seek a better trade-off between the steady state performance and the convergence of the estimation algorithm. For this matter, we combine various norms of the error with a logarithmic term, that alleviate the stability issues due to the impulsive noise while improving the convergence rate of the conventional first and second order methods. Hence, the algorithms significantly enhance the stability against the abrupt changes while achieving a comparable convergence rate to the faster algorithms. We specifically provide an extensive analysis of the proposed algorithms for the tracking and steady-state performances in the presence of impulsive noise. Apart from the impulsive noise, we also consider frequency and phase offsets that commonly affect the UWA channel in real life.

Underwater acoustic (UWA) communication has been widely studies in recent years due to the advent of new applications such as the exploitation of underwater resource exploitation and the study of marine life [85–87]. However, the

UWA channel is considered to be extremely hostile due to the constant movement of waves, large delay spreads, multi-path propagation, Doppler effects, and frequency dependent propagation loss [87,88]. These effects make the UWA channel fastly varying, non-stationary and afflicted by abrupt changes. The channel impulse response is usually long and non-stationary, i.e., the channel is represented by a large number of coefficients that are also time varying. To mitigate these effects, orthogonal frequency division multiplexing (OFDM) is used as an elegant solution [92]. However, the OFDM requires the channel to be perfectly matched, and hence, channel equalization [90,91] plays a key role in providing reliable high data rate communication [87]. We need the channel estimation algorithms to be robust to the rapid changes due to unpredictable nature of underwater environment [87]. Besides, the impulsive noise results in stability issues for adaptive channel estimation [167,168]. Consequently, we introduce a family of novel adaptive nonlinear channel estimation algorithms to achieve faster convergence and strong stability counter to the impulsive noise in the UWA channels.

In the digital and wireless communication contexts, the additive white Gaussian noise (AWGN) model is widely used, however, it is highly insufficient to counter the effects of the ambient noise in UWA channels [169–171]. For instance, in warm shallow waters, the high frequency noise component is dominated by the "impulsive noise" [172–174] resulted from numerous noise sources such as shipping traffic, underwater explosives, marine life, and offshore oil exploration-production [175]. The impulsive noise is infrequent and consists of relatively short duration, however it's amplitude can be extremely high. We specifically propose a radical approach to the robust channel estimation in order to mitigate the effects of the impulsive noise, in this chapter. In particular, we seek to provide adaptive algorithms that show faster convergence as well as robustness against the impulsive noise.

Linear adaptive channel estimation methods (e.g., sign algorithm (SA), least mean squares (LMS) or least mean fourth (LMF) algorithms) are the simplest methods in terms of computational complexity [176,177]. However, in the impulsive noise environment of UWA, the conventional linear estimators either have a slow convergence (e.g., sign algorithm (SA)) or suffer from stability issues (e.g., LMF) [178]. Generally, such methods are based on the minimization of a loss function $C(e[m]) \triangleq E\left[|e[m]|^k\right]$ (where $E[.]$ denotes the expectation), using the stochastic gradient descent technique [4, 5]. However, there is always a trade-off between the convergence and the robustness against impulsive noise, in the estimation algorithms [178].

The mixed norm algorithms, taking advantage of the robustness offered by low powers and the convergence speed offered by high powers, are used to achieve a better trade-off [179, 180]. However, such algorithms needs "a priori" knowledge

of the noise and input signal statistics [178]. On the other hand, the mixture of experts algorithms [181, 182], learn the best combination parameters on the fly, however offer significantly higher computational complexity [144, 178].

In this chapter, we use a logarithmic function as a correction term in the cost function of the well-known robust estimation methods. In this sense, we obtain a comparable performance to the robust algorithms, while retaining the fast convergence of conventional least square methods. We choose a conventional method that uses a certain power of the error as the cost, e.g, least mean squares (LMS), and enhance it by adding a logarithmic term to its cost function. In the occurrence of impulse, the error is high and diminishes the roll of logarithmic term. On the other hand, for the small errors, i.e., in the absence of impulsive noise, a correction term is added, that leads to a faster convergence.

To this end, we use first order methods, i.e., logarithmic cost least mean absolute (LCLMA) and logarithmic cost least mean square (LCLMS) algorithms, and second order methods, i.e., logarithmic cost recursive least squares (LCRLS), and logarithmic cost recursive least absolutes (LCRLS), for channel estimation using the logarithmic cost. We provide the results showing improved performance of the proposed algorithms through simulations that imitate real life UWA communication . Subsequently, we provide an extensive analysis of the tracking and steady state performance of our algorithms.

In the following, we give an organizational flow of the chapter: We describe the problem and the experimental setup, in Section 6.1. In Section 6.2, we propose and investigate a family of stable and fast converging channel estimation algorithms based on the logarithmic cost functions. We provide several simulation results in Section 6.3, where we show the performance gain of the proposed methods in terms of bit error rate (BER) and mean square error (MSE), over the conventional estimation algorithms. We finally conclude the chapter with brief discussion on the efficacy of our contributions, in Section 6.4.

6.1 Problem description

6.1.1 Notations

All vectors are column vectors and are denoted by boldface lower case letters and all matrices are denoted by boldface upper case letters. For a vector $\boldsymbol{x}$, $\boldsymbol{x}^H$ is the Hermitian transpose. The zero vector of size L is denoted by $\mathbf{0}_L$. We denote the expectation of the random variable x by $E[x]$ and the trace of the matrix

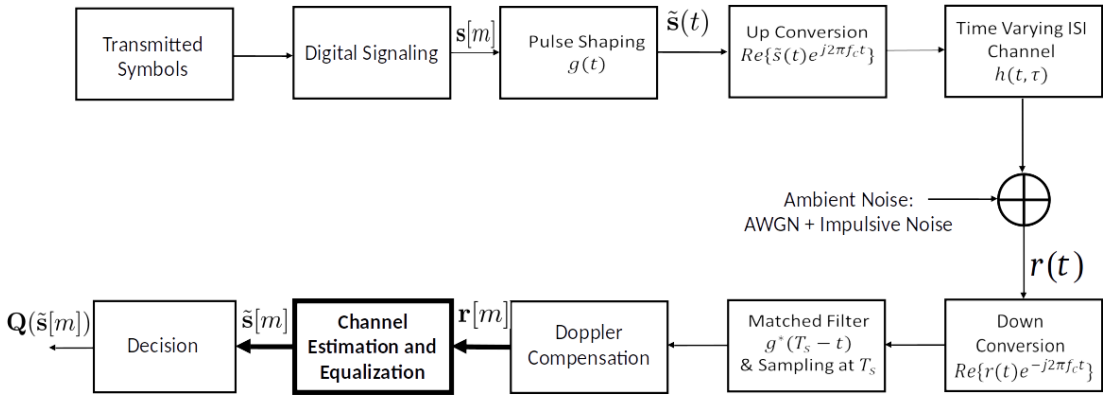


Figure 6.1: The block diagram shows the flow of transmitted and received signals through a time varying intersymbol interference (ISI) channel that is affected by AWGN and impulsive noise $n(t)$. The transmitted data $\{s[m]\}_{m\geq 1}$ are modulated, "pulse shaped" with a raised cosine filter $g(t)$ and up-converted to the carrier frequency f_c. The channels equalizer receives $r[m]$ and generates the soft output $\tilde{s}[m]$. $Q(\tilde{s}[m])$ is the hard estimation for the m^{th} transmitted symbol. We use use the channel estimation to adapt the equalizer for reduction of ISI.

$\boldsymbol{X}$ by $\mathrm{Tr}(\boldsymbol{X})$. We show the discrete time variables enclosed in braces and the continuous time variable t in parenthesis. Furthermore, we define $\|\boldsymbol{x}\|^2 \triangleq \boldsymbol{x}^H\boldsymbol{x}$ as the squared l_2-norm and $\|\boldsymbol{x}\|^2_{\boldsymbol{Q}'} \triangleq \boldsymbol{x}^T\boldsymbol{Q}'\boldsymbol{x}$ as the weighted squared l_2-norm, of vector $\boldsymbol{w}$, where $\boldsymbol{Q}'$ is a positive definite weighting matrix.

6.1.2 Setup

As shown in the Fig. 6.1, $r(t)$ is the received signal where $r(t) \in \mathbb{R}$, and we seek to extract the transmitted symbols $\{s[m]\}_{m\geq 1}$. We pass the symbols $\{s[m]\}_{m\geq 1}$ through the raised cosine pulse shaping filter $g(t)$ before transmission, that generates the signal $\tilde{s}(t)$. We then up-convert the signal to the carrier frequency f_c followed by sending it through the channel. The received signal at time t is given as [189],

$$r(t) = \int_0^{ds_{max}} \tilde{s}(t-\tau)h(t,\tau)d\tau + \eta(t), \tag{6.1}$$

where $h(t,\tau)$ shows the CIR at time t and ds_{max} represents the maximum delay spread of the channel. Furthermore, $n(t)$ represents the ambient noise of the channel, which is,

$$\eta(t) = \eta_w(t) + \eta_i(t), \tag{6.2}$$

i.e., a combination of the the white Gaussian noise $\eta_w(t)$ and the impulsive noise $\eta_i(t)$. We emphasize that the effects of time delay and phase deviations are usually

addressed at the front-end of the receiver, and hence, are out of the explicit scope of this chapter. The received signal is then sampled at each symbol duration T_s seconds such that the discrete time channel model is represented as [189],

$$r[m] = \sum_{k=n}^{N-1} s[m-n]h[m,n] + \eta[m], \tag{6.3}$$

and

$$\eta[m] = \eta_a[m] + \eta_i[m], \tag{6.4}$$

where $N = \lfloor ds_{max}/T_s \rfloor$ indicates the length of the CIR, i.e., the channel is represented as an $N-$tap filter. In the following sections, we use the discrete time model of the channel to address the estimation and equalization problems.

6.1.3 Channel Estimation and Causal ISI Removal

We seek to estimate the communication channel CIR, $\boldsymbol{h}[m]$, followed by estimating the transmitted symbols $\{s[m]\}_{m\geq 1}$ from the channel outputs $\{r[m]\}_{m\geq 1}$. We propose a novel adaptation method to obtain an accurate estimate of the causal part of the channel $\hat{\boldsymbol{h}}[m]$. Furthermore, the output of the estimated channel is given by $\hat{r}[m] = \hat{\boldsymbol{h}}^T[m]\boldsymbol{s}[m]$, where $\boldsymbol{s}[m] \triangleq [s[m], \ldots, s[m-N+1]]^T$ is a vector containing the current and past $N-1$ transmitted symbols. We use the instantaneous error $e[m] = r[m] - \hat{r}[m]$ to adjust the estimated channel parameters after each iteration, as shown in Fig. 6.2. In the training mode, we use $s[m]$ for the online estimation of the channel, whereas in the decision directed mode we use the quantized estimates $Q(\tilde{s}[m])$

As a first step at the receiver, we remove the inter-symbol interference (ISI) effect generated by the causal part of the channel, i.e., $\boldsymbol{h}[m]$, to get "clean" symbols $\tilde{r}_m$. We then use the past N_c clean symbols at each time, in the causal part of the equalizer, to further mitigate the ISI effect. For a channel of length N, we need to mitigate the effect of the past $N-1$ symbols from each received symbol $r[m]$, based on the estimated channel at time m. Thereby, inserting in a matrix form, we obtain

$$\tilde{\boldsymbol{r}}[m+1] = \boldsymbol{r}[m+1] - \hat{\boldsymbol{H}}[m][s[m-N_c-N+2], ..., s[m]]^T, \tag{6.5}$$

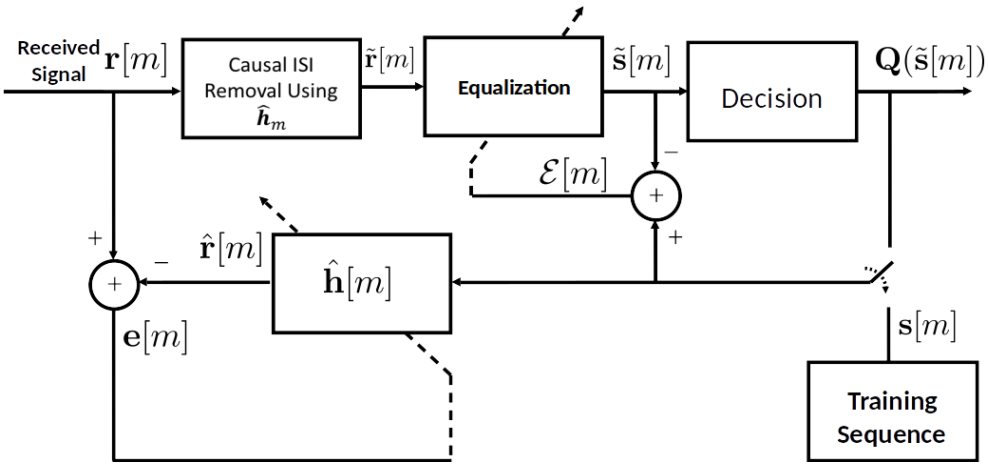


Figure 6.2: A detailed diagram for channel equalization and estimation block of Fig. 6.1. We introduce algorithms for channel estimation to determines how should $\hat{\boldsymbol{h}}[m]$ be updated based on the error $e[m]$), for the specific case of channels suffered from the impulsive noise.

where $\hat{\boldsymbol{H}}[m]$ is the channel convolution matrix given by,

$$\hat{\boldsymbol{H}}[m] \triangleq \begin{pmatrix} \hat{\boldsymbol{h}}^T[m-N_c+1] & & \mathbf{0}^T_[N_c-1] \\ 0 & \hat{\boldsymbol{h}}^T_[m-N_c+2] & \mathbf{0}^T_[N_c-2] \\ \vdots & \ddots & \vdots \\ \mathbf{0}^T[N_c-1] & & \hat{\boldsymbol{h}}^T[m] \end{pmatrix}_{N_c\times(N_c+N-1)}.$$

In addition, $\boldsymbol{r}[m] = [r[m-N_c+1], ..., r[m]]$ is the $N_c \times 1$ received signal vector, and $\tilde{\boldsymbol{r}}[m] = [\tilde{r}[m-N_c+1], ..., \tilde{r}[m]]$ is the cleaned version of the received signal vector ($\boldsymbol{r}[m]$) to be used as the input of the causal part of the equalizer.

We define a cost function $C(e[m])$ for the update of the channel estimate $\hat{\boldsymbol{h}}[m]$, e.g., in the LMS method $\Lambda(e[m]) = E[|e[m]|^2]$. Then, we develop an algorithm for updating $\hat{\boldsymbol{h}}[m]$ based on the minimization of this cost function using the stochastic gradient descent method [4]. Therefore, we update the tap weights vector as follows.

$$\hat{\boldsymbol{h}}[m+1] = \hat{\boldsymbol{h}}[m] - \frac{1}{2}\epsilon\nabla_{\hat{\boldsymbol{h}}_{[m]}}\tilde{\Lambda}(e[m]),$$

where $\nabla_{\hat{\boldsymbol{h}}_{[m]}}\tilde{\Lambda}(e[m])$ is an instantaneous approximation of the gradient of the cost function $\Lambda(e[m])$ with respect to $\hat{\boldsymbol{h}}[m]$ [4]. We take the LMS method as an example, for which $\Lambda(e[m]) = E[|e[m]|^2]$. Then we update the tap weights vector of an LMS estimator as

$$\hat{\boldsymbol{h}}[m+1] = \hat{\boldsymbol{h}}[m] + \epsilon e[m]\boldsymbol{s}[m], \tag{6.6}$$

where $\epsilon > 0$ is the learning rate (step size) of the LMS update.

To achieve a more robust adaptive algorithm, we propose an updating algorithm through the minimization of a different cost function. The cost functions of the form $\Lambda(e[m]) = E[|e[m]|^n]$, which consist of the n^{th} power of the error, depending on the value of n, either have a slow convergence, or do not perform well from the stability viewpoint, in the presence of impulsive [178]. Therefore, we next add a logarithmic term in the cost function, in Section 6.2. The introduction of logarithmic term intrinsically employs different powers of the error into the cost function. Consequently, when a significant error occurs due to the impulsive noise, the algorithm mitigates the effect on the current sample by updating the equalizer coefficients using the lower order norms of the error, i.e., since the logarithmic term is much smaller than the regular cost function. On the other hand, in the impulse-free environments, the algorithm accelerates the convergence using higher order norms of the error.

6.2 Robust Channel Estimation Based on Logarithmic Cost Functions

In this section, we explain our method mathematically and introduce two channel estimation algorithms based on the logarithmic costs [178]. Here, we first describe the most important characteristics of the logarithmic functions in this context. Let $\Psi(e[m]) : \mathbb{R} \to [0, \infty)$ represents a non-negative function of the error, which is a monotonically increasing function of $|e[m]|$. It can be readily shown that the function $K(e[m]) \triangleq \ln(1 + \Psi(e[m]))$ has the following properties:

1. $\lim\limits_{|e[m]|\to\infty} \frac{K(e[m])}{\Psi(e[m])} = 0.$
2. By Maclaurin series expansion of the natural logarithm:
   As $\Psi(e[m]) \to 0 : K(e[m]) \approxeq \sum\limits_{j=1}^{\infty} \frac{(-1)^{j+1}\Psi(e[m])^j}{j}$

The first property reveals that, for high error values, e.g., noise impulses, the logarithmic function $K(e[m])$ is negligible relative to the original cost function $\Psi(e[m])$. However, the second property shows that the greater powers of the cost function $\Psi(e[m])$ are contributing towards the logarithmic function $K(e[m])$ when the error values are small. By leveraging these two properties, we proceed to define a new cost function, $\Lambda(e[m])$, based on a primary cost function $\Psi(.)$, as

$$\Lambda(e[m]) \triangleq \Psi(e[m]) - \frac{1}{b}\ln(1 + b\Psi(e[m])), \tag{6.7}$$

where $b > 0$ is a design parameter. From the properties of the logarithmic function [178], we deduce that when $b\Psi(e[m]) \ll 1$,

$$\Lambda(e[m]) \to \frac{b}{2}\Psi^2(e[m]) - \frac{b^2}{3}\Psi^3(e[m]) + \dots$$

and,

$$\Lambda(e[m]) \to \Psi(e[m]), \quad \text{as} \;\; e[m] \to \pm\infty.$$

The new cost function $\Lambda(e[m])$ is a convex function of $e[m]$ if the cost function $\Psi(e[m])$ is convex. Hence, in the new algorithm, the cost functions is equivalently consisted of the first and second powers of the primary cost function $\Psi(e[m])$, based on the amount of error.

6.2.1 First Order Methods for the Update of CIR Coefficients

Here, we use first order methods to adjust the estimate of CIR $\hat{\boldsymbol{h}}[m]$. We use the stochastic gradient descent method [4] to derive a recursion expression for updating $\hat{\boldsymbol{h}}[m]$. This yields,

$$\begin{aligned}\hat{\boldsymbol{h}}[m+1] &= \hat{\boldsymbol{h}}[m] - \frac{1}{2}\epsilon\nabla_{\hat{\boldsymbol{h}}[m]}\Lambda(e[m]) \\ &= \hat{\boldsymbol{h}}[m] - \epsilon\,\frac{b\Psi(e[m])}{1+b\Psi(e[m])}\left[\nabla_{\hat{\boldsymbol{h}}[m]}\Psi(e[m])\right].\end{aligned}$$

We use the squared error norm $\Psi(e_m) = E[|e[m]|^2]$, as our primary cost function $\Psi(e[m])$. Let $\Psi(e[m])$ is the expectation of another function $\Theta(e[m])$, i.e., $\Psi(e[m]) = E[\Theta(e[m])]$. Then, using the instantaneous approximation for $\Psi(e[m])$ [4], the general stochastic gradient update is given by,

$$\hat{\boldsymbol{h}}[m+1] = \hat{\boldsymbol{h}}[m] + \epsilon\,\frac{b\Theta(e[m])}{b\Theta(e[m])+1}\left[\nabla_{e[m]}\Theta(e[m])\right]\boldsymbol{s}[m]. \tag{6.8}$$

With respect to our defined primary cost function $\Theta(e[m])$, we present two different first order channel estimation methods based on the squared norm of the error and the absolute norm of the error, i.e.,

1. *The logarithmic cost least mean squares (LCLMS)*:
   Here, we use $\Theta(e[m]) = |e[m]|^2$, which yields the following update on the

tap coefficients vector

$$\hat{\boldsymbol{h}}[m+1] = \hat{\boldsymbol{h}}[m] + \epsilon \, \frac{b \, |e[m]|^2}{b \, |e[m]|^2 + 1} \, [2e[m]] \, \boldsymbol{s}^*[m]$$
$$= \hat{\boldsymbol{h}}[m] + \epsilon' \, \frac{e[m] \, |e[m]|^2}{b \, |e[m]|^2 + 1} \boldsymbol{s}^*[m]$$

2. *The logarithmic cost least mean absolutes (LCLMA)*:
   Here, we use $\Theta(e[m]) = |e[m]|$, which leads to,

$$\hat{\boldsymbol{h}}[m+1] = \hat{\boldsymbol{h}}[m] + \epsilon \, \frac{b \, |e[m]|}{b \, |e[m]| + 1} \, [\mathrm{csgn}(e[m])] \, \boldsymbol{s}^*[m]$$
$$= \hat{\boldsymbol{h}}[m] + \mu' \, \frac{e[m]}{b \, |e[m]| + 1} \boldsymbol{s}^*[m].$$

The simulation results show that the LCLMS algorithm considerably improve the speed of convergence with comparable stability over the conventional LMS algorithm. Furthermore, the LCLMA algorithm results in significant performance improvement as compared to the SA algorithm, while retaining the robustness in countering the impulsive noise. Hence, the proposed methods are elegant alternatives to the conventional methods for UWA channel estimation. *Remark 1:* These methods can be directly applied to decision feedback equalizers (DFEs). In this sense, we can update the feed-forward and feedback filter coefficients in such equalizers, with different methods. However, since we use the same error as the feed-forward filter, to update the feedback filter, this part is also affected by the impulsive noise. Thus, we use the same robust adaptive algorithm in both feed-forward and feedback parts of the equalizer. To this end, we append the past decided symbols to the received signal vector as

$$\check{\boldsymbol{r}}[m] \triangleq [r[m], \ldots, r[m-l+1], \bar{s}[m-1], \ldots, \bar{s}[m-l_f]]^T,$$

where l_f is the length of the feedback part of the equalizer. Also, $\hat{b}[m] = Q(\hat{s}[m])$ denotes the hard decided (quantized) symbol for the transmitted bit $b[m]$. Furthermore, corresponding to this extension in the received signal vector, we merge the feed-forward and feedback equalizers to obtain an extended filter of length $l + l_f$ as $\check{\boldsymbol{h}}[m] \triangleq [\boldsymbol{h}^T[m] \quad \boldsymbol{h}_f^T[m]^T]$, where $\boldsymbol{h}_f[m]$ represents the feedback filter at time m. Hence, $\hat{s}[m] = \check{\boldsymbol{h}}^T[m] \, \check{\boldsymbol{r}}[m]$. We update the extended tap weights vector $\check{\boldsymbol{h}}[m]$ with the desired method, e.g., LCLMA.

Remark 2: For further robustness, we normalize the updates with respect to the received signal vector to make the updates more independent from the received signal. Hence, we proceed to minimize $\Lambda(\frac{e[m]}{\|\boldsymbol{r}[m]\|})$ with respect to $\hat{\boldsymbol{h}}$, which yields

the following update

$$\hat{\boldsymbol{h}}[m+1] = \hat{\boldsymbol{h}}[m] + \epsilon \, \frac{a\Theta(\frac{e[m]}{\|\boldsymbol{s}[m]\|})}{b\Theta(\frac{e[m]}{\|\boldsymbol{s}[m]\|}) + 1} \left[\nabla_{\frac{e[m]}{\|\boldsymbol{s}[m]\|}} \Theta(\frac{e[m]}{\|\boldsymbol{s}[m]\|})\right] \frac{\boldsymbol{s}[m]}{\|\boldsymbol{s}[m]\|}.$$

For instance, the update in case of normalized LCLMA will be

$$\begin{aligned}\hat{\boldsymbol{h}}[m+1] &= \hat{\boldsymbol{h}}[m] + \epsilon \, \frac{b \, \frac{e[m]}{\|\boldsymbol{s}[m]\|}}{b \, |\frac{e[m]}{\|\boldsymbol{s}[m]\|}| + 1} \frac{\boldsymbol{s}[m]}{\|\boldsymbol{s}[m]\|} \\ &= \hat{\boldsymbol{h}}[m] + \epsilon \, \frac{b \, e[m]}{(a \, |e[m]| + \|\boldsymbol{s}[m]\|)\|\boldsymbol{s}[m]\|} \boldsymbol{s}[m].\end{aligned}$$

6.2.2 Second Order Methods for the Update of CIR coefficients

Here, we define the cost function as $\Gamma[j] \triangleq \sum_{j=0}^{k} \lambda^{k-j} \Lambda(e[j])$. We seek to update the channel coefficients $\hat{\boldsymbol{h}}[k]$ such as to minimize $\Gamma[k]$, i.e., $\nabla_{\hat{\boldsymbol{h}}} \Gamma[k]\big|_{\hat{\boldsymbol{h}}=\hat{\boldsymbol{h}}[k+1]} = 0$. Then, we have

$$\begin{aligned}\nabla_{\hat{\boldsymbol{h}}} \Gamma[k] &= \sum_{j=0}^{k} \lambda^{k-j} \nabla_{\hat{\boldsymbol{h}}} \Lambda(e[j]) \\ &= \sum_{j=0}^{k} \lambda^{k-j} \frac{\partial \Lambda(e[j])}{\partial \psi(e[j])} \, \frac{\partial \psi(e[j])}{\partial e[j]} \, \nabla_{\hat{\boldsymbol{h}}} e[j] \\ &= \sum_{j=0}^{k} \lambda^{k-j} \frac{b\psi(e[j])}{b\varphi(e[j]) + 1} \, \frac{\partial \psi(e[j])}{\partial e[j]} \, (-\boldsymbol{s}[j]) \\ &= \sum_{j=0}^{k} \lambda^{k-j} \frac{b\frac{\psi(e[j])}{e[j]}}{b\varphi(e[j]) + 1} \, \frac{\partial \psi(e[j])}{\partial e[j]} \, (-\boldsymbol{s}[j]) e[j] \\ &= \sum_{j=0}^{k} \lambda^{k-j} \omega(e[j])(-\boldsymbol{s}[j])(r[j] - \boldsymbol{s}^H[j]\hat{\boldsymbol{h}}), \end{aligned} \tag{6.9}$$

where $\omega(e[j]) = \dfrac{b\frac{\psi(e[j])}{e[j]}}{b\varphi(e[j]) + 1} \, \dfrac{\partial \psi(e[j])}{\partial e[j]}$ is the weight of the j^{th} sample. The equation $\nabla_{\hat{\boldsymbol{h}}} \Gamma[k]\big|_{\hat{\boldsymbol{h}}=\hat{\boldsymbol{h}}[k+1]} = 0$ yields the following equation for $\hat{\boldsymbol{h}}[k+1]$,

$$\sum_{j=0}^{k} \lambda^{k-j} \omega(e[j]) \boldsymbol{s}[j] \boldsymbol{s}^H[j] \hat{\boldsymbol{h}}[k+1] = \sum_{j=0}^{k} \lambda^{k-j} \omega(e[j]) \boldsymbol{s}[j] r[j],$$

and [5],

$$\hat{\boldsymbol{h}}[k+1] = \boldsymbol{\Delta}^{-1}[k]\boldsymbol{\phi}[k],$$

where $\boldsymbol{\Delta}[k] = \sum_{j=0}^{k} \lambda^{k-j}\omega(e[j])\boldsymbol{s}[j]\boldsymbol{s}^T[j]$ and $\boldsymbol{\phi}[k] = \sum_{j=0}^{k} \lambda^{k-j}\omega(e[j])\boldsymbol{s}[j]r[j]$. Furthermore, we note that,

$$\boldsymbol{\Delta}[k] = \lambda\boldsymbol{\Delta}[k-1] + \omega(e[k])\boldsymbol{s}[k]\boldsymbol{s}^H[k], \tag{6.10}$$

and

$$\boldsymbol{\phi}[k] = \lambda\boldsymbol{\phi}[k-1] + \omega(e[k])\boldsymbol{s}[k]r[k]. \tag{6.11}$$

Thus, by using the matrix inversion lemma [4], $\boldsymbol{\Delta}^{-1}[k]$ can be calculated as follows,

$$\boldsymbol{\Delta}^{-1}[k] = \lambda^{-1}\left(\boldsymbol{\Delta}^{-1}[k-1] - \frac{1}{\frac{\lambda}{\omega(e[k])} + \boldsymbol{s}^H[k]\boldsymbol{\Delta}^{-1}[k-1]\boldsymbol{s}[k]}\ \boldsymbol{\Delta}^{-1}[k-1]\boldsymbol{s}[k]\boldsymbol{s}^H[k]\boldsymbol{\Delta}^{-1}[k-1]\right).$$

We next define $\boldsymbol{R}[k] \triangleq \boldsymbol{\Delta}^{-1}[k]$, and $\boldsymbol{g}[k] \triangleq \frac{\omega(e[k])\boldsymbol{R}[k-1]\boldsymbol{s}[k]}{\lambda+\omega(e[k])\boldsymbol{s}^H[k]\boldsymbol{R}[k-1]\boldsymbol{s}[k]}$. Then, we have

$$\boldsymbol{R}[k] = \lambda^{-1}(\boldsymbol{R}[k-1] - \boldsymbol{g}[k]\boldsymbol{s}^H[k]\boldsymbol{R}[k-1]). \tag{6.12}$$

Rearranging the terms in the definition of $\boldsymbol{g}[k]$ leads to,

$$\begin{aligned}\boldsymbol{g}[k] &= \frac{\omega(e[k])}{\lambda}\left[\boldsymbol{R}[k-1] - \boldsymbol{g}[k]\boldsymbol{s}^H[k]\boldsymbol{R}[k-1]\right]\boldsymbol{s}[k] \\ &= \omega(e[k])\boldsymbol{R}[k]\boldsymbol{s}[k].\end{aligned} \tag{6.13}$$

Substituting (6.11) and (6.12) in the expression $\hat{\boldsymbol{h}}[k+1] = \boldsymbol{R}[k]\boldsymbol{\psi}[k]$ and expanding it, results in

$$\begin{aligned}\hat{\boldsymbol{h}}[k+1] &= \lambda\boldsymbol{R}[k]\boldsymbol{\phi}[k-1] + \omega(e[k])\boldsymbol{R}[m]\boldsymbol{s}[k]r[k] \\ &= (\boldsymbol{R}[k-1] - \boldsymbol{g}[k]\boldsymbol{s}^H[k]\boldsymbol{R}[k-1])\boldsymbol{\phi}[k-1] + \boldsymbol{g}[k]r[k] \\ &= \hat{\boldsymbol{h}}[k] + \boldsymbol{g}[k](r[k] - \boldsymbol{s}^H[k]\hat{\boldsymbol{h}}[k]).\end{aligned} \tag{6.14}$$

Hence, the final second order updating algorithm is,

$$\begin{aligned}&e[k] = r[k] - \boldsymbol{s}^H[k]\hat{\boldsymbol{h}}[k], \\ &\boldsymbol{g}[k] = \frac{\omega(e[k])\boldsymbol{R}[k-1]\boldsymbol{s}[k]}{\lambda + \omega(e[k])\boldsymbol{s}^H[k]\boldsymbol{R}[k-1]\boldsymbol{s}[k]}, \\ &\hat{\boldsymbol{h}}[k+1] = \hat{\boldsymbol{h}}[k] + e[k]\boldsymbol{g}[k], \\ &\boldsymbol{R}[k] = \lambda^{-1}\left(\boldsymbol{R}[k-1] - \boldsymbol{g}[k]\boldsymbol{s}^H[k]\boldsymbol{R}[k-1]\right),\end{aligned} \tag{6.15}$$

where $\boldsymbol{R}[0] = \nu^{-1}\boldsymbol{I}$, and $0 < \nu \ll 1$. We point out that for the $\Theta(e[k]) = |e[k]|^2$, we have $\omega(e[k]) = \frac{2b\ |e[k]|^2}{b\ |e[k]|^2+1}$. However, according to (6.9) multiplying the $\omega(e[k])$ by a constant term does not affect the algorithm. Therefore, we use $\omega(e[k]) = \frac{|e[k]|^2}{b\ |e[k]|^2+1}$ in (6.15) to obtain the logarithmic cost recursive least squares (LCRLS) algorithm. Furthermore, we can readily extend our analysis to the LCRLA algorithm by using $\Theta(e[k]) = |e[k]|$. Here, we get $\omega(e[k]) = \frac{1}{b\ |e[k]|+1}$.

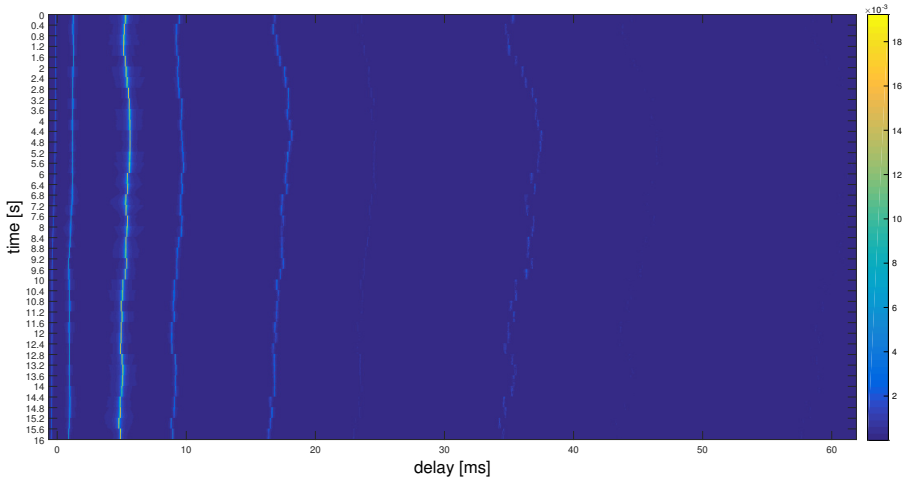


Figure 6.3: Time evolution of the magnitude baseband impulse response of the generated channel [195]

6.3 Simulation Results

6.3.1 Setup

In this section, we analyze the performance of our algorithm under a highly realistic underwater acoustic channel equalization scenario through a highly accurate modeling of the underwater channels introduced in [195]. We generate 60000 bits by repeating a Turyn sequence [194] (with 28 bit length). We then simulate the transmission of these bits over the UWA channel shown in Fig. 6.3. The simulation configurations and the parameters used for simulating the channel are given in the Table 6.1. Furthermore, we calculate the SNR from the baseband signal. In all of our experiments, we use a 10-tap NLMS equalizer, with the causal and anti-causal parts each of length 5. We use the step size $\epsilon = 0.01$ for all the algorithms. The length of the channel estimator is set to 11, since according to Fig. 6.3, $_{max} \approx 10$ ms and $T_s = 1$ ms (for the symbol rate = 1kHz). The results are averaged over 30 repetitions for the fairness of analysis, and show the superior performance of our algorithms over other methods. The performance parameters we use to compare the competing algorithms in our analysis are bit error rate (BER) and normalized MSE in several experiments.

Parameters	Values
Depth	100 m
Transmitter height	20 m
Receiver height	50 m
Distance between Tx and Rx	1 km
Carrier Frequency (f_c)	10.8 kHz
Bandwidth	1.6 kHz
Minimum frequency ($f_{\min}$)	10 kHz
Frequency resolution (d_f)	100 Hz
Time resolution (d_t)	10/16 ms
Symbol rate	1 kHz
Coherence time of the small scale variants (T_{SS})	60 s
Total duration of simulated channel (T_{total})	60 s

Table 6.1: The simulated channel configurations.

Algorithm	Cost function
SA	$\|e[m]\|$
LMS	$\|e[m]\|^2$
LCLMS	$\|e[m]\|^2 - \frac{1}{b}\ln(b\|e[m]\|^2 + 1)$
LCLMA	$\|e[m]\| - \frac{1}{b}\ln(b\|e[m]\| + 1)$
ILSE	$\frac{1}{\lambda}\cosh(\lambda e[m])$
RZALMP	$\|e[m]\|^j + \lambda\sum_{l=1}^{N}\log(\frac{\|\hat{h}[l]\|}{\delta} + 1)$
RLS	$\sum_{k=1}^{n}\lambda^{n-k}\|e[n]\|^2$
LCRLS	$\sum_{k=1}^{n}\lambda^{n-k}\|e[k]\|^2 - \frac{1}{b}\ln(b\sum_{i=1}^{m}\lambda^{n-k}\|e[k]\|^2 + 1)$
LCRLA	$\sum_{k=1}^{n}\lambda^{n-k}\|e[k]\| - \frac{1}{b}\ln(b\sum_{i=1}^{m}\lambda^{m-i}\|e[i]\| + 1)$
l_0-RLS	$\sum_{k=0}^{m}\lambda^{m-k}\|e[k]\|^2 + \xi\|\hat{\mathbf{h}}[n]\|_0$, where $\|\hat{\mathbf{h}}[n]\|_0 \approx \sum_{k=0}^{K-1}(1 - \exp(-\eta\|\hat{h}[k]\|))$

Table 6.2: Cost function of each proposed and competing algorithms used in the analysis throughout this chapter

6.3.2 Experimental Results and Analysis

We compare the performance of the proposed algorithms under two settings, i.e., different impulsive noise probabilities under fixed AWGN noise and different SNR values with fixed impulsive noise factor. In the first experiment, we use different first and second order logarithmic cost based channel estimation algorithms, i.e, LCLMS, LCLMA, LCRLA and LCRLS. Here, we use a mixture of AWGN and impulsive noise model with 5% probability. To set a higher amplitude of impulsive noise, we choose the variance as 10^4 times that of the Gaussian noise. We plot the BER versus SNR in the range $\{0, 30\}$ dB as shown in Fig. 6.4 and Fig. 6.5.

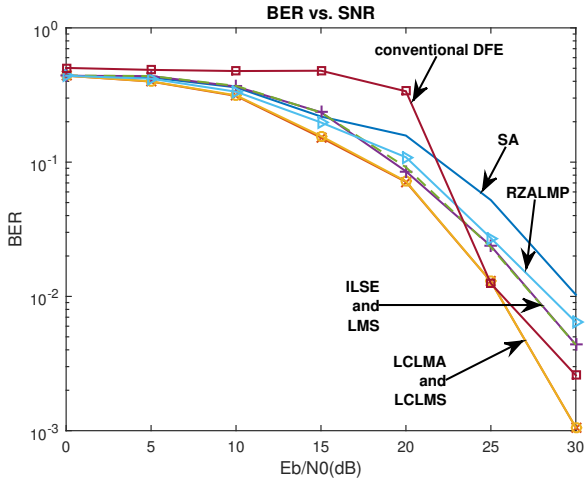


Figure 6.4: BER analysis of the first order methods, in a channel with 5% impulsive noise.

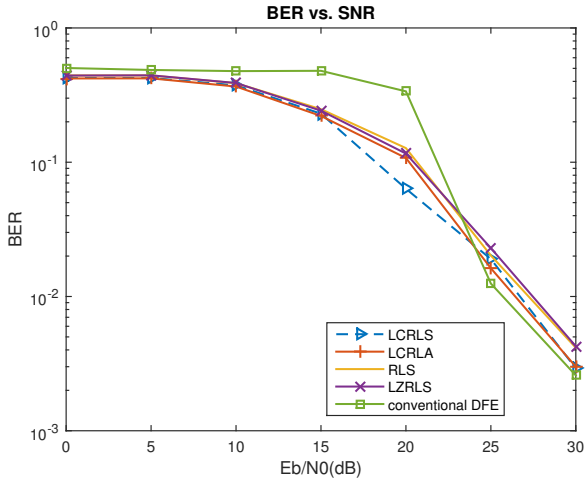


Figure 6.5: BER analysis of the second order methods, in a channel with 5% impulsive noise.

We also compare the BER and MSE performance of the proposed estimation algorithms with the state-of-the-art algorithms, e.g., SA, LMS, RLS, improved least sum of exponentials (ILSE) [185], the conventional DFE, reweighted zero-attracting least mean p-power (RZALMP) [186], and l_0-RLS [183]. Furthermore, we give the cost function of each of the proposed as well as competing algorithms in Table 6.2.

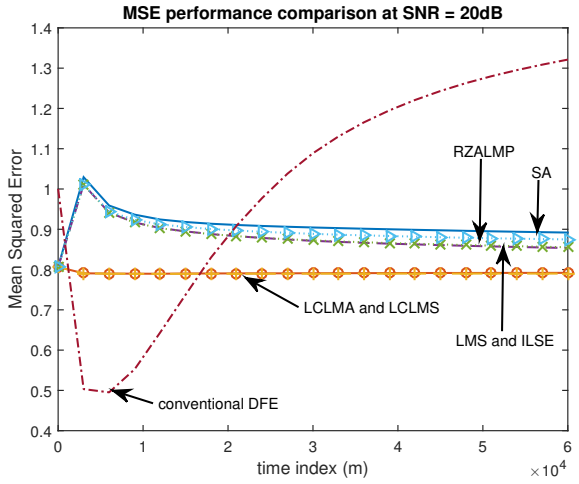


Figure 6.6: MSE analysis in a 20 dB SNR and 5% impulsive noise channel. The proposed first order methods, LCLMS and LCLMA outperform the conventional algorithms in terms of convergence rate

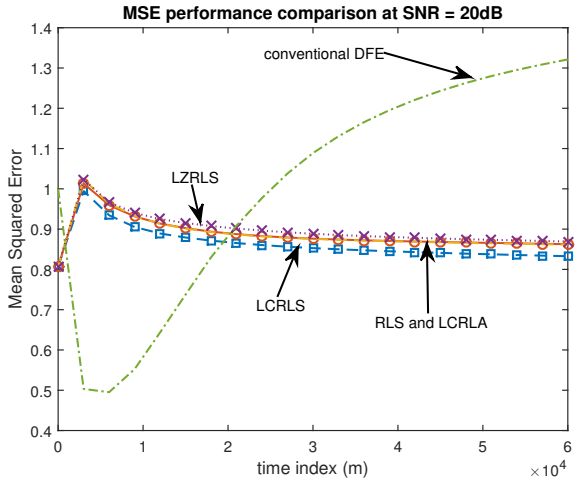


Figure 6.7: MSE analysis in a 20 dB SNR and 5% impulsive noise channel. The proposed second order method, LCRLS outperforms the conventional algorithms in terms of convergence rate

We compare the performance of competing algorithms and show the significant performance gain of the proposed first order algorithms (Figs. 6.4) and second order algorithms (6.5). In the next series of results, i.e., Figs. 6.6 and 6.7, we demonstrate the MSE performance gain of the proposed methods at fixed SNR = 20dB. We show similar results in Fig. 6.8 and 6.9 for shorter time intervals.

Again, the MSE results in Figs. 6.6 and 6.7, show the strong capability of our algorithms in terms of tracking the channel variations in impulsive noise

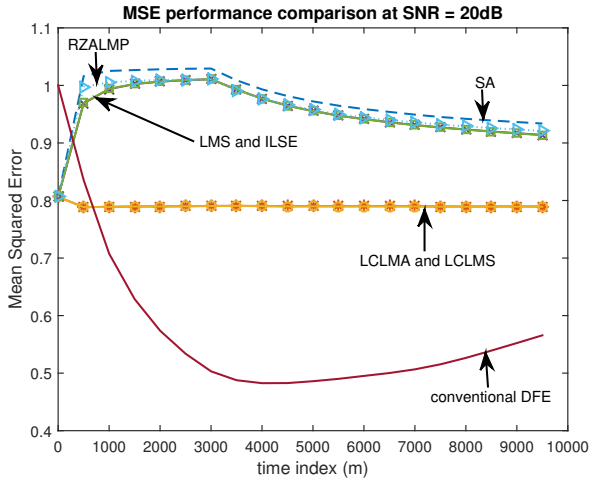


Figure 6.8: MSE analysis of different first order methods.

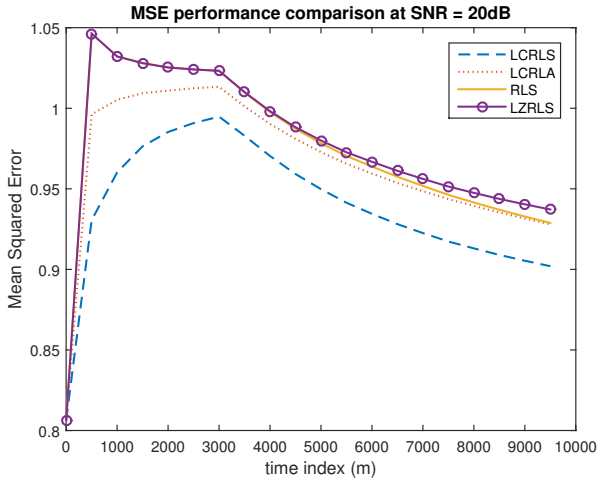


Figure 6.9: MSE analysis of different second order methods.

environment, that results in a significant performance gain compared to the other conventional estimation methods.

In the next series of experiments, we analyze the effect of the impulse probability, ν_i, on the performance of the proposed algorithms. We keep the SNR fixed at 20dB and plot the BER versus probabilities of impulsive noise components in Figs 6.10 and 6.11. We can observe that the proposed algorithm specifically outperform the competing algorithm in the adverse conditions. We can deduce that for a highly non-stationary communication channels affected by impulsive noise, the proposed algorithms performs extremely well. Therefore, the proposed algorithms are well suited for the estimation real life UWA channels.

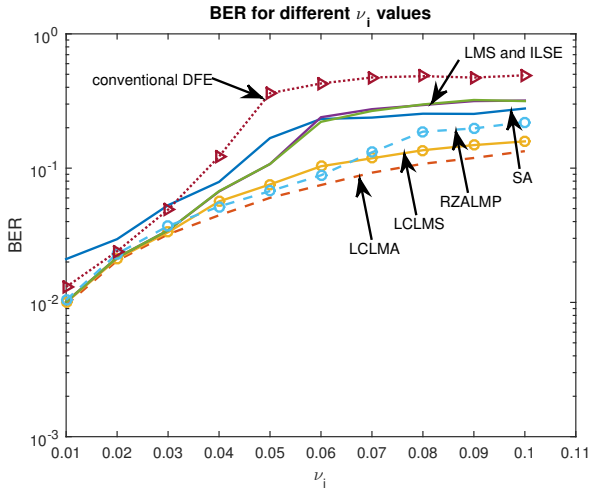


Figure 6.10: BER versus the probabilities of impulsive noise at 20 dB SNR, for the first order algorithms.

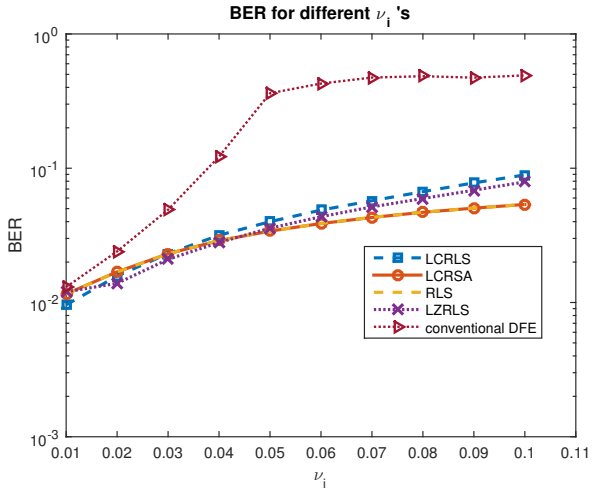


Figure 6.11: BER versus the probabilities of impulsive noise at 20 dB SNR, for the second order algorithms.

6.4 Discussion

In this chapter, we investigate nonlinear modeling of the highly non-stationary underwater acoustic (UWA) channel. We use logarithmic cost function to track the variations in a non-stationary channel and present a novel family of adaptive algorithms for the UWA channel estimation. Specifically, we introduce robust, first and second order estimation algorithms for the detrimental UWA channel. We achieve the stability of lower power cost functions with similar computational complexity. We follow a decision feedback equalization framework while

implementing the proposed methods. We demonstrate the performance of the proposed algorithms over state-of-the-art channel estimation algorithms, through several realistic simulations and experiments.

Chapter 7

Conclusion

Big data is a constantly evolving field that draws enormous interest from data science, machine learning and adaptive signal processing literatures. With the advent and development of internet and distributed networked application such as social networks, databases, e-commerce and threat intelligence, the amount of data that is generated is almost inconceivable. We need to intelligently analyze big data for insights that lead to better decisions. We investigate the challenges in the field of big data analytics that are usually defined by five V's, i.e., velocity, veracity, volume, variety and value. In principle, we focus on the analysis and learning in a data driven manner where the data arrives in high speed, have non-stationary structure and consists of features that may have different importance (" value "). Furthermore, we investigate the sequential and contextual meaning of the data by exploiting the inherent structure.

We investigate the online learning approaches, i.e., where we process the data as soon as it's available rather than working on a "batch", in a nonlinear and non-stationary setting. We propose novel algorithms for regression, i.e., when the desired output is continuous, classification, i.e., when the desired output is discrete, and estimation of unknown parameter such as the coefficients of adaptive filters that describe an unknown model. We focus on the challenges within the input data, e.g, structural variations, curse of dimensionality, corruptions, noise and missing information, as well as the complexities of the relationship between input and output.

We study online nonlinear regression in a high dimensional setting, i.e., $D \gg 1$. We assume a time varying manifold structure having a curvature with intrinsic dimension $d_{\text{intrinsic}} \ll D$. We propose novel algorithm called adaptive hierarchical trees (AHT) that dynamically learn the underlying manifold structure of the

input space, as well the nonlinearity by using a piecewise model. We specifically use context tree weighting method to model the nonlinearity by a piecewise linear model. We successfully argue the superiority of proposed algorithm over state-of-the-art algorithms in terms of performance and computational efficiency, by using mathematical justification and experimental results.

Next we consider applications of online nonlinear learning in a big data perspective using deep learning approaches. We base our analysis on two real life applications, namely network intrusion detection systems (IDS) and churn prediction. We thoroughly analyze the classic IDS approaches with a view of advancements in internet applications and novel intrusion methods. We devise a multi-level learning of the network traffic that is independent of the application and scale of the network, and works equally well to detect previously unknown attacks. Furthermore, our intrusion detection techniques directly applies on the application layer payloads of the network and requires minimal amount of preprocessing. We use a natural language processing (NLP) approach to analyze the payloads that results is a highly efficient and robust IDS.

Furthermore, we use ensemble learning and boosting approaches that are online and adaptive to the variation in the underlying system parameters. We develop a novel online boosting method for regression and classification that is independent of the statistical distribution of the data. Specifically, we use mixture of experts methods that works on combination of weak learners (WL), where the number of WL, there combination weights, individual parameters and update, all are deterministic and learn from the current data without a priori knowledge.

Finally, we use a deterministic approach to learn the parameters of UWA channel by assuming a highly realistic setting. The UWA channel suffers from high amplitude but short and less frequent noise. The modeling and estimation of such channel is challenging if classic gradient descent algorithms are used where the cost function if usually considered as powers of the error. Therefore, we propose to add a logarithmic term to the cost function as a correction that in result achieves better trade-off between stability and convergence during training. The proposed estimation techniques significantly outperform the state-of-the-art techniques in literature and can be seen by the results of a number of experiments.

Bibliography

[1] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[2] V. Vovk, "Aggregating strategies," *COLT*, pp. 371–383, 1990.

[3] ——, "Competitive on-line linear regression," *Advances in Neural Information Processing Systems*, pp. 364–370, 1998.

[4] A. H. Sayed, *Fundamentals of Adaptive Filtering*. NJ: John Wiley & Sons, 2003.

[5] S. O. Haykin, *Adaptive Filter Theory*. Prentice Hall, 2002.

[6] 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.

[7] 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.

[8] 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.

[9] I. G. Ali and Y.-T. Chen, "Design quality and robustness with neural networks," *IEEE Transactions on Neural Networks*, vol. 10, no. 6, pp. 1518–1527, 1999.

[10] L. Rutkowski, "Generalized regression neural networks in time-varying environment," *IEEE Transactions on Neural Networks*, vol. 15, no. 3, pp. 576 - 596, 2004.

[11] R. Setiono, W. K. Leow, and J. M. Zurada, "Extraction of rules from artificial neural networks for nonlinear regression," *IEEE Transactions on Neural Networks*, vol. 13, no. 3, pp. 564 - 577, 2002.

[12] 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.

[13] R. Gribonval, "From projection pursuit and CART to adaptive discriminant analysis?" *IEEE Transactions on Neural Networks*, vol. 16, no. 3, pp. 522–532, 2005.

[14] S. Bengio and Y. Bengio, "Taking on the curse of dimensionality in joint distributions using neural networks," *IEEE Transactions on Neural Networks*, vol. 11, no. 3, pp. 550–557, 2000.

[15] N. C. Bianchi, P. M. Long, and M. K. Warmuth, "Worst-case quadratic loss bounds for prediction using linear functions and gradient descent," *IEEE Transactions on Neural Networks*, vol. 7, no. 3, pp. 604–619, 1996.

[16] K. Muller, S. Mika, G. Ratsch, K. Tsuda, and B. Scholkopf, "An introduction to kernel-based learning algorithms," *IEEE Transactions on Neural Networks*, vol. 12, no. 2, pp. 181–201, 2002.

[17] X. Wu, X. Zhu, and G.-Q. Wu, W. Ding, "Data mining with big data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 1, pp. 97–107, 2014.

[18] H. Liu and L. Yu, "Toward integrating feature selection algorithms for classification and clustering," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 4, pp. 491–502, 2005.

[19] Q. Song, J. Ni, and G. Wang, "A Fast Clustering-Based Feature Subset Selection Algorithm for High-Dimensional Data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 1, pp. 1–14, 2012.

[20] O. Egecioglu, H. Ferhatosmanoglu, and U. Ogras, "Dimensionality reduction and similarity computation by inner-product approximations," *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 6, pp. 714–726, 2004.

[21] C. Kreibich, M. Handley, and V. Paxson, "Network intrusion detection: Evasion, traffic normalization, and end-to-end protocol semantics," in *Proc. USENIX Security Symposium*, vol. 2001, 2001.

[22] P. Laskov, P. Düssel, C. Schäfer, and K. Rieck, "Learning Intrusion Detection: Supervised or Unsupervised?," in *Image Analysis and Processing – ICIAP 2005: 13th International Conference, Cagliari, Italy, September 6-8, 2005. Proceedings* (F. Roli and S. Vitulano, eds.), pp. 50–57, Berlin, Heidelberg: Springer Berlin Heidelberg, 2005.

[23] B. Mukherjee, L. T. Heberlein, and K. N. Levitt, "Network intrusion detection," *IEEE network*, vol. 8, no. 3, pp. 26–41, 1994.

[24] L. Shen and E. C. Tan, "Dimension redunction-based penalized logistic regression for cancer classification using microarray data", *IEEE Transactions on Computational Biology and Bioimformatics*, vol. 2, no. 2, pp. 166–175, 2005.

[25] J. Nilsson, F. Sha, and M. Jordan, "Regression on manifolds using kernel dimension reduction", *Proceedings of the 24th International Conference on Machine Learning (ICML)*, pp. 697–704, 2007.

[26] A. Lakhina , M. Crovella, and C. Diot, "Diagnosing network-wide traffic anomalies," *Proceedings of ACM SIGCOMM*, 2004.

[27] Y. Xie, J. Huang, and R. Willett, "Multiscale online tracking of manifolds," *IEEE Statistical Signal Processing Workshop (SSP)*, 2012

[28] Y. Xie, J. Huang, and R. Willett, "Change-point detection for high-dimensional time series with missing data," *IEEE Journal of Selected Topics in Signal Processing*, vol. 7, no. 1, pp. 12–27, 2013.

[29] K.-C. Lee and D. Kriegman, "Online learning of probabilistic appearance manifolds for video-based recognition and tracking," *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 1, pp. 852–859, June 2005.

[30] R. T. Collins, A. J. Lipton, and T. Kanade, "Introduction to the special section on video surveillance," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 8, pp. 745–746, 2000.

[31] Richard. E. Bellman, *Adaptive Control Processes: A Guided Tour*. Princeton University Press, 1961.

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

[33] S. Kpotufe and S. Dasgupta, "A tree-based regressor that adapts to intrinsic dimension, " *Journal of Computer and System Sciences*, vol. 78, no. 5, pp. 1496–1515, 2011.

[34] N. Cristianini and J. Shawe-Taylor, *Support Vector Machines and Other Kernal-Based Learning Methods*. Cambridge University Press, 2000.

[35] I. T. Jolliffe, *Principal Component Analysis*. Springer, 2002.

[36] S. T. Roweis and L. K. Saul, "Nonlinear dimensionality reduction by locally linear embedding," *Science, New Series*, vol. 290, No. 5500, pp. 2323–2326, 2000.

[37] J. B. Tenenbaum, V. de Silva and J. C. Langford, "A global geometric framework for nonlinear dimensionality reduction," *Science, 290,2319*, 2000.

[38] Y. Xie and R. Willett, "Online logistic regression on manifolds," *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2013.

[39] J. M. Lee, *Riemannian Manifolds: An Introduction to Curvature*. Springer, 1997.

[40] J. Jost, *Riemannian Geometry and Geometric Analysis*. Springer, 2008.

[41] F. M. J. Willems,"The context-tree weighting method: extensions," *IEEE Transactions on Information Theory*, vol. 44, no. 2, pp. 792–798, 1998.

[42] 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.

[43] J. Lafferty, A. McCallum, and F. C. Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," 2001.

[44] C. Wressnegger, G. Schwenk, D. Arp, and K. Rieck, "A Close Look on N-grams in Intrusion Detection: Anomaly Detection vs. Classification," in *Proceedings of the 2013 ACM Workshop on Artificial Intelligence and Security*, AISec '13, (New York, NY, USA), pp. 67–76, ACM, 2013.

[45] B. Pang, L. Lee, and S. Vaithyanathan, "Thumbs up?: sentiment classification using machine learning techniques," in *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, pp. 79–86, Association for Computational Linguistics, 2002.

[46] G. Sidorov, F. Velasquez, E. Stamatatos, A. Gelbukh, and L. Chanona-Hernández, "Syntactic n-grams as machine learning features for natural language processing," *Expert Systems with Applications*, vol. 41, no. 3, pp. 853–860, 2014.

[47] A. Lazarevic, L. Ertoz, V. Kumar, A. Ozgur, and J. Srivastava, "A comparative study of anomaly detection schemes in network intrusion detection," in *Proceedings of the 2003 SIAM International Conference on Data Mining*, pp. 25–36, SIAM, 2003.

[48] K. Rieck, "Machine learning for application-layer intrusion detection," 2009.

[49] T. H. Ptacek and T. N. Newsham, "Insertion, evasion, and denial of service: Eluding network intrusion detection," tech. rep., SECURE NETWORKS INC CALGARY ALBERTA, 1998.

[50] A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani, "Toward developing a systematic approach to generate benchmark datasets for intrusion detection," *Computers & Security*, vol. 31, pp. 357–374, May 2012.

[51] L. R. Medsker and L. C. Jain, "Recurrent neural networks," *Design and Applications*, vol. 5, 2001.

[52] R. Setiono, B. Baesens, and C. Mues, "Recursive neural network rule extraction for data with mixed attributes," *IEEE Transactions on Neural Networks*, vol. 19, no. 2, pp. 299–307, 2008.

[53] Y. LeCun and Y. Bengio, "Convolutional networks for images, speech, and time series," *The handbook of brain theory and neural networks*, vol. 3361, no. 10, p. 1995, 1995.

[54] S. Axelsson, "Intrusion detection systems: A survey and taxonomy," tech. rep., Technical report, 2000.

[55] N. Hoque, M. H. Bhuyan, R. C. Baishya, D. K. Bhattacharyya, and J. K. Kalita, "Network attacks: Taxonomy, tools and systems," *Journal of Network and Computer Applications*, vol. 40, pp. 307–324, 2014.

[56] F. Anjum, D. Subhadrabandhu, and S. Sarkar, "Signature based intrusion detection for wireless ad-hoc networks: A comparative study of various routing protocols," vol. 3, pp. 2152–2156, IEEE, 2003.

[57] P. Garcia-Teodoro, J. Diaz-Verdejo, G. Maciá-Fernández, and E. Vázquez, "Anomaly-based network intrusion detection: Techniques, systems and challenges," *computers & security*, vol. 28, no. 1, pp. 18–28, 2009.

[58] F. Khan and S. S. Kozat, "Sequential churn prediction and analysis of cellular network users #x2014; A multi-class, multi-label perspective," in *2017 25th Signal Processing and Communications Applications Conference (SIU)*, pp. 1–4, May 2017.

[59] F. Khan, I. Delibalta, and S. S. Kozat, "Online churn detection on high dimensional cellular data using adaptive hierarchical trees," in *2016 24th European Signal Processing Conference (EUSIPCO)*, pp. 2275–2279, Aug. 2016.

[60] K. Kim and J. Lee, "Sequential manifold learning for efficient churn prediction," *Expert Systems with Applications*, vol. 39, no. 18, pp. 13328–13337, 2012.

[61] S. Pan, J. Wu, X. Zhu, G. Long, and C. Zhang, "Boosting for graph classification with universum," *Knowledge and Information Systems*, vol. 50, no. 1, pp. 53–77, 2017. [Online]. Available: http://dx.doi.org/10.1007/s10115-016-0934-z

[62] E. Bauer and R. Kohavi, "An empirical comparison of voting classification algorithms: Bagging, boosting, and variants," *Machine Learning*, vol. 36, no. 1, pp. 105–139, 1999. [Online]. Available: http://dx.doi.org/10.1023/A:1007515423169

[63] T. G. Dietterich, "An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization," *Machine Learning*, vol. 40, no. 2, pp. 139–157, 2000. [Online]. Available: http://dx.doi.org/10.1023/A:1007607513941

[64] A. Habrard, J.-P. Peyrache, and M. Sebban, "A new boosting algorithm for provably accurate unsupervised domain adaptation," *Knowledge and Information Systems*, vol. 47, no. 1, pp. 45–73, 2016. [Online]. Available: http://dx.doi.org/10.1007/s10115-015-0839-2

[65] B. X. Wang and N. Japkowicz, "Boosting support vector machines for imbalanced data sets," *Knowledge and Information Systems*, vol. 25, no. 1, pp. 1–20, 2010. [Online]. Available: http://dx.doi.org/10.1007/s10115-009-0198-y

[66] K. Dela Rosa, V. Metsis, and V. Athitsos, "Boosted ranking models: a unifying framework for ranking predictions," *Knowledge and Information Systems*, vol. 30, no. 3, pp. 543–568, 2012. [Online]. Available: http://dx.doi.org/10.1007/s10115-011-0390-8

[67] S. Shalev-Shwartz and Y. Singer, "On the equivalence of weak learnability and linear separability: new relaxations and efficient boosting algorithms," *Machine Learning*, vol. 80, no. 2, pp. 141–163, 2010. [Online]. Available: http://dx.doi.org/10.1007/s10994-010-5173-z

[68] E. Cesario, F. Folino, A. Locane, G. Manco, and R. Ortale, "Boosting text segmentation via progressive classification," *Knowledge and Information Systems*, vol. 15, no. 3, pp. 285–320, May 2008. [Online]. Available: http://dx.doi.org/10.1007/s10115-007-0085-3

[69] M. J. Hosseini, A. Gholipour, and H. Beigy, "An ensemble of cluster-based classifiers for semi-supervised classification of non-stationary data streams," *Knowledge and Information Systems*, vol. 46, no. 3, pp. 567–597, 2016. [Online]. Available: http://dx.doi.org/10.1007/s10115-015-0837-4

[70] C. Preisach and L. Schmidt-Thieme, "Ensembles of relational classifiers," *Knowledge and Information Systems*, vol. 14, no. 3, pp. 249–272, 2008. [Online]. Available: http://dx.doi.org/10.1007/s10115-007-0093-3

[71] A. L. Prodromidis and S. J. Stolfo, "Cost complexity-based pruning of ensemble classifiers," *Knowledge and Information Systems*, vol. 3, no. 4, pp. 449–469, 2001. [Online]. Available: http://dx.doi.org/10.1007/PL00011678

[72] Y. Kim and J. Kim, "Convex hull ensemble machine for regression and classification," *Knowledge and Information Systems*, vol. 6, no. 6, pp. 645–663, 2004. [Online]. Available: http://dx.doi.org/10.1007/s10115-003-0116-7

[73] S. Pan, Y. Zhang, and X. Li, "Dynamic classifier ensemble for positive unlabeled text stream classification," *Knowledge and Information Systems*, vol. 33, no. 2, pp. 267–287, 2012. [Online]. Available: http://dx.doi.org/10.1007/s10115-011-0469-2

[74] A. Fern and R. Givan, "Online ensemble learning: An empirical study," *Machine Learning*, vol. 53, no. 1, pp. 71–109, 2003. [Online]. Available: http://dx.doi.org/10.1023/A:1025619426553

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

[76] Y. Freund, "An adaptive version of the boost by majority algorithm," *Machine Learning*, vol. 43, no. 3, pp. 293–318, 2001. [Online]. Available: http://dx.doi.org/10.1023/A:1010852229904

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

[78] S. Mannor and R. Meir, "On the existence of linear weak learners and applications to boosting," *Machine Learning*, vol. 48, no. 1, pp. 219–251, 2002. [Online]. Available: http://dx.doi.org/10.1023/A:1013959922467

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

[80] Q. Wu, X. Zhou, Y. Yan, H. Wu, and H. Min, "Online transfer learning by leveraging multiple source domains," *Knowledge and Information Systems*, pp. 1–21, 2017. [Online]. Available: http://dx.doi.org/10.1007/s10115-016-1021-1

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

[82] S. Ben-David, E. Kushilevitz, and Y. Mansour, "Online learning versus offline learning," *Machine Learning*, vol. 29, no. 1, pp. 45–63, 1997. [Online]. Available: http://dx.doi.org/10.1023/A:1007465907571

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

[84] J. Arenas-Garcia, L. A. Azpicueta-Ruiz, M. T. M. Silva, V. H. Nascimento, and A. H. Sayed, "Combinations of adaptive filters: Performance and convergence properties," *IEEE Signal Processing Magazine*, vol. 33, no. 1, pp. 120–140, Jan 2016.

[85] M. K. Banavar, J. J. Zhang, B. Chakraborty, H. Kwon, Y. Li, H. Jiang, A. Spanias, C. Tepedelenlioglu, C. Chakrabarti, A. Papandreou-Suppappola, An overview of recent advances on distributed and agile sensing algorithms and implementation, Digital Signal Processing 39 (2015) 1–14.

[86] A. Gunes, M. B. Guldogan, Joint underwater target detection and tracking with the bernoulli filter using an acoustic vector sensor, Digital Signal Processing 48 (2016) 246–258.

[87] W. Zhu, M. Zhu, Y. Wu, B. Yang, L. Xu, X. Fu, and F. Pan, "Signal processing in underwater acoustic communication system for manned deep submersible "Jiaolong"," *The Journal of the Acoustical Society of America*, vol. 131, pp. 3238–3238, Apr. 2012.

[88] D. B. Kilfoyle, A. B. Baggeroer, The state of the art in underwater acoustic telemetry, Oceanic Engineering, IEEE Journal of 25 (1) (2000) 4–27.

[89] M. Stojanovic, J. Preisig, Underwater acoustic communication channels: Propagation models and statistical characterization, IEEE Communications Magazine 47 (1) (2009) 84–89.

[90] D. Peng, Y. Xiang, H. Trinh, Z. Man, Adaptive blind equalization of time-varying simo systems driven by qpsk inputs, Digit. Signal Process. 23 (1) (2013) 268–274.

[91] J. G. Proakis, Digital Communications, McGraw-Hill, 1995.

[92] A. Song, M. Badiey, Generalized equalization for underwater acoustic communications, Proceedings of OCEANS 2005 MTS/IEEE (2005) 1522 – 1527.

[93] F. Khan, D. Kari, I. A. Karatepe, and S. S. Kozat, "Universal Nonlinear Regression on High Dimensional Data Using Adaptive Hierarchical Trees," *IEEE Transactions on Big Data*, vol. 2, no. 2, pp. 175–188, 2016.

[94] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with LSTM," 1999.

[95] A. H. Lashkari, G. D. Gil, M. S. I. Mamun, and A. A. Ghorbani, "Characterization of Tor Traffic using Time based Features," pp. 253–262, Nov. 2017.

[96] D. L. Streiner and J. Cairney, "What's under the ROC? An introduction to receiver operating characteristics curves," *The Canadian Journal of Psychiatry*, vol. 52, no. 2, pp. 121–128, 2007.

[97] J. A. Hanley and B. J. McNeil, "The meaning and use of the area under a receiver operating characteristic (ROC) curve.," *Radiology*, vol. 143, no. 1, pp. 29–36, 1982.

[98] 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.

[99] F. M. J. Willems,"Coding for a binary independent piecewise-identi- cally-distributed source," *IEEE Transactions on Information Theory*, vol. 42, no. 6, pp. 2210–2217, 1996.

[100] N. D. Vanli, and S. S Kozat, "A Comprehensive Approach to Universal Piecewise Nonlinear Regression Based on Trees," *IEEE Transactions on Signal Processing*, vol. 62, no. 20, pp. 5471–5486 , 2014.

[101] W. C. Stirling, and M. Morf, "An adaptive empirical Bayes decision-directed detector," *21st IEEE Conference on Decision and Control*, December 1982.

[102] S. L. France, J. Douglas Carroll, and H. Xiong, "Distance metrics for high dimensional nearest neighborhood recovery: Compression and normalization," *Information Sciences*, vol. 184, no. 1, pp. 92–110, Feb. 2012.

[103] Y. Chi, Y. C. Eldar, and R. Calderbank, "PETReLS: Subspace estimation and tracking from partial observations," *Int. Conf. on Acoustic, Speech, and Sig. Processing*, March 2012.

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

[105] K. Fernandes, P. Vinagre, and P. Cortez, "A proactive intelligent decision support system for predicting the popularity of online news," *Portuguese Conference on Artificial Intelligence. Springer, Cham*, 2015.

[106] S. D. Bay, D. Kibler, M. J. Pazzani, and P. Smyth, "The UCI KDD archive of large data sets for data mining research and experimentation," *ACM SIGKDD Explorations Newsletter*, vol. 2, no. 2, pp. 81–85, 2000.

[107] J. Alcala-Fdez, A. Fernandez, 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, 2011.

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

[109] K. Rieck and P. Laskov, "Language models for detection of unknown attacks in network traffic," *Journal in Computer Virology*, vol. 2, no. 4, pp. 243–256, 2007.

[110] C. Kruegel and T. Toth, "Using decision trees to improve signature-based intrusion detection," pp. 173–191, Springer, 2003.

[111] S. Northcutt and J. Novak, *Network intrusion detection*. Sams Publishing, 2002.

[112] S. D. Shanklin, T. E. Bernhard, and G. S. Lathem, "Intrusion detection signature analysis using regular expressions and logical operators."

[113] K. Wang and S. J. Stolfo, "Anomalous payload-based network intrusion detection," in *RAID*, vol. 4, pp. 203–222, Springer, 2004.

[114] S. J. Stolfo, W. Fan, W. Lee, A. Prodromidis, and P. K. Chan, "Cost-based modeling for fraud and intrusion detection: Results from the JAM project," in *DARPA Information Survivability Conference and Exposition, 2000. DISCEX'00. Proceedings*, vol. 2, pp. 130–144, IEEE, 2000.

[115] V. Engen, J. Vincent, and K. Phalp, "Exploring discrepancies in findings obtained with the KDD Cup'99 data set," *Intelligent Data Analysis*, vol. 15, no. 2, pp. 251–276, 2011.

[116] V. Engen, *Machine learning for network based intrusion detection: an investigation into discrepancies in findings with the KDD cup'99 data set and multi-objective evolution of neural network classifier ensembles from imbalanced data.* PhD thesis, Bournemouth University, 2010.

[117] R. Socher, C. C. Lin, C. Manning, and A. Y. Ng, "Parsing natural scenes and natural language with recursive neural networks," in *Proceedings of the 28th international conference on machine learning (ICML-11)*, pp. 129–136, 2011.

[118] H. B. Demuth, M. H. Beale, O. De Jess, and M. T. Hagan, *Neural network design*. Martin Hagan, 2014.

[119] S. K. Sahu, S. Sarangi, and S. K. Jena, "A detail analysis on intrusion detection datasets," in *Advance Computing Conference (IACC), 2014 IEEE International*, pp. 1348–1353, IEEE, 2014.

[120] H. Debar, M. Dacier, and A. Wespi, "Towards a taxonomy of intrusion-detection systems," *Computer Networks*, vol. 31, no. 8, pp. 805–822, 1999.

[121] R. Pang and V. Paxson, "A High-level Programming Environment for Packet Trace Anonymization and Transformation," in *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '03, (New York, NY, USA), pp. 339–351, ACM, 2003.

[122] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "LSTM: A search space odyssey," *IEEE transactions on neural networks and learning systems*, 2017.

[123] Y. Tang, Y. Huang, Z. Wu, H. Meng, M. Xu, and L. Cai, "Question detection from acoustic features using recurrent neural network with gated recurrent unit," in *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*, pp. 6125–6129, IEEE, 2016.

[124] D. Tang, B. Qin, and T. Liu, "Document Modeling with Gated Recurrent Neural Network for Sentiment Classification.," in *EMNLP*, pp. 1422–1432, 2015.

[125] R. Jozefowicz, W. Zaremba, and I. Sutskever, "An empirical exploration of recurrent network architectures," in *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pp. 2342–2350, 2015.

[126] N. Kalchbrenner, E. Grefenstette, and P. Blunsom, "A convolutional neural network for modelling sentences," *arXiv preprint arXiv:1404.2188*, 2014.

[127] "Flowmeter | Datasets | Research | Canadian Institute for Cybersecurity | UNB."

[128] W. Huang, Y. Qiao, and X. Tang, "Robust scene text detection with convolution neural network induced mser trees," in *European Conference on Computer Vision*, pp. 497–511, Springer, 2014.

[129] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[130] M. Sundermeyer, R. Schlüter, and H. Ney, "LSTM neural networks for language modeling," in *Thirteenth Annual Conference of the International Speech Communication Association*, 2012.

A. H. Lashkari, G. D. Gil, M. S. I. Mamun, and A. A. Ghorbani, "Characterization of Tor Traffic using Time based Features," pp. 253–262, Nov. 2017.

[131] W. Reinartz, M. Krafft, and W. D. Hoyer, "The Customer Relationship Management Process: Its Measurement and Impact on Performance," *Journal of Marketing Research*, vol. 41, pp. 293–305, Aug. 2004.

[132] F. F. Reichheld and J. W. Sasser, "Zero defections: quality comes to services.," *Harvard business review*, vol. 68, no. 5, pp. 105–111, 1990.

[133] A. Prinzie and D. Van den Poel, "Incorporating Sequential Information into Traditional Classification Models by Using an Element/Position-sensitive SAM," *Decis. Support Syst.*, vol. 42, pp. 508–526, Nov. 2006.

[134] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Comput.*, vol. 9, pp. 1735–1780, Nov. 1997.

[135] J.-H. Ahn, S.-P. Han, and Y.-S. Lee, "Customer churn analysis: Churn determinants and mediation effects of partial defection in the Korean mobile telecommunications service industry," *Telecommunications Policy*, vol. 30, pp. 552–568, Nov. 2006.

[136] A. Niculescu-Mizil, C. Perlich, G. Swirszcz, V. Sindhwani, Y. Liu, P. Melville, D. Wang, J. Xiao, J. Hu, M. Singh, W. X. Shang, and Y. F. Zhu, "Winning the KDD Cup Orange Challenge with Ensemble Selection," in *Proceedings of the 2009 International Conference on KDD-Cup 2009 - Volume 7*, KDD-CUP'09, (Paris, France), pp. 23–34, JMLR.org, 2009.

[137] J. H. Friedman, "Greedy Function Approximation: A Gradient Boosting Machine," *The Annals of Statistics*, vol. 29, no. 5, pp. 1189–1232, 2001.

[138] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[139] N. Duffy and D. Helmbold, "Boosting methods for regression," *Machine Learning*, vol. 47, no. 2, pp. 153–200, 2002. [Online]. Available: http://dx.doi.org/10.1023/A:1013685603443

[140] C. K. Reddy and J.-H. Park, "Multi-resolution boosting for classification and regression problems," *Knowledge and Information Systems*, vol. 29, no. 2, pp. 435–456, 2011. [Online]. Available: http://dx.doi.org/10.1007/s10115-010-0358-0

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

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

[143] ——, "A gradient boosting approach to the kaggle load forecasting competition," *International Journal of Forecasting*, pp. 1–19, 2013.

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

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

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

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

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

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

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

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

[152] Y. Freund, "Boosting a weak learning algorithm by majority," *Inf. Comput.*, vol. 121, no. 2, pp. 256–285, Sep. 1995.

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

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

[155] 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.

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

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

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

[159] 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.

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

[161] E. Hazan, A. Agarwal, and S. Kale, "Logarithmic regret algorithms for online convex optimization," *Mach. Learn.*, vol. 69, no. 2-3, pp. 169–192, Dec. 2007. [Online]. Available: http://dx.doi.org/10.1007/s10994-007-5016-8

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

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

[164] M. Lichman, "UCI machine learning repository," 2013. [Online]. Available: http://archive.ics.uci.edu/ml

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

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

[167] X. Kuai, H. Sun, S. Zhou, and E. Cheng, "Impulsive Noise Mitigation in Underwater Acoustic OFDM Systems," *IEEE Transactions on Vehicular Technology*, vol. 65, pp. 8190–8202, Oct. 2016.

[168] T. Suzuki, H. M. Tran, and T. Wada, "An underwater acoustic OFDM communication system with shrimp (impulsive) noise cancelling," in *2014 International Conference on Computing, Management and Telecommunications (ComManTel)*, pp. 152–156, Apr. 2014.

[169] S. Banerjee, M. Agrawal, Underwater acoustic communication in the presence of heavy-tailed impulsive noise with bi-parameter cauchy-gaussian mixture model, Ocean Electronics (SYMPOL), 2013 (2013) 1–7.

[170] S. Banerjee, M. Agrawal, Underwater acoustic noise with generalized gaussian statistics: Effects on error performance, OCEANS - Bergen, 2013 MTS/IEEE (2013) 1–8.

[171] S. Banerjee, M. Agrawal, On the performance of underwater communication system in noise with gaussian mixture statistics, 2014 Twentieth National Conference on Communications (NCC) (2014) 1–6.

[172] D. Zha, T. Qiu, Underwater sources location in non-gaussian impulsive noise environments, Digital Signal Processing 16 (2) (2006) 149–163.

[173] J. He, Z. Liu, Underwater acoustic azimuth and elevation angle estimation using spatial invariance of two identically oriented vector hydrophones at unknown locations in impulsive noise, Digit. Signal Process. 19 (3) (2009) 452–462.

[174] J. Zhang, Y. Pang, Pipelined robust m-estimate adaptive second-order volterra filter against impulsive noise, Digital Signal Processing 26 (2014) 71–80.

[175] J. A. Hildebrand, Anthropogenic and natural sources of ambient noise in the ocean, Inter-Research, Marine Ecology Progress Series (MEPS) 395 (2009) 5–20.

[176] M. U. Otaru, A. Zerguine, L. Cheded, Channel equalization using simplified least mean-fourth algorithm, Digital Signal Processing 21 (3) (2011) 447–465.

[177] A. Zerguine, Convergence and steady-state analysis of the normalized least mean fourth algorithm, Digit. Signal Process. 17 (1) (2007) 17–31.

[178] M. O. Sayin, N. D. Vanli, S. S. Kozat, A novel family of adaptive filtering algorithms based on the logarithmic cost, Signal Processing, IEEE Transactions on 62 (17) (2014) 4411 – 4424.

[179] J. A. Chambers, O. Tanrikulu, A. G. Constantinides, Least mean mixed-norm adaptive filtering, Electron. Lett. 30 (19) (1994) 1574–1575.

[180] J. Chambers, A. Avlonitis, A robust mixed-norm adaptive filter algorithm, IEEE Signal Process. Lett. 4 (2) (1997) 46–48.

[181] J. Arenas-Garcia, V. Gomez-Verdejo, A. R. Figueiras-Vidal, New algorithms for improved adaptive convex combination of LMS transversal filters, IEEE Trans. Instrumentation and Measurement 54 (6) (2005) 2239–2249.

[182] M. T. M. Silva, V. Nascimento, Improving the tracking capability of adaptive filters via convex combination, IEEE Trans. Signal Process. 56 (7) (2008) 3137–3149.

[183] K. Pelekanakis, M. Chitre, Adaptive sparse channel estimation under symmetric alpha-stable noise, IEEE Transactions on Wireless Communications 13 (6) (2014) 3183–3195.

[184] K. Pelekanakis, M. Chitre, Robust equalization of mobile underwater acoustic channels, IEEE Journal of Oceanic Engineering 40 (4) (2015) 775–784.

[185] S. Wang, Y. Zheng, S. Duan, L. Wang, C. K. Tse, A class of improved least sum of exponentials algorithms, Signal Processing 128 (2016) 340 – 349.

[186] W. Ma, B. Chen, H. Qu, J. Zhao, Sparse least mean p-power algorithms for channel estimation in the presence of impulsive noise, Signal, Image and Video Processing 10 (3) (2016) 503–510.

[187] C. van den Bos, M. H. L. Ksuwenhoven, W. A. Serdijn, Effect of smooth nonlinear distortion on ofdm symbol error rate, IEEE Transactions on Communications 49 (9) (2001) 1510–1514.

[188] N. R. Yousef, A. H. Sayed, Ability of adaptive filters to track carrier offsets and channel nonstationarities, IEEE Transactions on Signal Processing 50 (7) (2002) 1533–1544.

[189] F. Qu, L. Yang, Basis expansion model for underwater acoustic channels?, in: OCEANS 2008, 2008, pp. 1–7.

[190] C. P. Shah, C. C. Tsimenidis, B. S. Sharif, J. A. Neasham, Low-complexity iterative receiver structure for time-varying frequency-selective shallow underwater acoustic channels using bicm-id: Design and experimental results, Oceanic Engineering, IEEE Journal of 36 (3) (2011) 406–421.

[191] X. Wang, H. V. Poor, Joint channel estimation and symbol detection in Rayleigh flat-fading channels with impulsive noise, IEEE Comm. Lett. 1 (1) (1997) 19–21.

[192] S.-C. Chan, Y.-X. Zou, A recursive least M-estimate algorithm for robust adaptive filtering in impulsive noise: fast algorithm and convergence performance analysis, IEEE Trans. Signal Process. 52 (4) (2004) 975–991.

[193] T. Y. Al-Naffouri, A. H. Sayed, Transient analysis of adaptive filters with error nonlinearities, IEEE Trans. Signal Process. 51 (3) (2003) 653–663.

[194] R. J. Turyn, Sequences with small correlation, In Error Correcting Codes: Proceedings of a Symposium (1968) 195–228.

[195] P. Qarabaqi, M. Stojanovic, Statistical characterization and computationally efficient modeling of a class of underwater acoustic communication channels, IEEE J. Ocean. Eng. 38 (4) (2013) 701–717.

[196] P. C. Etter, Underwater Acoustic Modeling and Simulation, 4th Edition, 2013.

Appendix A

Approximate Mahalanobis distance

For the quadratic distance we mentioned in (2.8), Mahalanobis distance is a good choice since it incorporates the covariance of the data along with the euclidean distance [28]. However, since we need to measure the distance between $D-$dimensional $\boldsymbol{x}[n]$ and $d-$dimensional $\Re_\eta[n]$, we use approximate Mahalanobis distance [27, 28, 38]. The Mahalanobis distance measures the quadratic distance between a vector $\boldsymbol{x}$ and region $\Re$, containing data with mean $\boldsymbol{c} = \mathbb{E}(\boldsymbol{x})$ and covariance $\boldsymbol{\Sigma} = \mathbb{E}\left((\boldsymbol{x}-\boldsymbol{c})(\boldsymbol{x}-\boldsymbol{c})^T\right)$, and is given by:

$$d_M(\boldsymbol{x}, \Re) \triangleq (\boldsymbol{x}-\boldsymbol{c})^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{x}-\boldsymbol{c}). \tag{A.1}$$

Here we dropped the subscripts, η and n for simplicity. By eigen decomposition of $\boldsymbol{\Sigma}$, we define:

$$\boldsymbol{\Sigma} \triangleq \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^T, \tag{A.2}$$

where $\boldsymbol{\Lambda} = \text{diag}(\lambda_1, \lambda_2, ..., \lambda_D)$ and $\lambda_1 \geq ... \geq \lambda_D$ are eigen-values of $\boldsymbol{\Sigma}$. $\boldsymbol{Q}$ is the eigen-vector matrix and specifies the basis for subset closest to $\boldsymbol{x}[n]$. We then use only d largest eigen-values and their corresponding eigen vector matrix to approximate the covariance matrix, $\boldsymbol{\Sigma} \approx \boldsymbol{Q}_1\boldsymbol{\Lambda}_1\boldsymbol{Q}_1^T + \delta\|\boldsymbol{Q}_2\|^2$. The Mahalanobis distance is approximated as:

$$d_M(\boldsymbol{x}, \Re) \approx (\boldsymbol{x}-\boldsymbol{c})^T \boldsymbol{Q}_1 \boldsymbol{\Lambda}_1^{-1} \boldsymbol{Q}_1^T (\boldsymbol{x}-\boldsymbol{c}) + \delta^{-1}\|\boldsymbol{Q}_2^T(\boldsymbol{x}-\boldsymbol{c})\|^2. \tag{A.3}$$

We use the following scaled version of the above definition for the distance measure to determine the optimal node η^* in (2.8),

$$D_M(\boldsymbol{x}, \Re) \triangleq \delta(\boldsymbol{x}-\boldsymbol{c})^T \boldsymbol{Q}_1 \boldsymbol{\Lambda}_1^{-1} \boldsymbol{Q}_1^T (\boldsymbol{x}-\boldsymbol{c}) + \|\boldsymbol{Q}_2^T(\boldsymbol{x}-\boldsymbol{c})\|^2. \tag{A.4}$$